Master of Science thesis – Fighting Spam

Lasse Lerkenfeld Jensen, c991079

Jacob Irsberg, c991342

Vejleder: Christian Damsgaard Jensen, cdj@imm.dtu.dk

Indholdsfortegnelse

1 Indledning 9
2 Spammail 11
2.1 Definition af spammail 11
2.2 Konsekvenser af spammail 12
3 Spambekæmpelse 15
3.1 Filtrering 15
3.1.1 Filtreringsmetoder 15
3.1.1.a Blacklisting 15
3.1.1.b Whitelists 15
3.1.1.c Intelligente og naive filtre 16
3.1.1.d Duplikat detektion ved samarbejde 16
3.1.1.e Kombinationer 17
3.1.2 Begrænsninger ved filtrering 18
3.2 Betaling 19
3.2.1 Ressourcebetaling 19
3.2.2 Monetær betaling 20
3.3 Autentifikation 21
3.3.1 Afsenderautentifikation 21
3.3.2 Tillid 23
3.4 Challenge/Response 24
3.5 Channels 24
3.6 Lovgivning 25
3.7 Opsummering 26
4 Kryptografi 29
4.1 Hash-funktioner 29
4.2 Symmetrisk kryptografi 30
4.3 Asymmetrisk kryptografi 34
4.4 Blinde signaturer i RSA 36
4.5 Opsummering 36
5 Analyse 37
5.1 Udviklingsmodel 37
5.2 Grundlæggende ide 38
5.3 Sideeffekter 38
5.3.1 Møntjægere 38
5.3.1.a Channels 39
5.3.2 Brugeradfærd 39
5.4 Betalingssystemer 40
5.4.1 Double spending 41
5.4.2 Anonymitet 42
5.4.2.a Secret-splitting 42
5.4.2.b Opdeling af viden 45
5.5 Kravspecifikation 47
5.6 Trusler 49

5.6.1 Økonomisk misbrug 49
5.6.2 Netværkstrusler 50
5.6.3 Fysiske trusler 51
6 Design 53
6.1 Systemstruktur 53
6.1.1 Infrastruktur 53
6.2 Autentifikation 54
6.2.1 Oprettelse 54
6.2.1.a Traditionelt certifikatsystem 55
6.2.1.b SpamCash certifikatsystem 55
6.2.2 Pengeoverførsler 56
6.3 Design af elektroniske penge 56
6.3.1 Betaling 57
6.3.2 Møntstruktur 58
6.3.2.a Transaktionslisten 59
6.3.3 Økonomisk misbrug 61
6.3.3.a Modifikation 61
6.3.3.b Tyveri 62
6.4 Blacklisting 63
6.4.1 Opdatering af blacklist 64
6.4.2 Detektion af kopier 66
6.4.2.a Detektion i pengeserver 66
6.4.2.b Detektion i klient 66
6.5 Anonymitet 67
6.5.1 Anonymitet overfor systemet 67
6.5.2 Anonymitet overfor andre brugere 68
6.6 Use-cases 69
6.6.1 Use Case 1a: Start program 69
6.6.2 Use Case 1b: Start program 70
6.6.3 Use Case 2: Konfigurer program 70
6.6.4 Use Case 3: Penge-status 70
6.6.5 Use Case 4a: Afsend email 71
6.6.6 Use Case 4b: Afsend email 71
6.6.7 Use Case 5: Modtag email 71
6.6.8 Use Case 6: Indkasser ”SpamCash” 72
6.6.9 Use Case 7: Hæv Penge 72
6.6.10 Use Case 8: Indløs penge 73
6.6.11 Use Case 9: Tilføj email til whitelist 73
6.6.12 Use Case 10: Fjern email fra whitelist 73
6.6.13 Use Case 11: Luk program 74
6.7 Sikkerhed 74
6.7.1 Lokal sikkerhed 74
6.7.2 Netværkssikkerhed 75
6.7.3 Nøglestørrelser 75
6.8 Kommunikationsprotokoller og serverdesign 77
6.8.1 Servere 77

6.8.2 Blacklistserver 78
6.8.3 Verifikationsserver 80
6.8.4 Pengeserver 81
6.8.4.a Login 81
6.8.4.b Hæv penge 82
6.8.4.c Indløsning af e-mønter 84
6.8.5 Registrationserver 85
6.9 Klientdesign 88
6.9.1 Protokoller 88
6.9.1.a Afsendelse af email 88
6.9.1.b Modtagelse af email 89
6.9.2 Klientens struktur 93
6.9.3 Klientens funktionalitet 95
6.9.3.a Registrering (Use Case 1) 96
6.9.3.b Se pengestatus (Use Case 3) 97
6.9.3.c Send email (Use Case 4) 98
6.9.3.d Modtagelse af email (Use Case 5) 99
6.9.3.e Indkasser “SpamCash” (Use Case 6) 102
6.9.3.f Hæv penge (Use Case 7) 102
6.9.3.g Indsæt penge (Use Case 8) 103
6.10 Trusselshåndtering 104
6.10.1 Økonomisk misbrug 104
6.10.2 Netværkstrusler 105
6.10.3 Fysiske trusler 107
7 Implementation 109
7.1 Programoversigt 109
7.1.1 Tredjeparts-software 110
7.1.1.a BouncyCastle 110
7.1.1.b Logi 110
7.1.1.c Javamail 111
7.1.1.d OpenSSL 111
7.1.1.e Base64 112
7.2 Begrænsninger 112
7.2.1 Sletning af certifikater 112
7.2.2 Justering af mønters værdi 112
7.2.3 Emailadresser under oprettelse 112
7.2.4 Ejerskab af emailadresser under oprettelse 112
7.2.5 Afsendelse af emails til flere modtagere 113
7.2.6 Stikprøvekontrol 113
7.3 Udspecificering af implementeringsdetaljer 114
7.3.1 Proxy 114
7.3.2 Emailbehandling 115
7.3.3 Kryptering 115
7.3.4 Session 116
7.3.5 Nedarvning i servere 116
7.3.6 GUI 117

7.3.7 Log ... 118
7.3.8 Konfiguration ... 119
7.3.9 Blinde signaturer ... 120
7.3.10 Filer ... 122
8 Evaluering ... 123
8.1 Afprøvning ... 123
8.1.1 Testmetode ... 123
8.1.2 Funktionel test ... 123
8.1.2.a Klient ... 123
8.1.2.b Servere ... 129
8.1.3 Test af sikkerhed ... 134
8.2 Justering af systemet ... 134
8.2.1 Serverparametre ... 134
8.2.2 E-penge parametre ... 135
8.2.2.a Oprettelsesgebyr og stikprøvekontrol ... 137
8.3 Ydeevne ... 140
8.3.1 Klient ... 140
8.3.1.a Emailhåndtering ... 141
8.3.1.b Oprettelse ... 142
8.3.1.c Opkrævning af e-mønter ... 143
8.3.1.d Udstedelse af e-mønter ... 143
8.3.1.e Indløsning af e-mønter ... 143
8.3.2 Blacklistserver ... 143
8.3.3 Skalerbarhed ... 145
9 Konklusion ... 147
9.1 SpamCash ... 147
9.2 Resultater ... 147
9.3 Fremtidigt arbejde ... 148
Appendiks A Installationsvejledning ... 151
Appendiks B Brugsvejledning ... 157
Appendiks C Litteratur ... 171
Appendiks D Diagrammer ... 174
Appendiks E Logfil ... 189
Appendiks F Maple kode ... 191
Appendiks G Matlab kode ... 193
Appendiks H JAVA Kode ... 195
10 Kildekode ... 199
10.1 Client ... 199
10.1.1 Control ... 199
10.1.2 CurrencyExchangeClient ... 214
10.1.3 Whitelist ... 217
10.1.4 GUI ... 218
10.1.4.a AmountPanel ... 218
10.1.4.b ConfigurationPanel ... 221
10.1.4.c Definitions ... 223
10.1.4.d DepositPanel ... 235

10.1.4.e ErrorDialog 237
10.1.4.f FormPanel 238
10.1.4.g ImageButton 239
10.1.4.h MainFrame 242
10.1.4.i OpenSafeDialog 247
10.1.4.j Progress 251
10.1.4.k RegistrationFrame 253
10.1.4.l SpamPanel 256
10.1.4.m SplashScreen 258
10.1.4.n WhitelistPanel 259
10.1.4.o WithdrawPanel 261
10.1.5 MailHandler 262
10.1.5.a CoinHeader 262
10.1.5.b IncomingMailProcessor 263
10.1.5.c MailFunctions 267
10.1.5.d NotificationHeader 268
10.1.5.e OutgoingMailProcessor 268
10.1.6 Proxy 270
10.1.6.a AbstractBlokingProxy 270
10.1.6.b AbstractNonBlokingProxy 273
10.1.6.c IMAPProxy 276
10.1.6.d NonBlokingProxyConnection 278
10.1.6.e POP3Proxy 283
10.1.6.f SMTPProxy 295
10.1.7 Safe 297
10.1.7.a Safe 297
10.1.7.b SafeHandler 302
10.2 Currency 309
10.2.1 BlindedCoin 309
10.2.2 Coin 311
10.2.3 EncryptedTransactionlist 316
10.2.4 Transactionlist 318
10.2.5 TransactionlistElement 318
10.2.6 UnencryptedTransactionlist 320
10.3 NetworkPackets 323
10.3.1 ActivationCompletedPacket 323
10.3.2 ActivationPacket 324
10.3.3 AmountDepositedPacket 324
10.3.4 BlacklistPacket 325
10.3.5 CoinSignaturePacket 325
10.3.6 CryptoPacket 325
10.3.7 CurrencyBlindingFactorPacket 326
10.3.8 CurrencyExchangeRequestPacket 327
10.3.9 DepositRequestPacket 328
10.3.10 GetBlacklistPacket 328
10.3.11 LogonPacket 329

10.3.12 NetPacket 329
10.3.13 RegistrationInfoPacket 329
10.3.14 RegistrationRequestPacket 330
10.3.15 ReportCoinsPacket 330
10.3.16 RequestBlindingFactorsPacket 330
10.3.17 StatusPacket 331
10.3.18 VerificationPacket 331
10.4 Servers 333
10.4.1 AbstractServer 333
10.4.2 CommunicationThread 334
10.4.3 ReceiveThread 334
10.4.4 BLS 335
10.4.4.a Blacklist 335
10.4.4.b BlacklistServer 337
10.4.4.c BLSCommunicationThread 340
10.4.5 CES 341
10.4.5.a Accounts 341
10.4.5.b BankAccount 343
10.4.5.c CESCommunicationThread 344
10.4.5.d CESConfiguration 346
10.4.5.e CoinSet 346
10.4.5.f CurrencyExchangeServer 346
10.4.6 RS 353
10.4.6.a PendingAccount 353
10.4.6.b PendingAccounts 353
10.4.6.c RegistrationServer 354
10.4.6.d RSCommunicationThread 357
10.4.7 VS 360
10.4.7.a VerificationServer 360
10.4.7.b VSCommunicationThread 362
10.5 Utilities 363
10.5.1 Base64 363
10.5.2 Config 384
10.5.3 Cryptotools 386
10.5.4 Debugger 400
10.5.5 FileHandler 403
10.5.6 MailInformation 406
10.5.7 MimeTools 407
10.5.8 PackageQueue 414
10.5.9 ProxyConfiguration 415
10.5.10 Session 415
10.5.11 Test 418

Abstract

Up until now spam email has become more and more of a problem for almost all users of the global internet mail system. It costs valuable time and resources to deal with and clogs up our inboxes increasingly.

Most solutions seen today try to deal with the problem through filtering and blacklisting. All but a few of them with limited success since spammers exploit holes in the filtering schemes and the email system itself is inherently insecure, making effective blacklisting of spammers impossible.

In this project we aim to stop spamming by removing the financial incentive for sending spam all together. The solution is based on the existing mail system and works by financially penalizing those who send spam while rewarding those who receive it. Briefly explained this is achieved by requiring a small payment sent along with each email. This payment is returned to the sender if the receiver does not categorize the email as spam. This means that ideally only spammers will pay for using the mail system while for those of us who use it as intended; it will remain cost-free.

1 Indledning

Da email-systemet blev designet, var det en grundlæggende antagelse, at systemet ikke ville blive misbrugt. Dette er en af årsagerne til, at spammail er et stadigt voksende problem, og i dag er det de færreste, der går fri af spammail i deres indbakke. Ved spammail forstås normalt uønsket email i form af f.eks. uopfordret reklame. Da afsendelse af email er forbundet med en meget lille udgift, og afsender ikke umiddelbart kan identificeres, vil sådanne reklamer ofte afsendes i stort antal til gene for de mange modtagere.

Spamproblemet har længe været stort, og derfor tilbydes i dag en række forskellige løsninger på dette. De fleste løsninger bygger på en eller anden form for filtrering af emails, inden de når modtagers indbakke. Problemet er dog, at selv de bedste filtre ikke fanger alle spammails, og der er altid risiko for, at en legitim email bliver fanget i filtret, og dermed ikke når frem til modtager. Såkaldte 'challenge-response'-løsninger sigter på at stoppe spammails ved at sende en udfordring (*challenge*) tilbage til afsender, som skal løse denne for at få sin email leveret. Dette vil i de fleste tilfælde være en uoverkommelig opgave for en spammer, som typisk sender tusindvis af emails. Denne løsning går dog ud over funktionaliteten af emailsystemet, da dette i stigende grad benyttes til afsendelse af maskinelt genererede emails (f.eks. ordrebekræftelser og lign.). Maskiner som har afsendt disse, vil ikke kunne svare på en sådan *challenge*. Af andre løsninger kan nævnes såkaldte 'proof-of-work'-løsninger. I disse skal afsenders maskine udføre f.eks. en beregningstung og dermed tidskrævende opgave og medsende svaret til modtager, for at få emailen leveret til dennes indbakke. Denne type løsning kan typisk gøre det mindre attraktivt at spamme, men ikke eliminere problemet.

I lyset af ovennævnte problemer med de gængse løsninger har vi konstrueret en betalingsbaseret løsning. Af disse findes nogle få etablerede allerede. Essensen i betalingsbaserede løsninger er at fjerne incitamentet for at sende spammail, hvilket opnås ved at straffe spammere økonomisk, mens det for andre forbliver gratis at sende email. De eksisterende løsninger har dog alle en række ulemper. I nogle kræves det, at der er en tredje part (f.eks. en betalingsserver) involveret ved hver emailoverførsel (*online*-system). Andre har problemer hvad angår *scalability* (egner sig ikke som globale løsninger) og anonymitet. I vores løsning har vi tilstræbt at eliminere disse ulemper ved at designe og implementere et *offline* og skalerbart system, hvor anonymiteten for den enkelte bruger samtidig er bevaret. At opnå skalerbarhed af systemet lykkedes kun delvist, da kravet om et offline system betød, at vi ikke kunne undvære en central enhed i systemet. Anonymitet i systemet blev sikret, dog på bekostning af ressourcekrævende udstedelse af elektroniske penge. Ved at lade systemet bygge på det eksisterende emailsystem og eliminere nødvendigheden for traditionelle filtreringsmekanismer, har vi dog konstrueret et system, hvor pålideligheden (*reliability*) er høj. I systemet er nemlig ingen risiko for, at legitime emails fejlagtigt frasorteres i filtreringsprocessen. Dette er efter vores opfattelse en meget vigtig egenskab ved en spamløsning, hvorfor vi mener, at der ligger et potentiale i systemet til trods for de ulemper, som dette besidder. Vi har i den afsluttende del af rapporten vurderet, hvorvidt ulemperne opvejer fordelene ved systemet. Til

syvende og sidst er det dog op til de potentielle brugere af systemet at afgøre, om dette er tilfældet.

Kapitlerne i denne rapport er grundlæggende opdelt i tre dele. Først diskuteres spamproblemet, eksisterende spamløsninger præsenteres og problemstillingen analyseres (kapitel 2, 3, 4 og 5). Derefter vil design og implementation af den udviklede løsning blive beskrevet (kapitel 6 og 7). Til sidst følger en evaluering af det implementerede system og der konkluderes på opnåede resultater og projektet som helhed (kapitel 8 og 9).

2 Spammail

I det følgende vil først blive fastlagt en definition af spammail, der kan benyttes i resten af denne rapport. Derefter vil det kort blive gennemgået, hvor stort spamproblemet er, og har været gennem tiden.

2.1 Definition af spammail

Det har altid været et stort problem at finde en brugbar definition af spammail. Det er dog vigtigt at dette gøres, så præcis de emails vi gerne vil undgå kan identificeres. Eksempler på definitioner af spammails er

"Unsolicited commercial e-mail sent to advertise a product or a service."
- Brad Smith, Chief Counsel, Microsoft

"An unwanted automated message."
- Joe Barrett, Senior Vice President, AOL

Den generelle opfattelse af spammails er, at disse enten er reklamemails fra firmaer, emails der er sendt uopfordret fra ukendte afsendere, eller i det hele taget emails der på den ene eller den anden måde er generende. Dette er meget vage definitioner, som ikke er direkte anvendelige i tekniske sammenhænge. Derfor er det nødvendigt med en mere præcis definition. Reklamemails der sendes uopfordret og til et stort antal modtagere, altså kommerciel spammail, vil de fleste opfatte som spam, men det er ikke nødvendigvis alle, der har denne opfattelse. Sendes f.eks. en reklame for viagra, vil de fleste gerne undgå denne, men der vil også findes personer, der gerne vil have denne reklame. Det er altså ikke entydigt, hvorvidt en email betragtes som spam eller ej. Dette er også grunden til at kommerciel spammail sendes i enorme mængder i dag. Nogle modtagere *vil* købe produkterne og dermed skabe den nødvendige profit både for spammerne og de involverede firmaer. En undersøgelse foretaget i USA viser, at i 2004 købte 4% af modtagerne et produkt, fra en kommerciel spammail [49].

Spammail kan også karakteriseres ud fra mere dybdegående undersøgelser. F.eks. ved at afsender har tilegnet sig modtageradresserne på uautoriseret vis, eller ved at afsenderadressen ikke eksisterer. Om en definition overhovedet er anvendelig afhænger bl.a. af den løsning denne skal indgå i. F.eks. vil en individuel definition ikke umiddelbart være forenelig med en filtreringsløsning på serverniveau, som altså vil påvirke mange brugere.

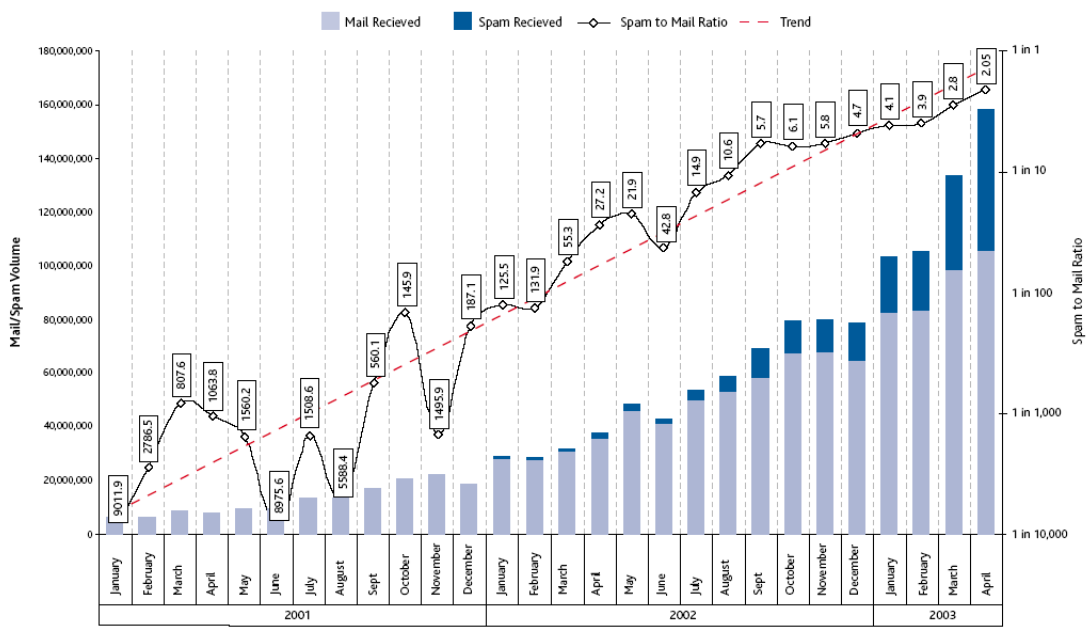
Generelt er det individuelt, hvad der opfattes som spam. Det er altså modtageren, der må betragtes som værende den bedste til at afgøre, hvorvidt en modtaget email er spam. Vi vil derfor benytte følgende simple definition af spammails:

- E-mail, der er uønskede af modtageren

Denne definition kan virke en smule vag, da den indbefatter mange forskellige typer emails, sendt af forskellige afsendere. Samtidig er den dog stærk, da en bruger der går fri for netop disse emails, må antages at være fuldstændig tilfreds. Målsætningen er altså, at brugeren ikke modtager uønskede emails. Dette er naturligvis kun muligt, hvis brugeren ikke alt for ofte ændrer sin opfattelse af uønsket mail. Det vil senere i denne rapport fremgå, hvordan denne definition naturligt kan indgå i systemets design.

2.2 Konsekvenser af spammail

Fra at være et stort set ikke-eksisterende problem for blot få år siden, er andelen af spammails i forhold til legitime mails nærmest eksploderet på det sidste (se Figur 1). Den generelle tendens er, at andelen af spammails i forhold til legitime mails vokser eksponentielt [1], og der er umiddelbart ingen udsigt til, at denne udvikling stopper. Som det ses på Figur 1 kunne omtrent en tredjedel af alle emails kategoriseres som spam ved udgangen af 2003. Ved udgangen af 2004 var denne andel vokset til to tredjedele [1].



Figur 1 – Messagelabs analyse

Den enorme mængde af spam i omløb har store konsekvenser på både kortere og længere sigt. Medarbejdere på stort set alle virksomheder, som modtager email bruger en stadig større del af deres arbejdstid på at gennemgå deres daglige post. Ifølge Ferris Research (december 2003) [1] bruger hver medarbejder gennemsnitligt 4 sekunder på at identificere og slette en spammail. Dette lyder umiddelbart ikke af meget, men modtager en virksomhed med f.eks. 50 medarbejdere blot 70 spammails om ugen pr. medarbejder, udgør den årlige samlede spildtid over 200 timer for virksomheden. Udover at være til irritation for den enkelte medarbejder er spammails altså også en betydelig udgift for den

enkelte virksomhed. Ifølge Ferris Research (Marts 2003) [2] var spammail årsag til en samlet udgift på 10.000.000.000 $ for virksomheder i USA alene i 2003.

Den generelle opfattelse blandt private brugere af email-systemet er, at det er internetudbydernes opgave at løse spamproblemet (Ifølge Gartner Group er det 74 %, der mener dette [6]). Dette har været med til at tvinge de enkelte internetudbydere til at installere filtre på deres mailservere i et forsøg på at komme størstedelen af spammails til livs. Problemet er dog, at spammere svarer igen ved at øge antallet af afsendte spammails. Derudover camouflerer de deres spammails, så de mere og mere ligner legitime emails. Dette gør det praktisk taget umuligt at undgå, at filtrene ind i mellem fanger legitime emails (*false positives*). I konkrete tilfælde kan det betyde at f.eks. vigtige forretningsmails ikke når frem, hvilket igen kan betyde større eller mindre tab for den pågældende virksomhed. På længere sigt kan konsekvensen være, at den generelle opfattelse af emailsystemet som værende et pålideligt kommunikationsmiddel lider et knæk. I værste fald kan dette være begyndelsen til enden på emailsystemet, som vi kender det. Et andet grundlæggende problemer er, at spammere ofte sender via såkaldte ’open-relay’ mailservere. En ’open-relay’ server er en mailserver, der ikke kræver nogen form for autentifikation ved afsendelse af email. Enhver kan således kontakte en sådan server og benytte denne til afsendelse eller videresendelse af email. Denne type servere findes ikke hos internetudbydere, da disse i reglen konfigurerer deres mailservere til kun at acceptere bestemte afsenderadresser eller til at kræve at afsenders IP-adresse tilhører et bestemt adresseområde. ’Open-relay’-servere vil dog ofte kunne findes hos private brugere med faste internetforbindelser. Den store udbredelse open-relay-servere gør det til noget nær en umulig opgave at forhindre afsendelse af spam. Derudover holdes spammerens identitet skjult overfor omverdenen, ved at benytte sådanne servere.

Nu er problemerne med spammail blevet diskuteret og en definition er blevet opstillet. I det følgende vil eksisterende løsninger blive beskrevet. De forsøger alle at begrænse det samme spamproblem, men tager udgangspunkt i forskellige spamdefinitioner. Flere af disse definitioner kan være problematiske at acceptere for brugerne, idet deres personlige opfattelse ikke altid afspejles. Styrken ved spamdefinitionen i afsnit 2.1 er netop, at den bygger på, at det er individuelt hvad der opfattes som spammail.

3 Spambekæmpelse

I dette afsnit vil vi give et overblik over eksisterende løsningsmetoder på spamproblemet, samt deres fordele og ulemper.

3.1 Filtrering

Forskning indenfor emailfiltrering har været udpræget de sidste mange år. Det er også den mest anvendte metode i praksis til spambekæmpelse, både til privat og kommerciel brug. Grundlæggende kan denne metode beskrives på følgende måde.

Et filter dannes ud fra et eller flere *datasæt*. Et datasæt kan indeholde f.eks. email-adresser, IP-adresser, DNS-adresser eller nøgleord. Herefter anvendes filteret hver gang en email modtages til at afgøre, om denne skal afvises eller ej. Filtreringen kan foregå mange steder på emailens vej, f.eks. i netværkskomponenter, på mailserveren eller lokalt på modtagerens personlige computer. Filteret bliver ofte opdateret løbende, når der forekommer ændringer i de datasæt, der danner grundlag for filteret. I de følgende afsnit vil de mest populære filtreringsmetoder blive diskuteret.

3.1.1 Filtreringsmetoder

3.1.1.a Blacklisting

Filtrering af email kan foregå ved at benytte et simpelt filter i form af en *blacklist.* Modtages en email, hvor afsenders emailadresse findes på blacklisten, afvises denne, ellers ikke. Listen kan også indeholde hele domæner eller IP-adresser **[**3**]**. Derudover giver nogle løsninger mulighed for at straffe mailservere, der sender spammails, f.eks. ved kontinuerligt at sende krævende forspørgsler til denne [4]. Der eksisterer også løsninger, hvor flere forskellige lister kombineres for at opnå et bedre resultat [5]. Fælles for disse systemer er, at de er effektive. Brugerne af mange af disse systemer vil opleve en betydelig reduktion i mængden af spam, og en spammer vil ikke kunne bruge den samme mailserver to gange. Problemet ved at opretholde en blacklist er risikoen for at tilføje legitime brugere/servere til listen (*false positives*). Dette er svært at undgå og kan være en relativt høj pris at betale. At blackliste *open-relay* servere kan også være problematisk, idet man på denne måde i mange tilfælde vil straffe offeret og ikke spammeren. Tilsvarende kan en spammer få et helt domæne blacklistet, hvilket vil skabe problemer for dets øvrige brugere. Desuden er det problematisk at anvende blacklist-løsninger i meget stor skala, da dem der styrer en sådan liste vil være i besiddelse af et magtfuldt globalt censurapparat. Dette er ikke umiddelbart noget, der normalt falder i god jord blandt internettets brugere.

3.1.1.b Whitelists

Ideen med en whitelist i emailsammenhæng er, at der udelukkende modtages email fra emailadresser, som befinder sig på denne. Alle andre emails slettes. Fordelen ved at benytte en whitelist som filtreringsmekanisme er, at spam effektivt holdes ude. Ulempen

er dog, at ønskede uopfordrede emails også holdes ude, hvilket gør, at systemer, der benytter sig af whitelists ofte vil være meget lukkede. Et sted, hvor whitelists næsten altid med fordel kan benyttes, er i *kombination* med andre spamløsninger. Whitelists kan nemlig generelt benyttes til at skabe en fornuftig overgangsfase, hvor kun få brugere endnu er med i spamløsningen. Hermed gives der mulighed for at modtage email fra brugere, som endnu ikke er en del af systemet, ved at placere disse på whitelisten.

3.1.1.c Intelligente og naive filtre

I stort set enhver email-klient er det muligt at blokere for emails fra bestemte afsendere eller for emails, som indeholder bestemte ord eller sætninger. Denne simple regelbaserede form for filtrering kaldes normalt mønstergenkendelses-filtrering (*pattern-matching*). Filtreringsmetoden er meget ineffektiv, da spammere med vilje introducerer f.eks. stavefejl i deres spammails for at slippe igennem disse filtre. Blokering af afsenderadresser har stort set ingen effekt, da spammere næsten altid benytter falske afsenderadresser, og sjældent den samme i to forskellige spammails. Udover at være ineffektivt er der risiko for, at filteret fanger og blokerer legitime emails. Man kunne f.eks. forestille sig at man for at slippe for spam med erotisk indhold har tilføjet ordet 'anal' til mængden af strenge filteret skal reagere på. Dette vil dog betyde at udover spammails indeholdende ordet 'anal', vil alle legitime mails, som f.eks. indeholder ordene 'analyse' eller 'kanal', pludselig *også* blive fanget af filteret (idet 'anal' indgår i disse ord). For at opnå en effektiv filtrering, som kun i få tilfælde fejlagtigt vil klassificere en legitim mail som spam, er det derfor nødvendigt at introducere mere avancerede filtre.

Heuristisk Filtrering er baseret på mønstergenkendelses-filtrering hvor reglerne, for hvornår en email betragtes som spam, er dannet gennem lang tids erfaring. Filtreringen foregår så ved at evaluere hver ny email på baggrund af det tilpassede regelsæt og give meddelelsen en 'score' baseret på statistiske beregninger, som herefter bruges til at afgøre, om emailen skal kategoriseres som værende spam eller ej. Da regelsættet i den heuristiske filtrering er dannet ud fra analyser af mange tusinder af emails, såvel legitime som spammails, er det relativt pålideligt.

En anden type avanceret filtrering er den såkaldte Bayesiske filtrering (*Bayesian*) [7]. Her analyseres først et stort antal af hhv. legitime emails og spammails. Hele emailen (header, emnelinie, domænenavn, etc.) analyseres i hvert tilfælde, og der tildeles 'spam-sandsynligheder' til hvert eneste ord, domæne eller anden *token* i emailen. Disse sandsynligheder kan så efterfølgende benyttes til at afgøre, hvorvidt en modtaget email kan klassificeres som spam eller ej. Derudover kan filteret 'oplæres' af den enkelte bruger, ved at denne tilføjer nye kendte spammails til filteret. Hvis en bruger opdager at filteret fejlagtigt har klassificeret en email som spam, kan denne rette fejlen og filteret vil så 'lære' af fejlen, og på den måde blive bedre.

3.1.1.d Duplikat detektion ved samarbejde

At danne filtre ud fra tidligere modtagne emails er effektivt, som beskrevet i foregående afsnit. Problemet kan dog være, at det der karakteriserer spammails, kan ændres fra dag til dag. Her vil selv et intelligent filter komme til kort, da et sådant filter ikke vil kunne

fange de allerførste spammails af en ny type. Filteret skal lære de nye mails at kende, det skal altså opdateres, før dette igen kan filtrere fornuftigt. Der findes en anden indgangsvinkel til filtrering, der netop fokuserer på denne opdateringshastighed, nemlig *collaborative filtering*. Her sigtes på at begrænse den type spammails, der udsendes i mange lignende kopier, og disse udgør da også langt størstedelen. Teknikken bygger ofte på grundtanken om, at den bedste til at filtrere spam er et menneske. Denne antagelse virker da også rimelig, eftersom maskinelle forsøg har haft begrænset succes. Skal en person filtrere al sin email manuelt, vil dette optage en stor del af dennes tid. Derfor er ideen, at koordinere filtreringen på en sådan måde, at hver enkelt bruger kun skal filtrere en lille del af det samlede antal spammails. Hver bruger filtrerer manuelt spam i sin indbakke. De spammails der sendes til mere end én, vil blive filtreret af flere brugere. Når tilstrækkeligt mange brugere har frasorteret en spammail, vil denne automatisk blive frasorteret hos de tusindvis af brugere der endnu ikke har fået den. På denne måde vil hver enkelt bruger kun modtage en mindre mængde spam.

I sådanne systemer [8, 9, 10] benyttes en checksum til at identificere en spammail. En mail kan gennemgå transformationer før denne checksum beregnes, så filteret ikke kan omgås ved små variationer som f.eks. store og små bogstaver. Derudover kan benyttes statistiske sammenligningsfunktioner, når en mail skal filtreres. Nogle løsninger, f.eks. Razor [9], vægter derudover brugernes input for at kunne identificere spam mere effektivt. Her gives også mulighed for at rapportere, at en given email ikke er spam. Det sidste er også tilfældet i det større DCC (Distributed Checksum Clearinghouse) netværk [10]. Her benyttes ydermere et distribueret servernetværk til at behandle og koordinere de mange checksummer. I DCC gives, da det er et open source projekt, mulighed for at store klienter, altså hele netværk, kan opstille deres egen DCC server. På denne måde belastes systemet mindre af brugerne på det pågældende netværk, og systemet styrkes ved at have endnu en node i DCC-servernetværket. Med denne opbygning vil en checksum for en ny spammail kunne spredes til alle brugere hurtigere.

Et system, der griber tingene lidt anderledes an er Brightmail [11]. I dette system er det ikke et stort antal brugere, der benyttes til at danne et filter, men et stort antal lokkedue-mailadresser. I Brightmail's service er oprettet over 2 millioner lokkedue-mailadresser [12]. Ideen er at disse bliver opsnuset af spammere og når en spammer derefter sender til alle tilegnede mailadresser, vil en betydelig del af disse tilhøre Brightmail-systemet. Herefter udarbejder et mailcenter, med kompetent filtreringspersonale, et filter på basis af disse emails, som brugerne herefter kan benytte. Systemet virker udmærket, og det er måske også derfor at mange store email-udbydere, herunder Hotmail, benytter Brightmail. Til gengæld kan de fleste, der benytter emailservicen hos Hotmail, bevidne at der til trods, stadig slipper en ikke ubetydelig mængde spam igennem.

3.1.1.e Kombinationer

Der findes en lang række kommercielle produkter, der kombinerer forskellige filtreringsmetoder. En ikke-kommerciel løsning, der samtidig er en af de bedste, er SpamAssassin [13]. I dette open source projekt benyttes en lang række tests for at afgøre, om en given email kan kategoriseres som spam. Af disse tests kan nævnes Bayes filtrering, blacklist/whitelist og collaborative filtrering, herunder DCC og Razor. Hver

test bliver tildelt en 'score' afhængig af resultatet, og til sidst kategoriseres emailen ud fra den samlede score. Dette betyder at en email, der fejlagtigt kategoriseres som spam af ét af filtrene måske ikke bliver det af de andre. Dermed vil emailen ikke nødvendigvis blive klassificeret som spam (*false positive*), som den ville være blevet, hvis det udelukkende var det 'fejlagtige' filter, som blev benyttet. Om dette vil være tilfældet for et givent 'kombineret' filter afhænger af, hvordan politikken for filtreringen er valgt. Dermed menes om forsigtighedsprincippet har været anvendt med henblik på at undgå *false positives*, eller en mere aggressiv filtreringspolitik benyttes, hvor det kræves at alle filtre 'siger god' for den pågældende email før den slippes igennem.

Det er ikke en umulig opgave for en spammer at undgå et filter, hvis denne kender funktionaliteten af dette. Det er dog straks en sværere opgave, når alle (eller i hvert fald de fleste) filtre i disse kombinationsløsninger skal omgås. En af ulemperne ved disse løsninger er dog, at de er meget ressourcekrævende. Dette skyldes de mange filtre, der anvendes simultant for hver enkelt besked, og dette kan være et problem i store organisationer, der modtager betydelige mængder email. Selvom disse løsninger er nogle af de bedste der findes i dag, er de stadig ikke tilfredsstillende. Hvorfor dette er tilfældet vil blive belyst i følgende afsnit.

3.1.2 Begrænsninger ved filtrering

Filtreringsmetoder er et effektivt og nok også det mest brugte middel mod spam. Generelt har filtrering dog sine begrænsninger. Udover de ulemper der allerede er blevet nævnt, vil der *altid* være en risiko både for at klassificere en spammail som legitim (*false negative*), og for at klassificere en legitim mail som spam (*false positive*). Det første vil betyde at nogle spammails vil slippe gennem filteret, hvilket godt kan accepteres i begrænset omfang, da det ikke koster mere end et par sekunder for modtageren. Det andet er sværere at acceptere, idet en mail indeholdende vigtig information kan blive tilbageholdt af filteret. Det kan være svært at finde en god balance mellem antallet af *false negatives* og antallet af *false positives*. Filtreringsmetoderne vil altid reducere det ene på bekostning af det andet. At der eksisterer en ikke ubetydelig sandsynlighed for *false positives*, betyder at mailsystemet som kommunikationsmiddel bliver mindre tilregneligt. I dag indsættes flere og flere filtreringsløsninger på brugerniveau, netværksniveau og sågar hos internetudbydere, hvilket altså betyder at emailsystemets pålidelighed reduceres. Fortsætter denne udvikling vil det i sidste instans betyde at brugerne af mailsystemet vil begynde at vælge andre veje, ja måske endda kuverter og frimærker i mange sammenhænge.

De fleste filterløsninger bygger på antagelsen om, at spammails sendes i mange kopier eller modificerede kopier (*duplicate detection*). Denne antagelse er korrekt for nærmest al den uønskede email en bruger modtager. Spammere har dog varieret indholdet af spammails på en sådan måde at budskabet, når et menneske ser på det, er uændret, mens variationen i en maskines fortolkninger er maksimeret. Dette har ofte givet dem mulighed for at undgå filtrene, som senere er blevet forbedret til at kunne håndtere de nye forsøg. Denne kamp har stået på længe, og det er stadig ikke lykkedes at lave et filter, der ikke kan omgås på denne måde. Forklaring på hvorfor dette er tilfældet, kan måske findes ved at betragte den teknik spammerne benytter mere eller mindre systematisk. Teknikken

kaldes *list-splitting* [15] og benyttes ved at tage udgangspunkt i noget tekst en spammer ønsker at sende. Herefter erstattes ord eller tegn systematisk med synonymer. Der kan både være tale om betydnings eller visuelle synonymer. På denne måde kan dannes et stort antal forskellige sætninger med samme betydning. Robert J. Hall viser faktisk i en artikel [15], at udnyttes denne teknik til fulde, vil filtre, der benytter duplikat detektion, *altid* kunne omgås.

Benyttes filtreringsmetoder til at filtrere spam, vil dette betyde, at mange vil skulle have en fælles opfattelse af, hvad der er spam og hvad der ikke er. Benyttes et personligt trænet Bayes filter, vil dette selvfølgelig ikke være tilfældet, men det er nok de færreste brugere, der vil bruge den tid, det løbende kræver at opretholde et sådant filter. Fælles filtre er altså umiddelbart eneste mulighed. Dette er i sig selv et stort problem hvis email mediet skal kunne bruges af alle som et frit og ucensureret medie. Derudover er det heller ikke foreneligt med den definition af spammail, der blev præsenteret i afsnit 2.1.

3.2 Betaling

En af de væsentligste årsager til at mængden af afsendte spammails er så kolossal er, at prisen for at afsende emails er meget lav. Dertil kommer, at en spammer kan afsende et meget stort antal emails på kort tid. Disse to egenskaber ved emailsystemet gør det attraktivt for spammere at sende millionvis af emails til alle de emailadresser, de kan komme i nærheden af. En udbredt løsning på dette problem er som beskrevet ovenfor, at bortfiltrere spammails, og lade legitime emails passere med de problemer som dette indebærer. I det følgende beskrives en helt anden type af løsninger. Disse forsøger at fjerne incitamentet for at spamme i første omgang.

3.2.1 Ressourcebetaling

Som nævnt ovenfor er et væsentligt problem ved emailsystemet, at man for meget få penge kan sende tusindvis af emails på kort tid. En løsning på dette problem kunne være at introducere et elektronisk frimærke på hver email, som det f.eks. er beskrevet i en artikel af Cynthia Dwork og Moni Naor [14] eller rent praktisk gøres i *HashCash*-systemet [16]. Frimærket sikrer at afsender har brugt en vis mængde computerkraft (og dermed tid) i forbindelse med afsendelsen af en email. I HashCash sikres dette ved at udnytte egenskaber ved envejs-funktioner (hashfunktioner). For at generere et frimærke skal afsender udføre et relativt stort antal beregninger på modtagers emailadresse og udvalgt data fra emailen (for at sikre at frimærket ikke kan genbruges, hverken til andre modtagere eller i andre emails). Beregningerne stopper, når afsender har fundet en såkaldt *delvis hashkollision* (se evt. afsnit 4.1). Dette betyder, at den beregnede hashværdi skal stemme overens med en forudbestemt værdi på de første *n* bits. Værdien *n* kan bruges til at bestemme hvor beregningstungt det skal være at generere hashkollisionen. Når kollisionen er fundet, sendes den med i headeren på emailen til modtageren (og udgør altså frimærket), som derefter blot skal kontrollere at en delvis hashkollision er fundet, og at frimærket ikke har været brugt før. Ideen med systemet er så, at for den almindelige bruger af emailsystemet, vil den tid det tager at generere de frimærker denne skal bruge være forsvindende, mens det for en spammer pludselig bliver uoverkommeligt at afsende emails i stort antal.

Et af problemerne med denne type system er dog, at de er meget afhængige af den cpu-kraft den enkelte har til rådighed. Dvs. at brugere med ældre og dermed langsommere maskiner vil opleve, at det tager væsentligt længere tid at sende emails end brugere med nyere hardware. Netop dette problem har flere forsøgt at løse [15, 17] ved at erstatte cpu-baserede beregninger med hukommelsesbaserede beregninger. Grunden til dette er, at memory-access hastigheder ikke varierer nær så meget som cpu-hastigheder på hhv. nyere og ældre maskiner. Som et konkret eksempel kan man sammenligne en nyere maskine med en Pentium4-3000 Mhz processor og 4GB ram med en ældre maskine, Pentium2-266 Mhz og 96 MB RAM. Førstnævnte er blevet målt til at være omtrent 10 gange hurtigere end sidstnævnte til rent cpu-afhængige beregninger, men 'kun' 2,67 gange hurtigere til en memory-access-afhængig beregning [15]. At erstatte cpu-baserede beregninger med hukommelsesbaserede virker altså umiddelbart fornuftigt i denne type løsninger.

Der findes også løsninger, hvor det er muligt at genbruge frimærker. Dette er f.eks. muligt ved at anvende en *ticketserver*, der står for udstedelse og indløsning af *tickets* (frimærker) [32]. Her er det naturligvis ikke serveren, der udfører de krævede beregninger, men klientmaskinerne. Med denne struktur vil beregninger, forbundet med dannelsen af et frimærke, blive begrænset, idet disse kan genbruges. Dette er en stor fordel, der gør løsningen langt mere brugbar i stor skala. Til gengæld introduceres en tredjepart i overførslen af hver email. Denne tredjepart vil blive et problematisk knudepunkt i emailsystemet.

Et generelt problem med disse beregnings- eller hukommelsesbaserede løsninger er, at de i bedste fald kun *reducerer* spamproblemet, men ikke eliminerer det. Man kunne forestille sig spammere forsøge at stjæle computerkraft, f.eks. vha. trojanske heste på almindelige brugeres computere (som det allerede ses i dag), og på den måde lave distribueret spamming. Et andet problem ved løsningen er, at der forbruges forholdsvis meget computerkraft til beregning af hvert frimærke. Dette koster penge (især globalt set), og man kunne argumentere for, at al denne på sin vis spildte computerkraft med fordel kunne bruges andetsteds. Et alternativ til ressourcebetaling, nemlig monetær betaling, som umiddelbart ikke har ovennævnte problemer, ser vi nærmere på i det følgende afsnit.

3.2.2 Monetær betaling

Den umiddelbare fordel ved at vedhæfte elektroniske penge frem for et 'proof-of-work' frimærke som i afsnit 3.2.1 er, at de elektroniske penge så at sige kan genbruges (da de ikke mister deres værdi når de bliver brugt). Derudover er det heller ikke forbundet med tunge beregninger hver gang der skal sendes en email, som det f.eks. er tilfældet med *HashCash*.

Et eksempel på et system, der tilbyder afsendelse og modtagelse af emails vedhæftet elektroniske penge, er *Cashette* [21]. Hvis modtager klassificerer en modtaget email som spam, kan denne vælge at beholde de vedhæftede elektroniske penge. I modsat fald får afsender sine penge tilbage. Den grundlæggende ide i systemet er altså, at den enkelte

bruger afgør, om en modtaget email kan betragtes som spam. Sætter man det beløb, der skal påhæftes hver email, passende højt er ideen, at spammere vil ophøre med deres aktiviteter, da det økonomiske incitament for at spamme ikke længere vil være til stede. Vælger spammerne alligevel at spamme, vil modtagerne blive kompenseret økonomisk for den modtagne spam. Løsningen kan eventuelt kombineres med en whitelist, indeholdende brugere (emailadresse) som kan sende til den pågældende bruger uden betaling. Ideen er beskrevet nærmere i IBM Systems Journal, 2002 [18].

En væsentlig begrænsning i Cashette-systemet er dog, at al e-mail, som bliver sendt imellem dets brugere, skal igennem Cashette's centrale (mail-) servere. Det er sådan der bliver holdt styr de forskellige brugeres pengebeholdning og sikres mod svindel eller misbrug fra brugernes side. Dette er dog et stort problem for anonymiteten i systemet, da administratorerne hos Cashette i princippet, udover at kunne læse alle brugeres email, kan lave komplette trafikanalyser på brugernes emailvaner og udnytte disse informationer i kommercielt øjemed.

Alternativt til Cashette kunne man forestille sig, at emailafsenderens internetudbyder betalte emailmodtagerens internetudbyder for hver enkelt afsendt email [22]. Forudsat en nogenlunde lige balance mellem modtagne og afsendte emails hos internetudbyderne, ville dette ikke være til udgift for den enkelte internetudbyder. En eventuel skæv fordeling ville kunne finansieres med højere abonnementspriser hos kunderne. Samtidig ville hver internetudbyder have en stor interesse i at stoppe spammere blandt kunderne så hurtigt som muligt. Løsningen vil dog kræve et omfattende samarbejde mellem de forskellige internetudbydere og vil ikke elimere spamproblemet, men blot reducere det. Spammere vil fortsat kunne spamme f.eks. via ubeskyttede mail-servere (*open-relays*) eller via trojanske heste installeret på almindelige brugeres computere.

At indføre en betaling på brugerniveau i stedet (så hver bruger betaler til modtager for modtagelse af mailen), ville give problemer, da der generelt er stor forskel på, hvor mange mails den enkelte bruger hhv. sender og modtager. Det er svært at argumentere for at brugere, som sender flere emails end de modtager, pludselig skal 'straffes' økonomisk.

Et generelt problem med betalingssystemer er, at de kræver tilpassede emailklienter eller proxyprogrammer installeret på servere eller den enkelte brugers maskine. Derudover vil der altid findes mennesker, som ikke vil kunne acceptere en betalingsløsning, og disse vil umiddelbart være afskåret fra at sende email til brugere af betalingssystemet.

3.3 Autentifikation

3.3.1 Afsenderautentifikation

Et af de største problemer ved det eksisterende mailsystem er muligheden for at *spoofe* afsenderadressen. Dette gør det muligt for spammere at sende spam med en falsk afsenderadresse, og på denne måde undgå at blive identificeret. At designe mailsystemet på ny, så dette er tilpasset nutidens krav, er tæt på at være en umulig opgave. Dette skyldes at internettets udvikling ikke kan styres centralt. Alle kan bidrage til dets udvikling, og alle kan benytte det. Derfor er det ikke let at blive enig om en ny standard.

En modifikation i mailsystemet, der kunne løse spamproblemet kunne være at kræve afsenders digitale signatur, som det f.eks. gøres i S/MIME. På denne måde er det ikke muligt at spoofe afsenderadressen, idet den digitale signatur identificerer afsender. Dette introducerer dog en række problemer. For det første vil brugerne af mailsystemet kunne identificere hinanden, *også* når der ikke er tale om spam, hvilket kan være et problem i mange sammenhænge, hvor en vis grad af anonymitet er vigtig for brugeren (*privacy*). For det andet er det langt fra en let opgave at give brugerne mulighed for at få en digital signatur, og sikre at dette ikke bliver misbrugt. I Danmark er det muligt at få sin egen digitale signatur, men i mange lande kan det være besværligt og dyrt at få en sådan. Digital signatur kan løse spamproblemet, men i praksis er det nok tvivlsomt om dette bliver anvendt. I praksis findes der heller ikke et globalt CA-hierarki (*Certification Authority*), hvilket er en nødvendighed, hvis signaturer skal anvendes ved afsendelse af email på tværs af landegrænser.

Løsninger med en svagere autentifikation er mere anvendelige i praksis og har stadig en betydelig effekt. For at besværliggøre netop spoofing af emailadresser kan en anden form for afsenderautentifikation benyttes. Før en email modtages kontrolleres det, ved at spørge afsenderdomænet om IP-adressen, som emailen kommer fra rent faktisk er en autoriseret mailserver på det pågældende domæne, og evt. også om emailadressen er gyldig. Er dette ikke tilfældet afvises emailen. Flere har forsøgt at få indført sådanne løsninger som standard. Microsoft forsøgte for mindre end et år siden at få indført en ny standard, *Caller ID*. Forslaget blev dog i første omgang afvist af IETF (The Internet Engineering Task Force), bl.a fordi Microsoft ville patentere dele af løsningen. Af andre løsninger kan nævnes DomainKeys fra Yahoo, der ikke er tilknyttet et patent, og *SPF* (Sender Policy Framework) udviklet af Meng Weng Wong fra Pobox. Ingen af ovennævnte er endnu blevet godkendt som RFC (Request For Comments) af IETF. Der blev dog udarbejdet et *draft* i August 2004 omkring det lignende system *Sender ID* [24]. Dette blev udarbejdet af arbejdsgruppen MARID (MTA Authorization Records In DNS), nedsat af IETF. *Sender ID* er en kombination af *Caller ID* og *SPF*, og er kompatibelt med *SPF*, idet information om hvilke mailservere der er gyldige i hvilke domæner gemmes i DNS-systemet. På denne måde bygges der på eksisterende løsninger, hvilket giver en lettere overgangsfase. Dette *draft* udløber dog februar 2005, og hvad der sker herefter indenfor dette felt er svært at spå om. Det virker dog ikke usandsynligt, at en Sender ID standard ser dagens lys indenfor nær fremtid.

At indføre afsender-autentifikation forhindrer adresse-spoofing, antaget alle servere benytter systemet. Dette kan begrænse svindel med email, og til en vis grad også begrænse spam. I dag skriver en spammer ofte blot en tilfældig afsenderadresse i fra-feltet i emailen. Det vil ikke være muligt at sende sådanne beskeder til en server, der benytter afsenderautentifikation, idet domænet i afsenderadressen enten ikke vil kunne findes, eller afsenderserveren ikke er godkendt på domænet. En spammer vil dog stadig kunne oprette et nyt DNS-navn og sætte dette til at pege på en *open-relay* server. Herefter vil der kunne sendes spammail via denne server med DNS-navnet som afsenderdomæne. Bliver dette også håndteret af systemet, vil en spammer kunne oprette et domæne med tilhørende mailserver i et land, hvor det ikke har retslige konsekvenser at sende spam. Herfra vil denne ugeneret kunne sende spam på trods af afsender-autentifikation (mere

om retslige forhold i afsnit 3.6). Et andet problem ved afsenderautentifikation er, at en bruger kan risikere at få blokeret sin email, hvis administratoren på hans domæne har lavet fejl i DNS-konfigurationen. Denne type løsninger kan altså også være med til at gøre emailsystemet mindre pålideligt.

3.3.2 Tillid

Når email modtages er problemet ofte at afsenderen er ukendt og muligvis kan være en spammer. Dette kan løses ved at have en pålidelig tredjepart tilstede ved overførslen af emailen. Sådanne løsninger vil altså involvere en tredjepart, som både afsender og modtager stoler på. Er denne situation opnået vil overførsel, hvor både afsender og modtager kontrolleres, være muligt [19]. Her indføres dog en del ekstra kommunikation, som i tilfælde af email virker overflødigt. I andre løsninger vil tredjeparten først blive involveret, hvis enten afsender eller modtager kræver det, altså *certificeret email*, hvor begge parter kan bevise den anden parts opførsel [20]. Benyttes denne teknik i emailsystemet, vil anonym spam heller ikke være muligt. Her kræves dog en PKI (Public Key Infrastructure) for at kunne anvende den nødvendige kryptografi. Derudover er der heller ikke længere tale om envejs-kommunikation. Findes en pålidelig tredjepart, kan overførsel foregå sikkert, både set fra afsenders og modtagers synspunkt. Problemet er, at det ofte er svært at finde denne tredjepart.

En anden mulighed, som måske passer bedre til strukturen af email-systemet er, at opbygge et tillids-netværk (*web of trust*). Ideen er her, at der kun modtages email fra afsendere, som modtager har en vis tillid til. F.eks. hvis afsender kender én, som kender en, som kender afsender. Om dette er tilfældet kan afgøres, hvis der opbygges et tillidsnetværk [22]. I praksis findes f.eks. GnuPG [23] baseret på PGP (Pretty Good Privacy), der opbygger et sådant netværk vha. kryptografiske nøgler. Dette bruges af mange til andre formål end spambekæmpelse. Et sådant netværk kunne være brugbart til bekæmpelse af spam, da en spammer (og alle andre) dermed kun kan sende til dem de kender via deres tillids-netværk, og derfor nok vil undlade at sende spam (for ikke at bryde bekendtskaber). En spammer kunne dog alligevel vælge at sende til dem, han kender indirekte, og dermed undgå at genere sine direkte bekendtskaber. Hans direkte bekendtskaber kunne dog stadig opdage denne opførsel, hvis det blev almindelig praksis at offentliggøre PGP-nøgler, der havde været anvendt til at spamme. Der er gode muligheder i denne metode, men at opbygge et tillidsnetværk kræver betydelig brugerinteraktion, hvilket vil afholde de fleste fra at bruge denne i praksis.

Der findes altså flere måder at indføre tillid i emailsystemet, med henblik på at begrænse eller undgå spam. Fælles for dem alle er dog, at de er mindre anvendelige i praksis. Nogle kræver meget ekstra kommunikation, mens andre bygger på systemer, der endnu ikke eksisterer. Sidst, men ikke mindst, skaber mange af sådanne løsninger problemer med at opretholde anonymiteten af den enkelte bruger. Dette er en meget vigtig egenskab ved det nuværende emailsystem, og en indskrænkning af dette vil reducere brugbarheden betydeligt.

3.4 Challenge/Response

Der findes en række Challenge/Response-systemer, som kan integreres i de mest populære emailklienter [26, 27]. Den grundlæggende ide i Challenge/Response-systemer er, at opretholde en liste (*whitelist*) over afsendere man gerne modtager emails fra. Modtages email fra ukendte afsendere, besvares denne automatisk med en såkaldt *Challenge*. Denne *challenge* er ofte en *captcha* [25], som er en simpel opgave at løse for et menneske, men meget besværlig og dyr at løse for en maskine. Besvares denne succesfuldt af afsender, afleveres emailen i modtagers indbakke, og afsender tilføjes til modtagers whitelist. I fremtiden vil afsenders emails få lov at passere, uden at denne først skal besvare en *challenge*. Da spammere normalt sender millionvis af emails, vil det være en uoverkommelig opgave at svare på disse *challenges* manuelt, og det er heller ikke umiddelbart muligt at besvare dem automatisk vha. en maskine. Dermed holdes spammails ude af indbakken, mens legitime emails får lov at passere.

Et sådant system kunne lyde som en ideel løsning på spamproblemet, men der er desværre en række generelle problemer med challenge/response-systemer. Grundet det store antal spammails i omløb vil størstedelen af de afsendte *challenges* være spild og dermed være årsag til en stor mængde unødvendig emailtrafik. Da afsenderadressen på emails let kan forfalskes kunne man forestille sig, at hvis challenge/response-systemer blev udbredt, ville angribere sende emails *fra* offerets emailadresse *til* emailadresser med challenge/response-systemer. Dermed ville disse automatiske systemer sende *challenge*-emails til offerets emailadresse og en angriber ville dermed kunne opnå at 'mail-bombe' denne med challenge/response-emails. Derudover kunne man forestille sig at spammere, for at undgå challenge/response-filteret, blot ville vælge afsenderadresser som et stort antal brugere antageligt vil have på deres *whitelist* (f. eks. news@cnn.com eller lignende). En løsning på dette problem kunne være at indføre digital signatur for at kunne verificere afsenders identitet.

Sidst men ikke mindst vil mange brugere føle det som et stort irritationsmoment at skulle besvare disse *challenges,* enkelte vil måske ligefrem tage det som en fornærmelse. I værste fald kan en *challenge* fejlfortolkes, som at modtagers mailsystem er i uorden og en vigtig email kan derfor forsinkes eller gå helt tabt.

3.5 Channels

"En spammer kan ikke sende email til dig, hvis ikke han kender din adresse" [28]. Det er netop denne tilstand der stiles efter i *email channel*–systemet [22, 28, 29, 44] opfundet af Robert J. Hall i 1998. Ideen er i bund og grund at tillade hver bruger at have mange varianter af den samme 'basis' email-adresse (hvor hver variant er en *channel*). Brugeren kan så f.eks. vælge at give venner og familie én *channel*, sine forretningsforbindelser en anden og offentliggøre en tredje til uopfordret mail (dog ikke skødesløst, så den med det samme havner i hænderne på alverdens spammere). Hvis der på et tidspunkt begynder at komme for meget spammail på en channel, kan den lukkes og en ny oprettes. Tilknyttet systemet er en såkaldt PCA (Personal Channel Assistant), som står for håndteringen af *channels,* og letter det administrative arbejde for den enkelte bruger.

Systemet minder om en simpel løsning på spamproblemet, hvor den enkelte bruger vælger kun at give sin 'rigtige' email-adresse til venner, familie og forretningsforbindelser han/hun stoler på, mens en 'skraldespands-mail' oplyses i tilfælde, hvor der kan være risiko for at denne misbruges (f.eks. tilmelding til nyhedsbreve eller lign.). Kommer der på et tidspunkt for meget spam på denne 'skraldespands-mail', kan den blot slettes og en ny 'frisk' (og dermed spamfri) oprettes. En af fordelene ved denne slags løsninger på spamproblemet er, at de kan kombineres med stort set alle andre løsninger. F.eks. kan man forestille sig at man *kun* filtrerer på den offentlige emailadresse (eller skraldespands-mailen), og dermed eliminerer risikoen for forekomster af *false positives* på sine vigtige personlige emailadresser. Derudover kan brugeren alene bestemme, hvad han/hun synes er spam. Hvis der f.eks. kommer en lind strøm af reklamer for produkt X på den offentlige email, og lige netop produkt X falder i den pågældende brugers smag, kan denne modtage disse uhindret. Hvis et filter var installeret, ville disse emails højst sandsynlig blive fanget og slettet inden de nåede brugerens indbakke.

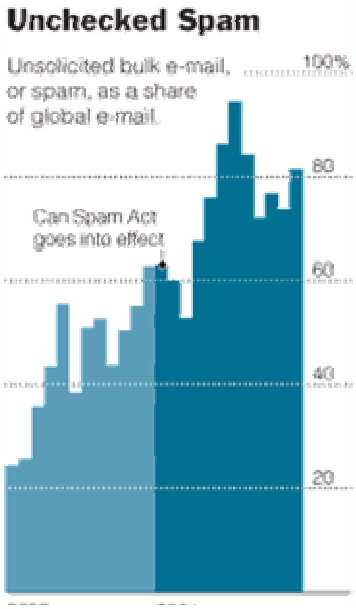
Der er dog en del ulemper ved disse løsninger. Det administrative arbejde forbundet med oprettelse og nedlæggelse af channels vil virke afskrækkende på en del brugere, især dem som endnu ikke er helt fortrolige med brugen af internettet og email-systemet. Desuden giver systemet intet svar på, hvordan spam kan undgås på den offentlige *channel*. En lukning af denne, når spammængden er vokset til et uacceptabelt niveau, har den negative konsekvens, at fremtidige legitime emails sendt til denne adresse vil gå tabt. Det er altså op til brugeren at afgøre, hvornår fordelene ved en lukning af den pågældende *channel* opvejer risikoen for mistede fremtidige legitime emails.

Kort sagt giver *email-channel*-løsningen mulighed for at reducere spammængden ved at 'tæmme' den, når den er løbet løbsk på én af vejene ind i indbakken. Ulempen er, at man i forbindelse med en lukning afskærer al fremtidig (legitim) kontakt ad denne vej, hvilket i mange tilfælde vil være problematisk.

3.6 Lovgivning

I diskussioner omkring spam har det ofte været foreslået at lovgive sig ud af problemet. Spammere burde kunne straffes og straffen burde være hård. Dette kunne måske afskrække andre spammere og give internetudbyderne det lovmæssige værktøj, de behøver for at holde deres netværk fri for spammere. Det mest nærliggende er, at det er internetudbyderne der retsforfølger spammerne, da det er dem, der umiddelbart har bedst mulighed for at finde synderne. Internetudbyderne har også en interesse i at standse spammere på deres netværk, idet disse er en stor belastning for netværket. På trods af dette er det begrænset hvor mange retssager internetudbyderne har ført. Dette kan skyldes, at det ikke kan betale sig at fange "de små fisk", da det er dyrt at føre retssager. Et andet problem er, at det ofte er svært eller endda umuligt at bestemme identiteten af spammerne. Så selvom der lovgives på området, er det ikke sikkert at den almindelige bruger vil opleve en reduktion i mængden af spam. I USA, hvor langt størstedelen af spammails afsendes, trådte der den 1. januar 2004 en ny stor spamlov i kraft. Den fik navnet CAN-SPAM (Controlling the Assault of Non-Solicited Pornography and Marketing Act of 2003), og skulle gøre en ende på (eller i hvert fald kraftig reducere

mængden af) spam. I denne sammenhæng er spam altså defineret som afsendelse af uopfordrede kommercielle eller pornografiske emails. Flere spammere blev straffet på basis af denne nye lovgivning, men måske fordi loven ramte forkert eller pga. manglende incitament til at benytte lovgivningen, voksede mængden af spam alligevel betydeligt i 2004 [30], som det blev beskrevet i afsnit 2.2 (se Figur 2). Et af problemerne med denne lov var, at lovens definition af spam ikke passede til virkeligheden (som beskrevet i afsnit 2.1). Det var kun en lille del af den spam, der blev sendt i 2004, der var omfattet af loven [31].



Figur 2 – Mængden af spam før og efter indførelsen af CAN-SPAM lovgivningen.

Det er ikke nogen let opgave at lovgive indenfor spamområdet. At definere hvilke emails der kan betragtes som spam og dermed hvilke emails, der er strafbare at sende, kan være uhyre svært. Dette er også grunden til, at CAN-SPAM og andre lovmæssige tiltag efter manges overbevisning kan risikere at krænke den enkeltes ytringsfrihed, da visse typer af emails forbydes helt. Derudover vil det altid være en hårfin balancegang på den ene side at kunne straffe spammere og på den anden side at respektere folks privatliv.

Herhjemme findes også eksempler på, at det er svært at lovgive sig ud af problemerne. I valgkampen i begyndelsen af 2005 benyttede flere politikere sig af afsendelse af uopfordrede emails for at påvirke befolkningen. Lovgivningen var dog ikke tilstrækkelig til at kunne gribe ind, da spammails i denne er defineret som uopfordrede emails med et kommercielt formål. Da formålet i denne sammenhæng ikke var kommercielt, kunne der ikke slås ned på de pågældende.

Én ting er at forsøge at lovgive sig ud af problemet, noget helt andet er at forsøge at håndhæve denne lov. Dette er praktisk taget umuligt at gøre effektivt, da internettet ikke kender til landegrænser. Findes der en god lov omkring spam i ét land vil dette ikke kunne begrænse mængden af spam der sendes *til* landet, men udelukkende den spam der sendes internt i landet og *fra* landet. Dette betyder at spammere blot kan flytte deres aktiviteter til lande, med lempelige regler på området. For at styre brugen af internettet vha. love kræves altså en global lovgivning, hvilket med nutidens magtstruktur er utopi. Så indtil der findes en fælles lovgivende enhed vil denne fremgangsmåde have begrænset effekt.

3.7 Opsummering

Vi har nu fået belyst en række mere eller mindre succesfulde løsninger på spamproblemet. Fælles for dem er, at ingen af dem er perfekte. At vælge en spamløsning i dag vil altså i høj grad bestå i at vælge *den* eller *de* bedste løsninger blandt de dårlige. Dette valg skal træffes under hensyntagen til hvilke *typer* af ulemper ved de pågældende løsninger, man er villig til at acceptere.

De mest udbredte løsninger på spamproblemet i dag er filtreringsløsninger. Deres popularitet skyldes sandsynligvis, at de er relativt simple at installere hos enten den enkelte bruger eller på mailserverne, og at de bedste af dem fanger langt størstedelen af den spam, der passerer dem. Som nævnt i afsnit 3.1.2 er filtreringsløsninger dog alle forbundet med risikoen for *false positives*. Denne risiko har de fleste internetbrugere efterhånden mere eller mindre frivilligt accepteret. Ofte installeres filtre på internetudbyderes mailservere, hvorved alle med opkobling hos denne får filtreret indkommende mails, om de vil det eller ej. Den udbredte holdning blandt fortalere for filtreringsløsninger synes at være, at den reduktion i spammails som opnås vha. filtrering, opvejer den relativt lille risiko for *false positives*.

Mange mener dog at filtre ikke er et særligt godt bud på en endelig løsning på spamproblemet. Dette skyldes hovedsageligt, at de skader emailsystemets pålidelighed, og dét uanset om risikoen for false positives er relativt lille. I situationer hvor man forventer at modtage en vigtig email, men denne lader vente på sig, vil det være svært at slippe den tanke, at den pågældende email *kunne* være blevet fanget af et filter. Denne usikkerhed omkring emailsystemet stiger i takt med udbredelsen af filterløsninger og har været en af hovedmotivationerne for at udvikle en spamløsning, der ikke er plaget af *false positive*-problemet og samtidig er et effektivt middel til bekæmpelse af spam.

4 Kryptografi

Dette afsnit vil give et indblik i de kryptografiske algoritmer, der ligger til grund for sikkerheden i dette projekt. Først vil hashfunktioner blive diskuteret, derefter symmetrisk og asymmetrisk kryptografi.

4.1 Hash-funktioner

En hashfunktion er en envejsfunktion, hvor størrelsen af outputtet er fast. En hashfunktion bruges til at danne en hash-værdi (eng: *message digest*) af data, som er et slags fingeraftryk af data. En hashværdi vil ofte være en relativt kort binær streng med en længde på typisk 128 eller 160 bit. Hvis data ændrer sig skal hash-værdien med stor sandsynlighed *også* ændre sig (i en ideel hashfunktion vil denne sandsynlighed være $1-\frac{1}{2^n}$, hvor *n* er antallet af bits i funktionens output). Dermed kan hash-værdien bruges til at sikre integritet af data. Dette kræver dog, at hash-værdien opbevares et sikkert sted, så denne ikke kan ændres. Hvis dette er tilfældet, kan data opbevares et usikkert sted (eller sendes til en modtager gennem en usikker kanal). Hash-værdien kan så genberegnes og resultatet sammenlignes med det tidligere beregnede hash-værdi, når man vil sikre sig at integriteten af data er bevaret.

Det er ønskeligt, at kryptografiske hashfunktioner kan bruges på vilkårligt data (og dermed også data, hvis længde overskrider bit-længden på den producerede hash-værdi). Dette betyder at det er uundgåeligt, at der vil eksistere forskellige meddelelser, som giver *samme* hash-værdi. Denne situation kaldes populært en kollision, eller *hash-kollision*. Haves to stykker data, som hver giver samme hash-værdi, vil disse kunne udbyttes uden at dette kan detekteres. Derfor er det vigtigt, at en hashfunktion er sikker. Dette er tilfældet hvis (og kun hvis) følgende tre opgaver *alle* er svære at løse for den pågældende hashfunktion *H:* $x \rightarrow y$:

1. **Envejs:** Givet et *y*, skal findes et *x*, så $H(x) = y$.
2. **Svagt kollisionsfri:** Givet et *x*, skal findes *x'* således at $x \neq x'$ og $H(x) = H(x')$.
3. **Stærkt kollisionsfri:** Find *x* og *x'*, således at $x \neq x'$ og $H(x) = H(x')$

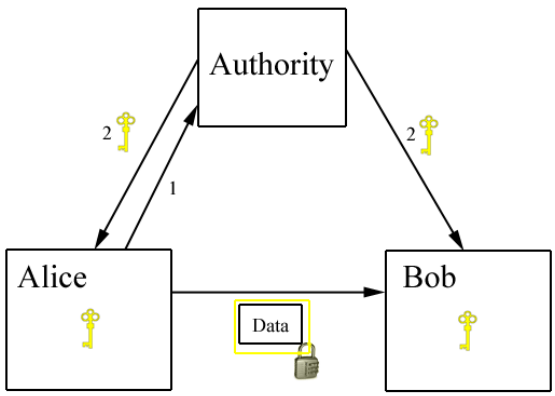
Bemærk at opgaven i punkt 2 altid vil være langt sværere at løse end opgaven i punkt 3, da der i opgave 2 kræves at en kollision findes for noget data, som er givet på forhånd. Benyttes en (envejs-) hashfunktion som genererer tilfældigt, helt ligeligt fordelt output, vil det kræve, at man i gennemsnit skal beregne hashværdier for halvt så mange meddelelser, som der er mulige outputs for at finde en kollision som angivet i punkt 2. Benyttes der således en hashfunktion, der returnerer hash-værdier med længden *n* bit, vil det kræve beregning af hashværdier for i gennemsnit 2^{n-1} meddelelser før en sådan hashkollision findes. I opgaven i punkt 3 skal 'blot' findes to vilkårlige meddelelser som giver samme hash-værdi. Dette kræver langt færre beregninger. Situationen svarer faktisk til det såkaldte fødselsdagsparadoks, som siger, at det blandt spillere og dommer på en fodboldbane (23 personer i alt) vil være mere end 50 % sandsynligt, at to af dem har fødselsdag samme dato (hvis man ser bort fra årstallet). Dette er naturligvis ikke et

paradoks, men et måske ikke så intuitivt faktum. Antages det at antallet af mulige outputs for en hashfunktion er 365, vil det således kun være nødvendigt i gennemsnit at beregne 23 hashværdier før en kollision findes. Nu har hashfunktioner normalt (heldigvis) langt flere mulige output. Et generelt resultat siger at det i gennemsnit er nødvendigt at beregne cirka $1.17\sqrt{M}$ hashværdier for at finde en kollision, hvor M er antallet af mulige outputs [42]. For en 40-bit hashfunktion vil det altså være i størrelsesordenen $2^{20} \approx 10^6$ beregninger, der skal udføres i gennemsnit. På en af nutidens maskiner kan dette beregnes indenfor kort tid. Derfor anbefales det normalt, at man som minimum benytter 160-bit hashfunktioner, hvor det så vil være nødvendigt at beregne i størrelsesordenen $2^{80} \approx 10^{24}$ hashværdier i forbindelse med et 'fødselsdags-angreb' (eng.: *birthday attack*). Dette er en uoverkommelig opgave for selv de kraftigste maskiner, der findes i dag. Størrelsen af den beregnede hashværdi har altså direkte indflydelse på sikkerheden af hashfunktionen. Bemærk endvidere, at sørger man for at opgaven i punkt 3 er svær at løse, vil man automatisk opfylde punkt 2 også.

I dag er nogle af de mest velkendte og benyttede hashfunktioner MD5 og SHA-1, der danner hhv. en 128 bit og en 160 bit hash-værdi. Der er blevet fundet små kollisionsproblemer i begge funktioner ([47] og [48]), men ingen af dem har haft stor betydning for sikkerheden (på kort sigt). Disse hashfunktioner anvendes derfor stadig i mange applikationer. Ønsker man at benytte mere sikre hash-funktioner findes f.eks. SHA-256 og SHA-512, der danner hhv. 256 og 512 bit hash-værdier.

4.2 Symmetrisk kryptografi

I symmetrisk kryptografi benyttes den samme nøgle til at kryptere og dekryptere data. Betragtes en situation, hvor Alice ønsker at sende data til Bob over en usikker kommunikationskanal ser situationen ud som på Figur 3.



Figur 3 – Alice sender en krypteret besked (data) til Bob vha. symmetrisk kryptografi.

Først skal Alice og Bob blive enige om hvilken nøgle, der skal benyttes ved overførslen. Dette kan ske ved at sende nøglen gennem en anden kommunikationskanal (f.eks. postvæsenet) eller mødes og udveksle nøglen. En tredje mulighed er, at lade en fælles betroet tredjepart (*authority*) uddele en nøgle som på Figur 3. I denne situation vil Alice først meddele, at hun ønsker at kommunikere med Bob. Herefter vil Alice og Bob

modtage en nøgle via en sikker kommunikationskanal. Til sidst kan data sendes krypteret mellem Alice og Bob.

Når data skal krypteres kan dette gøres med en række forskellige algoritmer. Lad os først betragte en klassisk krypteringsalgoritme, substitutions-algoritmen. I denne algoritme er nøglen en permutation af alfabetet. Der opstilles altså en permutation af alfabetet som herunder:

Alfabet	A	B	C	D	E	F	G	...
Nøgle	H	V	R	J	L	K	W	...

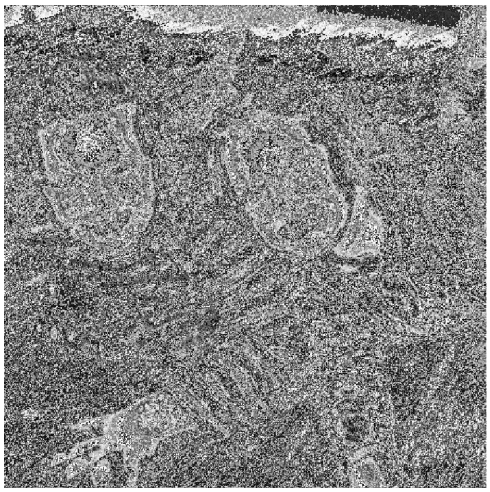
Kryptering foregår nu ved at ombytte hvert bogstav i beskeden med det tilsvarende bogstav i permutationen [42]. Altså ombyttes alle A'er med H'er, B'er med V'er osv. Dekryptering foregår ved at substituere den modsatte vej. Substitutionsalgoritmen blev først anvendt af romerne, og kan ikke umiddelbart brydes ved et *brute-force* angreb. Er der nemlig blot 28 tegn i alfabetet, vil der findes 28! ($\approx 10^{29}$) mulige nøgler. Der findes dog kryptoanalytiske teknikker, hvorved data relativt let kan dekrypteres uden kendskab til nøglen.

En anden klassisk krypteringsalgoritme er permutations-algoritmen. Her er den grundlæggende ide at ombytte bogstaver i beskeden. Her behøver nøglen ikke have samme længde som alfabetet, og indeholder indeks i stedet for symboler. Har nøglen f.eks. længden 8, kan denne se ud som herunder:

Indeks	1	2	3	4	5	6	7	8
Nøgle	5	3	4	7	2	8	1	6

Nøglen er altså en permutation af indeks fra 1 til 8. Kryptering foregår nu ved at beskeden krypteres 8 tegn ad gangen. Disse 8 tegn ombyttes som nøglen angiver. Altså vil det 1. tegn flyttes til plads 5, det 2. til plads 3 osv. Denne algoritme kan også brydes vha. kryptoanalyse, og bliver derfor heller ikke anvendt i praksis i dag. Disse to algoritmer kan ikke bruges til meget alene, da der altså findes velkendte angreb mod disse. Kombineres algoritmerne derimod, opnås langt bedre resultater, og netop disse to algoritmer er grundstenen i nutidens mest anvendte symmetriske algoritmer (DES og AES). På Figur 4 er illustreret hvordan resultatet forbedres betydeligt, når algoritmerne kombineres. Her ses et 8 bit sort-hvid billede af Lasses kærestes fætter og kusine (Lukas og Amalie). Det originale billede ses til venstre (a). I højre side ses øverst billedet krypteret med substitionsalgoritmen (b). I midten til højre ses billedet krypteret med permutationsalgoritmen (c). Nederst ses billedet krypteret med begge algoritmer (d), (med samme nøgler som i (b) og (c)). Kryptering af billederne er foretaget i Matlab, og koden kan ses i Appendiks G.

(a) Original

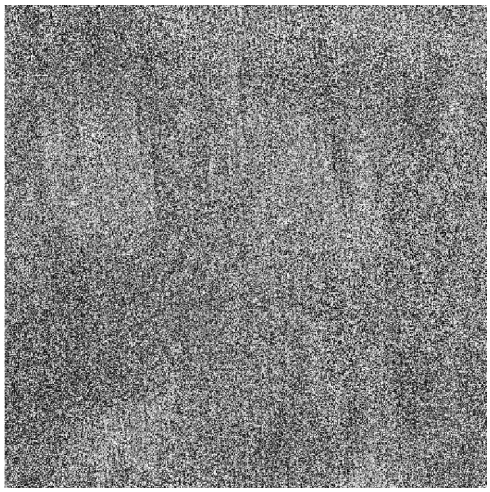


(b) Substitution

(a) Original

(c) Permutation

(a) Original



(d) Substitution og permutation

Figur 4 – Et 8 bit sort-hvid billede til venstre (a). Billedet krypteret med substitutionsalgoritmen (b), med permutationsalgoritmen (c) og begge algoritmer (d).

Efter kryptering af billedet med substitutionsalgoritmen kan ansigterne og striberne på Lukas' bluse stadig skimtes (b). Algoritmen har altså ikke skjult information i billedet særligt effektivt. Betragtes resultat efter permutationsalgoritmen (c) har billedet en helt anden karakter. Her kan børnenes placering i billedet dog stadig ses, og farverne på deres tøj kan også anes. I det sidste billede ses resultatet efter at have anvendt begge algoritmer. Her er det svært overhovedet at få en ide om hvad billedet forestiller. Den eneste information der kan udledes er, at visse områder er en smule lysere eller mørkere end andre.

At algoritmernes output har meget forskellig karakter, er netop grunden til kombinationens styrke. Grundlæggende er det samme princip, der benyttes i Feistel blokalgoritmer [42]. Her udføres substitution og permutation i flere *runder*, for derved at opnå bedre resultater og besværliggøre statistisk analyse. En Feistel-algoritme kan operere i flere forskellige *modes*. Disse er angivet herunder:

- *Electronic codebook mode* (ECB)
- *Cipher blockchaining mode* (CBC)
- *Cipher feedback mode* (CFB)
- *Output feedback mode* (OFB)

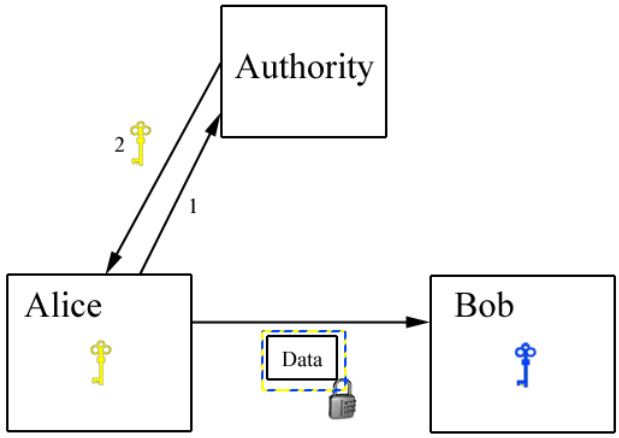
I ECB-mode krypteres hver enkelt blok data for sig selv. Dette betyder at alle blokke er uafhængige. Dette kan være en fordel hvis der forekommer bitfejl efter overførsel på et netværk, men også være en ulempe, idet blokke f.eks. kan udbyttes på netværket i et replay-angreb. I CBC- og CFB-mode sammenkædes forrige blok med den nuværende under kryptering. I CBC-mode anvendes *xor* på den krypterede blok fra forrige trin og blokken fra nuværende trin inden kryptering. I CFB-mode krypteres resultatet fra forrige trin, hvorefter dette *xor*'es med blokken i nuværende trin. I OFB-mode bliver algoritmen til en stream algoritme. En initialiseringsvektor (tilfældige tal) krypteres gentagne gange og hvert mellemresultat *xor*'es med en blok i beskeden.

De to mest velkendte Feistel-algoritmer er DES (*Data Encryption Standard*) og AES (*Advanced Encryption Standard*). DES blev udviklet i 1973 [42]. Algoritmen benytter en 56-bit nøgle til at kryptere blokke på 64 bit i 16 runder. I slutningen af halvfemserne blev nøglestørrelsen et problem. Derfor introduceredes en udvidet version (*triple-DES*), men behovet for en bedre algoritme, der kunne understøtte flere nøglestørrelser blev for stort. Dette resulterede i udviklingen af AES, der blev en standard i 2001. AES er en hurtigere Feistel-algoritme, der understøtter flere nøglestørrelser. Nøglestørrelserne er 128, 192 og 256 bit, og her benyttes hhv. 10, 12 og 14 runder. Derudover er AES-algoritmen designet, så denne bedre kan implementeres i hardware. AES benyttes i dag verden over og der er endnu ikke fundet succesfulde angreb mod AES.

Der findes mange andre algoritmer med lignende egenskaber. Her er dog blevet fokuseret på DES og AES, da det er disse algoritmer, der anvendes i systemet.

4.3 Asymmetrisk kryptografi

Et af de største problemer ved symmetrisk kryptografi er nøgledistribution. I den asymmetriske kryptografi lettes dette problem. Her haves et *nøglepar*, hvor den ene (offentlige) nøgle benyttes til at kryptere, mens den anden (private) nøgle benyttes til dekryptering. Når Alice ønsker at sende data til Bob kan dette foregå som vist på Figur 5.



Figur 5 – Alice sender krypteret data til Bob vha. asymmetrisk kryptografi. Den gule nøgle er Bobs offentlige nøgle og den blå er Bobs private nøgle.

Først kontakter Alice en autoritet (CA) for at få Bobs offentlige nøgle. Herefter kan Alice kryptere og sende beskeden til Bob, som herefter kan dekryptere beskeden vha. sin private nøgle. På denne måde skal Bob altså ikke tildeles en ny nøgle for at modtage beskeden fra Alice, og nøglen kan bruges af flere forskellige til at sende til Bob. Dette var ikke tilfældet med symmetrisk kryptografi, hvor nøglen dels skulle sendes til Bob og derudover kun kunne bruges af Alice.

Asymmetriske kryptosystemer kan også benyttes i forbindelse med digitale signaturer. Dette gøres grundlæggende ved at kryptere data med den private nøgle (*signering*), hvilket kun Bob er i stand til, da det kun er ham der kender sin private nøgle. Herefter kan data dekrypteres med den offentlige nøgle, når signaturen skal kontrolleres (*verificering*). Dette kan gøres af alle, da den offentlige nøgle ikke skal holdes hemmelig. Der findes i dag adskillige asymmetriske systemer. Fælles for dem er, at sikkerheden er baseret på vanskeligheden i at løse matematiske problemer. Det er dog kun få systemer, der er praktiske (mulige at implementere effektivt) og samtidig både kan bruges til kryptering og signering. I Tabel 1 er angivet de mest populære kryptografiske algoritmer og de matematiske problemer disse er baseret på.

Algoritme	**Matematisk problem**
RSA (*Rivest, Shamir, Adelman*)	Faktorisering af store tal
Rabin-Williams	Kvadratrodsberegning (eller faktorisering af store tal)
El-Gamal	Det diskrete logaritmeproblem
Elliptisk Kurve kryptografi	Det diskrete logaritmeproblem

Tabel 1 – Algoritmer og de matematiske problemer som disse bygger på.

Af disse er RSA nok den mest anvendte algoritme. De andre algoritmer omtales ikke yderligere her. Til gengæld vil RSA-systemet blive gennemgået i detalje for bedre at kunne forklare blinde signaturer i efterfølgende afsnit. RSA er bygget op på følgende måde. Lad $n = pq$, hvor p og q er primtal. Derudover skal det gælde at $ab \equiv 1 \bmod \varphi(n)$, hvor φ er Eulers φ-funktion. Den private nøgle er (p, q, a) og den offentlige nøgle er (n, b). Kryptering og dekryptering foregår nu som beskrevet herunder:

$$Encrypt_{[n,b]}(x) = x^b \bmod n$$

$$Decrypt_{[p,q,a]}(y) = y^a \bmod n$$

Lykkes det at faktorisere n, altså finde p og q, kan a let beregnes og sikkerheden er brudt [42]. Da p og q er store primtal, er det dog en *meget* tidskrævende opgave (der endnu ikke findes en hurtig løsning på). Derfor betragtes RSA dags dato stadig som et sikkert asymmetrisk kryptosystem. I RSA kan benyttes forskellige nøglestørrelser, og de fleste implementationer understøtter 512, 1024 og 2048 bits.

Asymmetriske kryptosystemer har en række fordele frem for de tilsvarende symmetriske kryptosystemer. En af de største ulemper er dog hastigheden; de er langt mere tidskrævende end symmetriske kryptosystemer. F.eks. er RSA ca. en faktor 100 langsommere end AES i software og en faktor 1000 langsommere i hardware. Derfor er det ikke praktisk at benytte asymmetrisk kryptografi til at kryptere store mængder data. I stedet benyttes det typisk til kryptering af symmetriske nøgler. Dette kan både benyttes til at distribuere symmetriske nøgler, men også i hybridkryptering[1]. Når algoritmerne skal anvendes til at signere data, er hastigheden også et problem. Derfor signeres normalt en hashværdi af data i stedet. Verifikation foregår da ved at kontrollere at signaturen er gyldig, og dernæst hashe data og sammenligne resultatet med den signerede hashværdi.

I følgende afsnit vil blinde signaturer i RSA blive gennemgået med henblik på senere anvendelse.

[1] Hybridkryptering foregår ved først at kryptere en tilfældig symmetrisk nøgle med en assymetrisk nøgle, og herefter kryptere data med den symmetriske nøgle. Deraf navnet hybrid, idet de to typer kryptografi kombineres.

4.4 Blinde signaturer i RSA

Blinde signaturer der bygger på RSA-algoritmen blev opfundet af David Chaum. I den simpleste form foregår det som beskrevet i det følgende [43]. Vi har et RSA-system som beskrevet ovenfor. Bob har altså en offentlig nøgle *b*, en privat nøgle *a*, og et offentligt modulus *n*. Alice (og Bob) ønsker nu at Bob skal signere en besked *m* fra Alice blindt. Dette gøres i RSA ved at Alice og Bob udfører følgende trin:

1. Alice genererer et tilfældigt tal *k* (blindingfactor) mellem 1 og *n* og 'blinder' *m* ved at beregne:
   $t = mk^b \pmod n$
2. Bob kan nu signere *t*, der ikke afslører noget om *m*, og danne den blinde signatur *bs*:
   $bs = t^a \pmod n$
3. Alice 'unblinder' herefter *bs* vha. *k*, og opnår Bobs signatur *s* af *m*:
   $s = bs / k \equiv m^a \pmod n$

Det sidste kongruenstegn gælder, idet $ab \equiv 1 \bmod n$, og dermed $bs/k = t^a/k \equiv((mk^b)^a)/k \equiv(m^a\, k^{ab})/k \equiv (m^a k)/k \equiv m^a \pmod n$ [42]. Hermed har Bob signeret *m* uden at kende *m*. Hvornår Bob vil signere noget uden at kende indholdet, afhænger i høj grad af sammenhængen, men disse primitiver anvendes i mange sammenhænge, bl.a. til at opnå anonymitet.

4.5 Opsummering

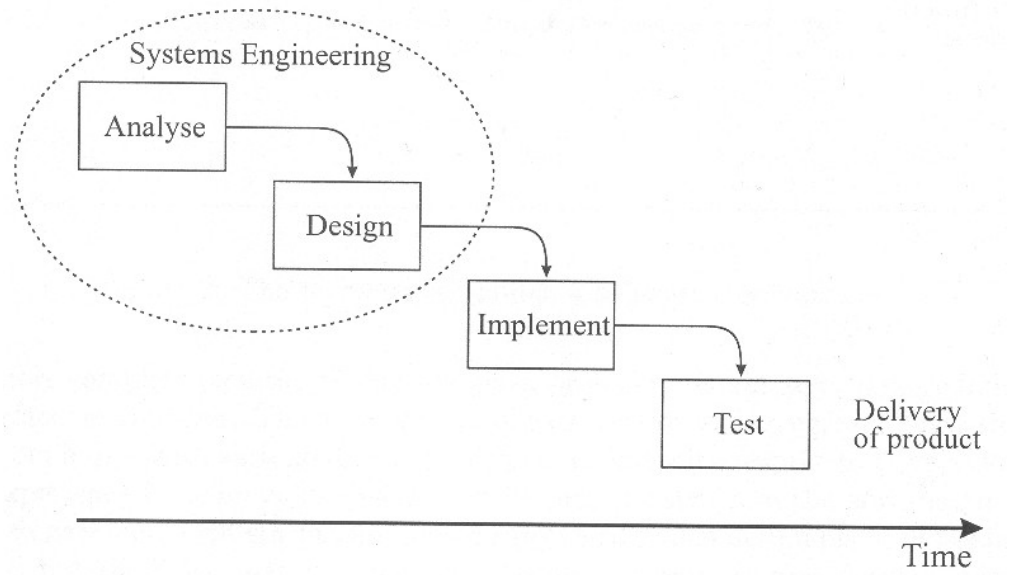
Nu er de kryptografiske algoritmer, der ligger til grund for dette projekt, blevet gennemgået. Det er herunder ikke blevet diskuteret, hvorledes disse værktøjer kan indgå i større sammenhænge, hvor f.eks. autentifikation og fortrolighed (confidentiality) skal sikres. Dette (og meget andet) vil i stedet blive behandlet, når selve systemets design udarbejdes (afsnit 6).

5 Analyse

5.1 Udviklingsmodel

Nu er nutidens spambekæmpelsesteknikker blevet diskuteret og en række kryptografiske primitiver er blevet opstillet. I den resterende del af rapporten vil blive udviklet et system, der benytter den omtalte kryptografi til at opnå en antispamløsning baseret på et betalingssystem. Inden dette sker, vil vi dog lige fastlægge hvilken udviklingsmodel, der benyttes.

Ethvert softwareprojekt består grundlæggende af fire faser: Analyse af krav, design, implementation og test. Disse faser kan sammensættes i flere forskellige designmodeller. I mindre projekter vælges typisk en lineær "vandfaldmodel" [46], som vist på Figur 1, da denne er den mest simple og umiddelbart mindst tidskrævende. Den har dog den ulempe, at fejl fundet under testfasen ofte kun rettes i implementeringsfasen i stedet for at blive re-analyseret og re-designet. I mere komplekse projekter eller i projekter, hvor man ikke fra starten besidder tilstrækkelig erfaring til at gennemføre én fase ad gangen, vælges typisk en anden udviklingsmodel, som f.eks. inkrementeringsmodellen eller prototypemodellen [46].



Figur 1 - Vandfaldmodellen - en designmetode ved softwareudvikling.

I dette projekt har vi valgt at lægge os op ad den lineære vandfaldsmodel. Vi springer dog lidt mere mellem faserne end der er lagt op til. Dette skyldes, at vi ikke i alle situationer har nok erfaring fra projektets start til at færdiggøre designfasen uden at eksperimentere med forskellige implementeringsmuligheder. Grundlæggende bliver der dog ikke ændret meget på tværs af faserne.

De resterende dele af denne rapport er derfor opbygget efter vandfaldsmodellen. Således vil de følgende fire kapitler indeholde hhv. analyse, design, implementation og evaluering. Til sidst vil en konklusion opsummere resultater og diskutere det implementerede system.

5.2 Grundlæggende ide

Betalingssystemer, der har til formål at bekæmpe spam, baseres på grundtanken om, at en spammer skal straffes økonomisk. Hermed fjernes incitamentet for at sende spam, idet prisen for at sende en stor mængde emails langt vil overstige gevinsten. Blot at kræve en økonomisk betaling for hver eneste email vil have store konsekvenser for brugbarheden af mailsystemet, og det vil muligvis ødelægge mere end det vil gavne. Et betalingssystem, hvor det sikres at det udelukkende er spammail, der kræver betaling, er derfor at foretrække. Her er det vigtigt, at brugerne kan acceptere definitionen af spam, sådan at ingen skal betale for noget, de mener, er legitim mail. I afsnit 2.1 blev spam defineret som email, der er uønsket af modtageren. Det er altså netop modtagers definition, der benyttes til at afgøre om en email kræver betaling eller ej.

Grundideen i betalingssystemet der udvikles i det følgende er, at når en bruger sender en email, vedhæftes betaling i form af en elektronisk mønt. I systemet modtager brugere således kun email med vedhæftet e-mønt. Som udgangspunkt vil email fra brugere, der ikke benytter systemet således blive afvist. Hvis en bruger modtager en email, som denne definerer som spam, kan denne opkræve betalingen vedhæftet emailen. Foretager brugeren sig intet, vil betalingen udløbe og returnere til afsender. Sendes ingen spam, vil det altså være gratis at benytte systemet. Sendes spam vil dette koste penge netop for spammerne og belønne ofrene tilsvarende. Hermed vil incitamentet for at sende spam med henblik på at tjene penge forsvinde og spam, der sendes på trods af betalingen, vil være indbringende for modtager.

5.3 Sideeffekter

Hovedeffekten ved indførelse af et betalingssystem er, at spam stort set vil blive elimineret, da det økonomiske incitament for at spamme ikke længere vil være til stede. Udover denne positive hovedeffekt, er der dog en række mere eller mindre negative sideeffekter, som vil blive beskrevet i det følgende.

5.3.1 Møntjægere

Så snart der introduceres et system baseret på betaling af en eller anden art, vil der findes folk, som forsøger at misbruge dette for egen vindings skyld. Når det alene er op til modtageren af en given email at afgøre, om denne ønsker at indløse den vedhæftede mønt eller ej, vil det være oplagt for en misbruger på en eller anden måde at få et stort antal mennesker til at sende emails til sin emailadresse. For en spammer kan disse mange modtagne mails betyde, at han eller hun kan udsende spam ’gratis’ ved at vedhæfte de modtagne mønter. Alternativt kan misbrugeren vælge blot at indløse mønterne. Der er umiddelbart en del måder, hvorpå man kan lokke brugere til at fremsende emails til sin emailadresse. Eksempelvis kunne man som misbruger udskrive en falsk konkurrence, hvor et krav for deltagelse er, at man fremsender en email med eksempelvis navn og

adresse. Misbrugeren kan så undlade at udsende præmier og blot indløse de mønter, som var påhæftet de modtagne emails. En anden mulighed kunne være, at sætte varer meget billigt til salg på f.eks. en internetside og som eneste kontaktmulighed angive en emailadresse. Man kan så forestille sig, at et stort antal brugere vil forsøge at kontakte "sælger", mens denne ignorerer indholdet af de mange emails og blot indløser de modtagne mønter. Noget lignende kunne gøre sig gældende med f.eks. web-sider, der kunne lokke med gratis adgang til indholdet på siden, hvis bare man sender en email med f.eks. ønsket brugernavn og adgangskode. Mere udspekulerede misbrugere kunne forsøge at sprede trojanske heste eller vira, som efter opstart på offerets maskine forsøger at sende så mange emails som muligt til misbrugers egen emailadresse. Misbrugeren kan så efter modtagelse af disse emails indløse de påhæftede mønter.

5.3.1.a Channels

Ovenstående scenarier viser at indførelse af et betalingsbaseret emailsystem, som det ovenfor beskrevne, kræver at sikkerheden på de lokale maskiner er i orden (for at undgå tilfælde med vira og trojanske heste). Brugeren har dog også et ansvar. I nogle tilfælde kan det dog være meget svært at gennemskue om f.eks. en konkurrence eller en salgsannonce er det rene fup eller ej. En løsning på dette kunne være at kombinere betalingsløsningen med et *email-channel*-system som beskrevet i afsnit 3.5. For f.eks. konkurrencer og salgsannoncer gælder nemlig næsten altid, at disse har en begrænset løbetid, hvorefter de bliver uaktuelle og nedlægges (når vinder er fundet, eller varen er solgt). Dermed er det oplagt for en (seriøs) konkurrencestifter eller sælger at oprette en email-channel til al kommunikation i forbindelse med konkurrencen/salget. På denne channel skulle der så *ikke* kræves betaling for modtagne emails. Når salget eller konkurrencen er slut kan den pågældende channel blot nedlægges. Den relativt korte tid den pågældende channel er åben, vil i reglen ikke være nok til, at spammere får fingrene i denne. Skulle det alligevel ske, vil der igen være lukket for spam så snart konkurrencen/salget ophører.

5.3.2 Brugeradfærd

Det er dog ikke kun inkarnerede misbrugere af systemet, der kan udgøre et problem. Brugere, som normalt ikke misbruger systemet, kan måske pludselig fristes til at indløse alle deres mønter i indbakken (f.eks. pga. pengemangel). Hver mønts relativt lave værdi gør dog, at brugeren først rigtig vil få noget ud af dette, hvis denne har et stort antal emails i sin indbakke. En løsning på problemet kunne være at sørge for, at afsendere får en notifikation, når modtager vælger at opkræve betaling. Det må formodes, at den afsendte notifikation vil afholde brugere fra at indløse emails, som de reelt *ikke* klassificerer som spam. Fordelen ved at indløse mønten (en mindre økonomisk gevinst) vil nemlig generelt ikke opveje ulempen, nemlig at afsender måske undlader at sende til modtager igen.

I det følgende vil betalingssystemer blive diskuteret med henblik på at finde det mest hensigtsmæssige design, når der tages udgangspunkt i ovenstående ideer.

5.4 Betalingssystemer

Når et nyt betalingssystem skal udvikles til brug i forbindelse med email, er det ønskeligt at systemets egenskaber er hensigtsmæssige med henblik på anvendelsen. Derfor er det nødvendigt med nogle kriterier for at kunne vurdere betalingssystemet. Okamoto og Ohta har opstillet seks kriterier for et *ideelt* betalingssystem [33].

1. **Independence** – De elektroniske penges (e-penge) sikkerhed afhænger ikke af hvor disse befinder sig, og skal endvidere kunne sendes over et computernetværk.
2. **Security** – E-penge kan ikke kopieres og genbruges.
3. **Untracability (Privacy)** – Det er ikke muligt at spore, hvilke handler en bruger har taget del i.
4. **Offline** – Det er muligt for køber og sælger at udføre betaling uden at involvere en central enhed i betalingssystemet.
5. **Transferability** – E-penge kan anvendes mere end én gang, før disse skal indløses.
6. **Divisibility** – En penge-enhed er tilknyttet et beløb. En penge-enhed kan opdeles i flere enheder, med lavere beløb hvis sum er den oprindelige enheds værdi.

Disse kriterier har siden været anvendt i bred udstrækning under udvikling af nye betalingssystemer. Et betalingssystem, der er anvendeligt til email, bør som minimum opfylde de fire første kriterier. Det er nødvendigt at opfylde kriterium 3 for ikke at indskrænke mulighederne i emailsystemet. Blev det muligt at spore brugernes opførsel vha. betalingssystemet, ville email ikke kunne bruges, som vi kender det i dag, hvor det at kunne afsende emails anonymt, tages for givet. Kriterium 1 og 2 er selvskrevne, idet email med påhæftet betaling sendes over internettet, og systemet ikke må kunne misbruges økonomisk. Systemet bør også være offline, idet emailsystemet anvendes globalt af millioner af mennesker i dag. Et online system ville gøre det vanskeligt, at servicere den samme mængde brugere uden væsentlige investeringer i dyr hardware.

Cashette-systemet er et online system, der opfylder meget få af ovennævnte kriterier, idet der er tale om en totalt centraliseret løsning. I Cashette-systemet er det i aller højeste grad muligt at spore brugernes adfærd. Her er det godt nok aldrig andet end email der handles med, men det kan spores, hvem der sender hvad til hvem. Om kriterium 1,5 og 6 er opfyldt er svært at udtale sig om, da overførsel af email såvel som penge styres internt på Cashette's server(e). Sikkerheden i systemet (kriterium 2) kan forventes at være i orden, da brugerne aldrig er i direkte besiddelse af elektroniske penge. Da løsningen er online, vil det dog være begrænset hvor mange brugere systemet kan optage. Derudover vil de fleste nok ikke acceptere, at Cashette kan læse deres emails. Denne løsning skal ikke opfattes som mere end den er: endnu en web-mail med en alternativ form for spam-filtrering.

Skal email kunne benyttes som i dag, og af lige så mange mennesker, kræver dette et offline system, som derudover er sikkert og holder privat information privat (kriterium 1-4). Der findes flere systemer, der baseres på LCP (Lightweight Currency Protocol), som opfylder disse kriterier [35, 36]. I disse systemer er det dog tanken, at emailservere betaler hinanden for *alle* emails og ikke blot spam. Et andet krav, der absolut ikke er

uvæsentligt er, at betaling kan foregå ved kun én overførsel. Altså at en betaling kan vedhæftes en email og sendes til modtager. Det er altså ikke ønskeligt, at en betaling kræver en længere protokol, hvor afsender og modtager udveksler information, som f.eks. matematiske opgaver eller kryptografiske nøgler. Der findes flere betalingsprotokoller, der opfylder kravet om envejskommunikation, herunder PayWord og Micromint [37], se evt. Ricarda Webers markedsanalyse for en oversigt over betalingssystemer [34]. Mange af disse er dog designet til internethandel og kræver at penge indløses efter en enkelt transaktion. De understøtter altså ikke *transferability* (kriterium 5). Derudover har anonymitet (eng.: *privacy)* heller ikke været et designkriterium i hverken Payword eller Micromint. Der findes også løsninger, hvor brugerens anonymitet er sikret og alle kriterier på nær transferability er opfyldt [38, 39]. Disse løsninger er ikke direkte anvendelige til emailbetalingssystemet. Dette skyldes, at systemet skal anvendes i meget stor skala, hvor det at kontakte en pengeserver efter hver emailoverførsel, vil skabe et problematisk knudepunkt. Transferability er altså en vigtig egenskab for systemet.

Et betalingssystem der understøtter alle kriterier på nær *divisibility* er blevet udviklet af Tewari, O'Mahony og Peirce [40]. I dette betalingssystem understøttes altså *tranferability*, men ikke *divisibility*. Skal der vælges mellem disse to egenskaber, må førstnævnte være at foretrække i et system, der skal anvendes i emailsammenhæng. Problemet i Tewari, O'Mahony og Peirce's system er dog, at en tredjepart er involveret ved overførsel af penge, hvilket ikke umiddelbart er foreneligt med et betalingssystem til emailbrug.

I de følgende afsnit diskuteres de problemer, der introduceres med offline-systemer, der understøtter transferability, herunder anvendelse af kopierede elektroniske penge (double spending) og anonymitet.

5.4.1 Double spending

Et problem, der stort set altid skal tages højde for, når elektroniske penge er med i spillet, er, at man ikke kan forhindre at de kopieres. Dette åbner mulighed for såkaldt *double spending*, hvilket betyder, at værdiløse kopier af elektroniske penge benyttes som betalingsmiddel. I nogle betalingssystemer er der mulighed for at opdage double spending samtidig med at betalingen finder sted, altså *inden* køber har fået sin vare. Dette kunne f.eks. ske ved, at det kontrolleres om de involverede mønter i transaktionen findes i en database over brugte mønter. Dette kræver dog en tredjepart, som altid er online, og som står for kontrollen af mønterne. At umuliggøre double spending kræver altså et online betalingssystem.

I offline betalingssystemer er det umiddelbart ikke muligt at forhindre double spending. Her kan det først opdages når transaktionen *er* foregået (*post-detection*) og kopisten *har* fået sin vare. Når det nu ikke er muligt at forhindre *double spending*, kunne man i stedet fjerne incitamentet herfor ved f.eks. at gøre det strafbart. Da det ikke altid er muligt at opkræve bøder i systemet (da kopisterne blot kan undlade at betale og fortsætte svindlen), er eneste effektive strafmulighed udelukkelse fra systemet. For at gøre straffen afskrækkende for potentielle misbrugere, skal det være forbundet med en passende stor omkostning at få lov at benytte betalingssystemet igen.

Brugere af et offline system kan ikke altid alene afgøre, om modtagne mønter er kopierede eller ej. Derfor må en udelukkelse af systemet nødvendigvis betyde, at det ikke længere er muligt at handle med betalingssystemet, uanset om man benytter kopierede mønter eller ej. Da der ikke er involveret en central enhed i forbindelse med en transaktion, kan modtager ikke informeres om, hvorvidt afsender er udelukket eller ej. Derfor skal modtager have denne information på forhånd, f.eks. i form af en liste over samtlige udelukkede brugere (*blacklist*). Dette nødvendiggør dog en mere eller mindre central enhed, som står for opdateringen og udsendelsen af en sådan liste til brugerne af systemet. Det kan ikke undgås, at der vil gå en vis tid, fra en bruger påbegynder svindlen med kopierede penge, til denne opdages og udelukkes. Denne tid forlænges hvis betalingssystemet skal understøtte transferability, da genbrug af mønter vil forlænge den tid der går inden de indløses.

5.4.2 Anonymitet

I begyndelsen af afsnit 5.4 blev nævnt flere systemer, der opfylder de fire første kriterier altså *independence, security, untracability* og *offline* [35, 36]. Skal et system kunne integreres i det eksisterende emailsystem, uden at generere en voldsom mængde ekstra trafik, er det nødvendigt at dette derudover understøtter *transferability* (kriterium 5). En væsentlig detalje er, at dette ikke må ske på bekostning af anonymiteten i systemet. Dette kan f.eks. sikres vha. *secret-splitting* teknikken, hvor ideen er, at anonymiteten i systemet bevares så længe der ikke bliver svindlet. Alternativt kunne anonymitet sikres vha. en opdeling af viden blandt servere. Dette ser vi nærmere på i det følgende.

5.4.2.a Secret-splitting

Som nævnt ovenfor er essensen i secret-splitting teknikken at forblive anonym, så længe man ikke benytter kopierede mønter. Teknikken bygger på simple matematiske relationer, som her vil blive kort forklaret.

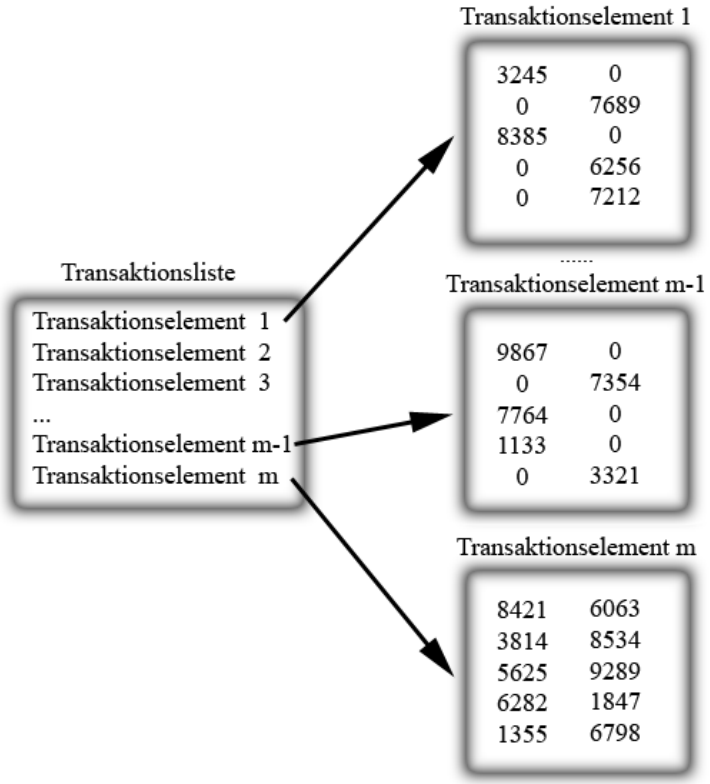
Lad i det følgende *A* være en værdi, der kan bruges til identifikation. Anonymitet afhænger således af, at værdien *A* ikke kan bestemmes. Der gælder følgende egenskab ved den boolske XOR-operator (exclusive or)

Ligning 1 : (A XOR B) XOR B = A

Ved at benytte en tilfældig værdi for *B* kan værdien *A* så at sige opdeles i to værdier, som begge er nødvendige at kende for at bestemme *A*. De to værdier er hhv. *A* **XOR** *B* (som skal udregnes) og *B*. Kendes kun én af værdierne har man ingen mulighed for at bestemme værdien *A*, men kendes begge, kan man blot udføre beregningen angivet i Ligning 1 for at bestemme *A*.

I det følgende vises det hvorledes denne teknik kan benyttes til at sikre anonymitet i et betalingssystem og samtidig afsløre identiteten af møntkopister.

Det antages her, at hver bruger i systemet kan identificeres vha. en firecifret værdi (*UID*). Denne begrænsning er naturligvis ikke et generelt krav, men er valgt for ikke at gøre eksemplet unødigt kompliceret. I hver enkelt elektroniske mønt i systemet integreres en såkaldt transaktionsliste (se Figur 6).



Figur 6 – Transaktionslistens indhold

For hver gang mønten får en ny ejer, skal der tilføjes et transaktionselement til slutningen af denne liste. Til hvert transaktionselement er således knyttet en ejer, som under visse omstændigheder kan identificeres vha. transaktionselementet. Hvert transaktionselement består af en tabel med to kolonner (se Figur 6). Tabellen dannes ved først at generere n tilfældige tal, hvor n er antallet af ønskede rækker i tabellen. De n genererede værdier ($R_1, R_2, \ldots, R_n$) udgør herefter kolonnen til *højre* oppefra og ned. Den *venstre* kolonne dannes ved for hver række at udføre en **XOR**-operation på den nuværende ejers UID og R_i, hvor i angiver rækkenummeret. Hver række i tabellen vil herefter således bestå af to værdier: Til venstre (*UID* **XOR** R_i) og til højre R_i, hvor i angiver rækkenummeret. Enhver række i tabellen kan således benyttes til at bestemme det samme UID (nemlig den nuværende ejers) ved blot at udføre en **XOR**-operation på værdierne i hhv. venstre side og højre side i rækken (jvf. Ligning 1).

I systemet præsenteret af Tewari, O'Mahony og Peirce anvendes *secret-splitting* i forbindelse med et betalingsystem til at opretholde brugernes anonymitet. Dette system vil i det følgende blive kaldt Tewarisystemet [40].

Tewarisystemet

En transaktion i systemet foregår på følgende måde. En betroet tredjepart indgår i transaktionen og kaldes i det følgende for *kontrolenheden*. Der oprettes forbindelse mellem køber, sælger og en kontrolenhed, hvorefter denne verificerer identiteten af køber og sælger, samt at mønten er gyldig. Herefter fjerner (*blinder*) kontrolenheden for hver række i det sidste transaktionselement *tilfældigt* enten værdien i venstresiden eller i højresiden. Transaktionselementet vil herefter ligne transaktionselement *m-1* på Figur 6. Derudover tilføjes et nyt element til transaktionslisten ved brug af den nye ejers UID efter proceduren beskrevet ovenfor (dette element vil ligne transaktionselement *m* på Figur 6). Alle på nær sidste element vil således altid være blinded. Bemærk at det er uhyre vigtigt, at det er *tilfældigt*, hvordan transaktionselementet blindes af kontrolenheden. Vælger køber nemlig at benytte en kopi af mønten til at betale for en anden vare, skal kontrolenheden involveret her med stor sandsynlighed *ikke blinde* på samme måde. Dermed vil man, når de to kopier af samme mønt bliver indløst, kunne sammenholde de to transaktionselementer og kunne finde en venstreside i en række, hvor den korresponderende højreside er at finde i den anden kopi af mønten. Hermed vil kopisten være identificeret, idet man blot udfører en **XOR**-operation på disse. Sandsynligheden for at kopister fældes afhænger af antallet af rækker (i det følgende betegnet *n*) i et transaktionselement. Denne sandsynlighed er $\frac{1}{2^n}$ og det ses dermed, at vælges en høj værdi af *n*, vil sandsynligheden for at to kontrolenheder foretager identiske blindings være meget lav. Til gengæld vil møntens størrelse vokse tilsvarende. Det vil således afhænge af det enkelte betalingssystem hvilken værdi for *n*, som er den optimale.

Som det fremgår af ovenstående, har transaktionslisten til formål at fælde kopister, og secret-splitting-teknikken sikrer anonymitet, med mindre man benytter kopierede mønter. I et system, der benytter secret-splitting-teknikken, er brugerne altså anonyme, så længe de ikke kopierer mønter. Systemet er samtidig skalerbart, sikrer mod møntkopiering og understøtter transferability. Anvendes mere end én kopi af en mønt, vil transaktionslisterne i de kopierede mønter afsløre identiteten af kopisten grundet secret-splitting, og denne kan derefter ekskluderes fra systemet vha. en blacklist. I det følgende afsnit diskuteres mulighederne for en løsning baseret på Tewarisystemet.

Emailbetalingssystem baseret på Tewari

Tawarisystemet er ikke direkte anvendeligt i emailbetalingssystemet, da overførsel af email ikke må afhænge af en tredjepart. Et andet system baseret på ideerne i Tawarisystemet kan dog muligvis anvendes. Formålet med betalingssystemet er at sikre betaling for spam. Det er modtager, der afgør om en email er spam og dermed om denne kræver betaling. Derfor kunne man forestille sig et system, der anvendte en transaktionsliste, som i Tewarisystemet, men hvor det var *modtagers* opgave at blinde sidste element i transaktionslisten. Modtager har en interesse i at afsender (spammeren) ikke kan sende flere kopier af mønten, hvorfor modtager vil gøre dette korrekt. Problemet

er dog her, at hvis det er afsenders opgave at tilføje et transaktionselement med sin egen identitet, og dermed selv vælge de tilfældige tal til secret-splitting, vil denne kunne kopiere mønten og vælge nye tilfældige tal til hver enkelt mønt. På denne måde vil afsenders svindel ikke kunne afsløres. Selvom opgaverne, forbundet med opdatering af transaktionslisten, fordeles på flere brugere, for at sikre at ingen enkelt bruger kan svindle, vil en samarbejdende gruppes møntkopiering aldrig kunne opdages, når kontrolenheden ikke findes. Det er altså ikke umiddelbart muligt, at undvære kontrolenheden i dette betalingssystem, hvilket gør det uegnet til anvendelse i emailsystemet.

5.4.2.b Opdeling af viden

For at understøtte transferability, er det nødvendigt med en transaktionsliste, som beskrevet i forrige afsnit. Anonymitetsproblemet, der følger med transaktionslisten, kan løses ved at *blinde* transaktionslisten indtil mønten kopieres. Dette er dog ikke problemfrit, som beskrevet i foregående afsnit. Derfor betragter vi i stedet en anden løsning, hvor transaktionslisten ikke *blindes*. Det grundlæggende problem er nu, at når mønterne indløses, vil banken kunne se hvilke brugere, der kommunikerer med hvem, ud fra transaktionslisterne. Denne information kan misbruges og en stor del af de potentielle brugere vil sandsynligvis ikke kunne acceptere dette. I følgende scenario opdeles derfor informationen om brugerne i to dele.

Brugerne kan identificeres vha. unikke certifikater. Certifikaterne behøver dog ikke indeholde personlig information (kan indeholde UID eller emailadresser). Indeholder transaktionslisterne sådanne certifikater, vil det være muligt for en blacklistserver at opretholde en blackliste, uden at kende personernes identitet. De elektroniske penge håndteres af pengeservere, der administreres af bankerne. Disse vil nødvendigvis kende brugernes identitet. For at en pengeserver ikke skal kunne udlede information om trafikken i systemet, må denne ikke kende transaktionslisterne. Derfor krypteres disse inden mønter indløses, med blacklistserverens offentlige nøgle. Hermed vil blacklistserveren stadig kunne udelukke brugere i tilfælde af kopiering. Information om trafikken i systemet er altså fordelt mellem pengeserver og blacklistserver. Selvom blacklistserveren kan udlede nogen information om trafikken i systemet, vil denne udelukkende vide hvilke certifikater der har været involveret og dette kun for kopierede mønter. Samtidig kan pengeserveren sammenkæde certifikater med personer, men har ingen information om trafikken. Opretholdes denne opdeling af viden, vil hverken blacklistserver eller pengeserver kunne udlede brugbar information om brugernes opførsel.

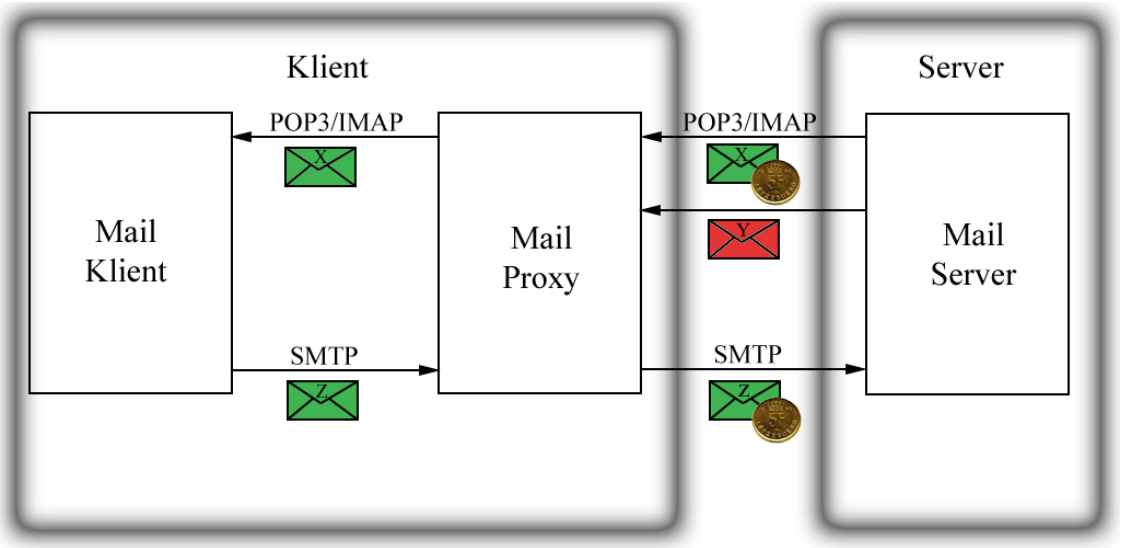
Selvom banken ikke kender transaktionslisterne for indløste mønter, vil denne stadig kende identiteten på både køber og indløser af en given mønt. Problemet er, at banken i begrænset omfang herved vil få information om trafikken i systemet. Problemets størrelse kan reduceres ved at øge længden på transaktionslisterne. Sikres det at pengeserveren ikke kender serienummeret ved udstedelse af en given mønt, vil problemet dog være helt forsvundet. Dette kan opnås ved at anvende blinde signaturer ved udstedelse af penge, som beskrevet i afsnit 4.4.

At opdele informationen om brugerne kan altså sikre brugernes anonymitet, så længe bankserver og blacklistserver holdes adskilt. Dette er dog en ulempe ved systemet, da dette i praksis ikke er en triviel opgave. Til gengæld findes der ikke umiddelbart en måde, hvorpå brugerne kan misbruge systemet økonomisk, som i tilfældet med løsningen baseret på secret-splitting-teknikken. En anden ulempe ved begge løsningsforslag er introduktionen af en blacklistserver. Denne vil blive et knudepunkt i systemet og det er vigtigt for skalerbarheden, at dennes arbejdsbyrde bliver minimal.

Vi har nu diskuteret to grundlæggende ideer til systemstrukturer som giver mulighed for transferability og samtidig sikrer brugernes anonymitet. Der vil nu blive opstillet en kravspecifikation på baggrund af denne diskussion.

5.5 Kravspecifikation

Betalingsløsningen skal bygge på det eksisterende emailsystem. Systemet implementeres på klientniveau, således at dette er uafhængigt af mailserveropsætning. Der skal altså udvikles en såkaldt proxy-løsning, til at håndtere emails med vedhæftet betaling (se Figur 7).



Figur 7 – Strukturen efter indførelse af emailproxyen. Mailklienten kommunikerer gennem proxyen med mailserveren. Proxyen vedhæfter elektronisk betaling (en "e-mønt") på afsendte emails, og kontrollerer og fjerner betaling på modtaget email. Den røde kuvert symboliserer email uden vedhæftet betaling, der frasorteres (kunne være spam), og de grønne kuverter symboliserer legitim email.

Som det fremgår af figuren, installeres et proxyprogram på den enkelte brugers maskine. Afsendelse og modtagelse af email foregår altså fortsat vha. den enkelte brugers foretrukne emailklient. I det følgende vil kravene til systemet blive opstillet.

Funktionelle krav:

- Proxyprogrammet skal uden at brugeren involveres (*transparency*) automatisk kunne:
  - Vedhæfte mønt(er) på afsendte emails
  - Opsnappe og gemme gyldige mønter på modtagne emails
  - Frasortere emails uden vedhæftet gyldig e-mønt

- Brugeren skal have mulighed for at udvælge afsenderadresser, hvorfra denne kan modtage emails uden vedhæftede mønter (whitelist).
- Proxyprogrammet skal understøtte de mest almindelige protokoller til afsendelse og modtagelse af email.

- Proxyprogrammet skal tilbyde sikker lokal opbevaring af brugerens elektroniske penge. De lokalt opbevarede penge må ikke gå tabt ved programnedbrud eller uforudset nedlukning.
- Det skal være muligt at opkræve den vedhæftede betaling for enhver modtaget email.
- Opkræver brugeren ikke betalingen på en email indenfor en fastsat tidsgrænse, skal systemet automatisk returnere betalingen til afsender.
- Det skal være muligt for brugeren at hæve og indsætte elektroniske penge på en af systemets pengeservere.
- Proxyprogrammet skal kunne kommunikere sikkert med systemets pengeservere.

Ikke-funktionelle krav:

- Proxyprogrammet skal være brugervenligt, da det skal kunne bruges af alle.
- Systemet skal være platformuafhængigt og så vidt muligt være kompatibelt med alle eksisterende e-mail klienter.
- Systemet skal baseres på det eksisterende emailsystem. Det skal altså ikke være nødvendigt at modificere det eksisterende emailsystem.
- Systemet skal være udgiftsneutralt for brugere, der ikke sender spammail.
- Spammere skal straffes økonomisk og spam-modtagere belønnes tilsvarende.
- Systemet skal derudover have følgende egenskaber:
    - **Independence:** De elektroniske penges sikkerhed skal ikke afhænge af speciel hardware og det skal være muligt at transportere penge over computernetværk.
    - **Security:** At misbruge systemet økonomisk skal være med ingen eller meget begrænset gevinst.
    - **Privacy:** Brugerne af systemet skal være anonyme og deres opførsel skal ikke kunne spores vha. det økonomiske system.
    - **Offline & Scalable:** Det skal være muligt at sende og modtage email uden at involvere en central server. Systemet skal altså være skalerbart.
    - **Integrity:** Integriteten af de elektroniske penge skal sikres.
    - **Availability:** Systemet skal ikke introducere nye begrænsninger på hvornår afsendelse og modtagelse af email kan finde sted.
    - **Transferability:** E-penge skal kunne benyttes mere end én gang, før disse skal indløses i banken.

5.6 Trusler

Et af kravene i kravspecifikationen er, at systemet skal være *sikkert*. For at kunne opnå dette, er det dog nødvendigt først at definere præcist hvad, der menes med *sikkert*. Derfor identificeres de potentielle trusler imod systemet i det følgende.

Kryptoanalyse

En angriber bryder sikkerheden i de benyttede kryptografiske algoritmer eller hashfunktioner. Dette kunne gøres ved at finde sårbarheder i disse, f.eks. ved 'known plain-text' eller brute-force angreb.

Reverse engineering

En angriber udfører *reverse engineering* på proxyprogrammet (eller serverapplikationer) og modificerer og misbruger herefter koden. Dette kunne gøres som forsøg på at lave en udgave af proxyprogrammet, som kan sende emails med falske penge, eller give mulighed for afsendelse af spam uden omkostninger for angriberen. Det må formodes, at så snart proxyprogrammet er frigivet, vil angribere forsøge at modificere dette på alle tænkelige måder, hvorfor sikkerheden ikke må afhænge af proxyprogrammets integritet. Dette svarer til det velkendte princip indenfor kryptografien, hvor det er essentielt at sikkerheden udelukkende afhænger af hemmeligholdelsen af de anvendte nøgler og aldrig af angriberens kendskab til de benyttede algoritmer (*Kerckhoff's princip* [42]). Det antages altså, at en angriber har adgang til hele programmet.

5.6.1 Økonomisk misbrug

Da systemet er baseret på 'rigtige' penge, kan det ikke undgås, at der vil forekomme forsøg på økonomisk misbrug. Herunder gennemgås forskellige typer af økonomisk misbrug, som systemet skal sikres imod.

Fabrikation af penge

En angriber får udskilt pengegenererings-mekanismen eller får på anden måde mulighed for at generere ubegrænsede mængder e-penge. Hermed vil denne kunne sende spammail uden at blive straffet økonomisk.

Kopiering af penge

Angriberen finder en metode til at kopiere mønter og benytter disse kopier til at sende emails (evt. spammails) eller veksler disse til 'rigtige' penge.

Hamstring af mønter

En eller flere brugere samler på mønter. Til sidst vil de resterende brugere ikke have flere mønter tilbage. I et lukket betalingssystem, hvor der findes et fast antal mønter, vil dette være et problem. I et åbent system, hvor det er muligt at generere flere mønter efter behov, vil dette problem ikke eksistere.

Grådige brugere

En bruger af systemet vælger at indløse alle modtagne mønter, uanset om der er tale om spam eller ej. Hvis alle brugere af systemet gjorde dette, ville systemet blot svare til et system, hvor det koster penge at sende emails, men hvor der samtidig tjenes når man modtager. Dette ville stadig være dårlig nyt for spammere, men ville tilgodese brugere, som modtager flere emails end de sender. Der findes betalingssystemer, der håndterer grådige brugere i lukkede betalingssystemer [45]. Men da systemet, der betragtes i denne sammenhæng, er åbent, har grådige brugere kun indflydelse på de der kontakter dem og ikke på hele systemet.

Møntfiskeri

Godtroende brugere lokkes til at sende emails til angriberens email-adresse. Se evt. afsnit 5.3.1 for en række metoder, hvorved dette kan opnås. Så snart angriberen modtager emails indløses disse. Der 'fiskes' altså efter andre brugeres mønter med forskellige former for 'madding'.

Tyveri af mønter

En angriber stjæler de e-penge, der er opbevaret hos klienten. Dette kunne evt. ske vha. et hackerangreb, vira eller trojanske heste, som giver angriberen adgang til klientens e-penge. Alternativt kan tyveri ske ved, at emails med vedhæftede mønter opsnappes på vej fra afsender til modtager.

Salg af kopierede mønter

En angriber sælger kopier af en modtaget mønt til andre, som derefter indløser eller bruger mønterne.

Modifikation af mønter

En angriber forsøger at ændre møntens serienummer eller værdi, for at undgå afsløring af dennes kopiering eller for at kunne sende emails uden at betale for dette.

5.6.2 Netværkstrusler

Så snart man har at gøre med et netværksbaseret system, er der risiko for at netværksprotokollerne bliver udsat for angreb.

Man in the middle

En angriber opfanger mails på vejen fra afsender til modtager. Angriberen kan så vælge, enten at forsøge at benytte de vedhæftede e-penge til at sende email gratis eller at indløse pengene. Alternativt kan angriberen forsøge at kontakte mailproxyen på en brugers maskine direkte i håb om, at kunne sende emails gratis via denne (hvor det så er offeret der kommer til at hæfte for de afsendte emails). Derudover kan en angriber forsøge, at misbruge de protokoller som benyttes, når brugere kommunikerer med systemets servere.

Eavesdropping

En angriber lytter på al trafik til og fra en bruger af systemet. Denne information kan angriberen udnytte til f.eks. at sende emails på offerets regning eller få denne smidt ud af systemet (blacklistet).

Modifikation

En angriber modificerer indholdet af emailbeskeder eller andre netværkspakker i systemet. En angriber kan f.eks. ændre alle udgående emails en bruger sender ved at fjerne al indholdet og tilføje en spam-meddelelse i stedet. Hermed vil alle brugerens mails højst sandsynligt blive betragtet som spam og denne vil derfor miste sine mønter, mens angriberen ikke vil lide noget tab.

Replay

En angriber opfanger emails og forsøger at sende et stort antal kopier til modtager. Herved vil modtager muligvis forsøge at indløse mange af disse e-mønter. Angriberen vil derefter kunne opnå at afsender kommer til at stå i dårligt lys overfor modtager, eller i værste fald at afsender blacklistes, hvis denne får skylden for kopieringen. Replay-angreb kan også forekomme i forbindelse med angreb på de protokoller, der benyttes når brugere kontakter systemets servere.

Denial of Service

Denial of service (DOS): En angriber belaster en server (f.eks. blacklistserveren) så voldsomt at denne ikke kan behandle normal trafik. Er der tale om blacklistserveren, vil angriberen kunne sende spammails med kopierede mønter, så længe denne er ude af normal drift.

Distribueret DOS (DDOS): Samme risici som ved regulære DOS angreb, dog er effektiviteten af DDOS angreb større, da flere maskiner deltager i forsøget på at lægge serveren ned.

5.6.3 Fysiske trusler

Da der i systemet vil findes en række servere af forskellig art, skal det vurderes i hvor høj grad disse skal sikres mod fysiske trusler. Der kan opstå brand i rummet hvor serverne befinder sig, serverne kan blive stjålet eller ødelagt på anden vis. Servernes placering skal overvejes nøje, med disse risici for øje. I hvor høj grad serverne skal sikres fysisk, afhænger naturligvis af hvilken rolle, den enkelte server har i systemet. Er der f.eks. tale om en central server, hvis tilstand er kritisk for alle systemets brugere, skal det fysiske sikkerhedsniveau for denne være meget højt. Er der derimod tale om en lang række servere placeret forskellige steder i verden, hvor systemet blot kræver en enkelt funktionsdygtig server, kan sikkerhedsniveauet sænkes uden at dette går ud over den overordnede sikkerhed.

En anden ikke uvæsentlig faktor i vurderingen af den fysiske sikkerhed er risikoen for ulovlig indtrængen ved højlys dag (social engineering). Det kan f.eks. ske ved at udefrakommende trænger ind i et firma ved f.eks. at udgive sig for at være en medarbejder. At uautoriserede kan få adgang til server- eller klientmaskiner i systemet kan have store konsekvenser og denne risiko skal derfor tages alvorligt.

6 Design

6.1 Systemstruktur

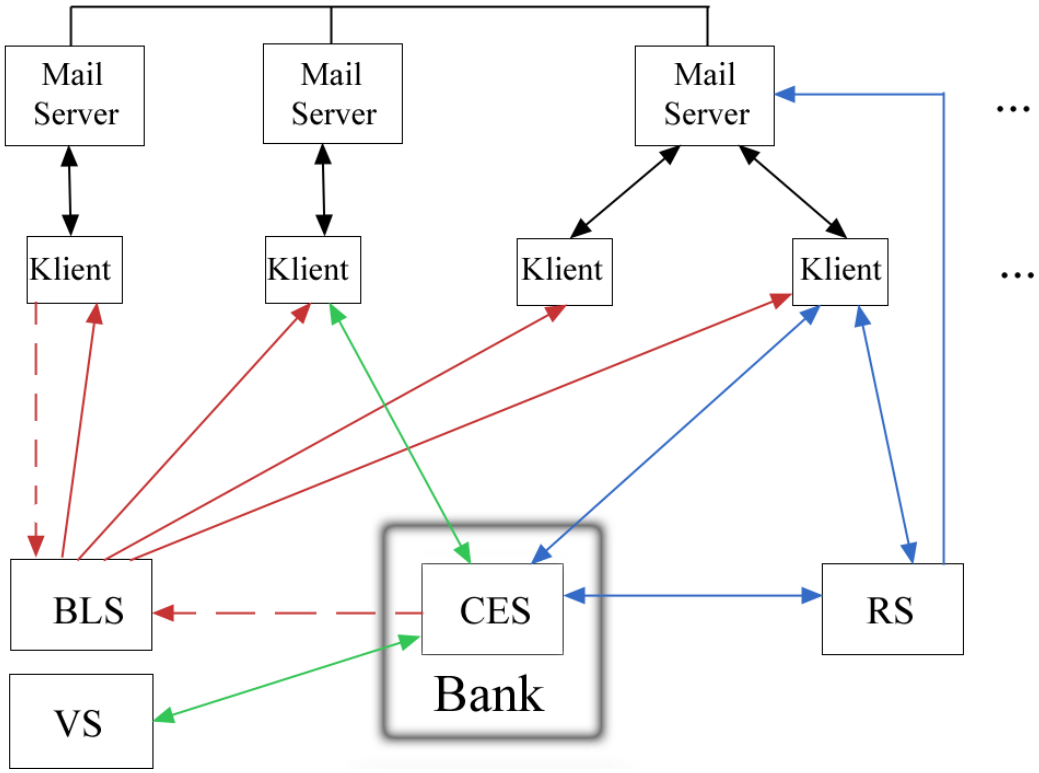
I analyse-afsnittet blev det gjort klart, at transferability ikke kan understøttes problemfrit. Systemet vil blive baseret på løsningen, hvor brugernes anonymitet bliver bevaret ved at opdele information om disse mellem to servere. Secret-splitting-teknikken kan ikke anvendes uden en kontrolenhed som tredjepart, hvilket vi ikke kan acceptere. Der designes således et system, som opfylder kriterium 1-5 fra afsnit 5.4, dvs. alle kriterier på nær *divisibility*. Systemet har vi valgt at kalde ”SpamCash”, da spam kan indløses til kontanter (cash) i systemet.

6.1.1 Infrastruktur

I systemet findes klientapplikationer og flere forskellige servere. Følgende punktopstilling viser hvilke applikationer, der findes i systemet. Forkortelserne indført her vil blive benyttet i resten af rapporten.

- Klient
- Pengeserver (Currency Exchange Server, CES)
- Verifikationsserver (VS)
- Blacklistserver (BLS)
- Registreringsserver (RS)

Der kan findes flere pengeservere i systemet. Disse står for udstedelse og indløsning af e-penge, og det er ideen, at disse i et endeligt system skal administreres af banker og integreres i et netbankinterface, når systemet skal implementeres. I det følgende antages det, at der kun er én enkelt pengeserver for at lette forståelsen af systemets design. Systemet kan dog umiddelbart udvides til at understøtte flere pengeservere. De tre andre serveres opgaver kan i princippet udføres af en enkelt server. Derved vil information om brugerne stadig være opdelt mellem denne server og pengeserveren. For at gøre systemet skalerbart, på trods af blacklistserverens centrale rolle, er det dog valgt at fordele opgaverne på følgende måde. Verifikationsservere har den simple opgave at verificere en mønts gyldighed for pengeserveren. Pengeserveren kan ikke stå for dette, hvis brugernes anonymitet skal bevares (uddybes senere). Blacklistserveren står for opretholdelsen af en blacklist, som klienterne bruger til emailfiltrering. Der findes kun én blacklistserver i systemet, men i et endeligt system kan dennes arbejde umiddelbart distribueres til flere servere, der tilsammen opretholder en fælles blackliste. Registreringsserveren står for oprettelse af nye brugere ved udstedelse af certifikater, og sikrer i samarbejde med en pengeserver, at der betales for disse. I et senere afsnit vil detaljer omkring hver enkelt del blive gennemgået, men for at give et overblik over kommunikationen i systemet vises først Figur 8:



Figur 8 – Oversigt over kommunikationen i systemet. De blå pile symboliserer kommunikation ved oprettelse i systemet. De røde stiplede pile symboliserer indrapportering af kopierede mønter. De røde pile viser udsendelse af blacklist. De grønne pile symboliserer, at en klient hæver og indsætter penge, og at pengeserveren efterfølgende kan kontrollere disse ved at benytte en verifikationsserver. De sorte pile viser almindelig emailkommunikation (mailudveksling).

I resten af designafsnittet vil først centrale aspekter af systemets design blive diskuteret herunder autentifikation, e-penge, blacklisting og anonymitet. Derefter vil der blive gået mere i detaljer vha. use-cases og sekvensdiagrammer. Til sidst vil der blive redegjort for sikkerheden i systemet.

6.2 Autentifikation

6.2.1 Oprettelse

Når en bruger ønsker at oprette sig i SpamCash-systemet, skal en eller flere af dennes emailadresser tilmeldes. Emailadresserne anvendes i forbindelse med det personlige certifikat, der udstedes til brugeren ved oprettelse. I dette certifikat indskrives en hashværdi af brugerens emailadresser. Emailadresserne hashes af anonymitetsgrunde (beskrives nærmere i afsnit 6.5). Ved oprettelse sikres det, at enhver emailadresse findes i ét og kun ét certifikat. Dette er med til at sikre systemet mod misbrug, idet der på denne måde opnås en direkte sammenhæng mellem emailadresser og certifikater. Dette løser et distributionsproblem, idet afsender ikke behøver kende modtagers certifikat, men blot dennes emailadresse, for at kunne sende betalings-emails til denne. I det følgende beskrives hvordan certifikaterne udstedes og anvendes i SpamCash-systemet, og senere

(bl.a. i afsnit 6.3.2.a) beskrives det, hvordan email-hashværdierne i certifikaterne benyttes til at sikre mod misbrug.

6.2.1.a Traditionelt certifikatsystem

Traditionelt benyttes certifikater f.eks. til at opnå sikker kommunikation over usikre netværk og distribueres på følgende måde. Certifikatudstedelse håndteres af en central certifikatautoritet (CA), der også sørger for at fjerne ugyldige certifikater (*revocation*). Alle brugere i systemet vælger at stole på CA'en. Når en bruger ønsker et certifikat udstedt, genererer denne et asymmetrisk nøglepar. Herefter kontaktes CA'en for at få denne til at underskrive den offentlige nøgle og danne et certifikat. Hvis brugeren kan autoriseres (og evt. betaler for certifikatet), udsteder CA'en et certifikat til brugeren. Dette indeholder information, der identificerer brugeren, samt CA'ens signatur af brugerens offentlige nøgle. CA'er sørger herefter for distribution af certifikatet, så længe dette er gyldigt. Ophører den gensidige tillid mellem CA og brugeren, vil CA'en fjerne certifikatet fra listen over gyldige certifikater.

Sikker kommunikation kan herefter foregå således. Når en bruger, A, ønsker at kommunikere sikkert med en anden bruger, B, kontakter A CA'en for at få B's certifikat. Herefter kan A oprette en sikker forbindelse til B ved at kryptere en symmetrisk sessionsnøgle med B's offentlige nøgle. Denne symmetriske nøgle benyttes efterfølgende til at kryptere alle beskeder mellem A og B. På denne måde ved A, at det vitterlig *er* B der kontaktes og at CA'en i øjeblikket stoler på B, hvilket kan give A grund til at stole på B.

6.2.1.b SpamCash certifikatsystem

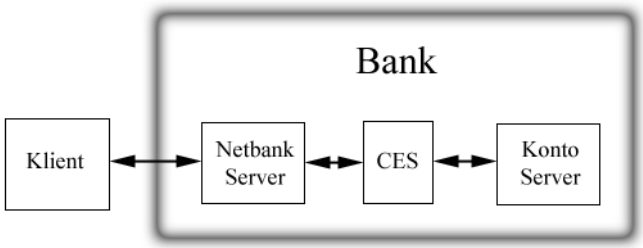
I SpamCash-systemet benyttes certifikater til at opnå sikker kommunikation med serverne på traditionel vis. Distribution af servernes certifikater foregår ligeledes på traditionel vis. Klienter er ligesom serverne i besiddelse af certifikater, men disse distribueres ikke på traditionel vis. I SpamCash systemet er det nemlig ikke nødvendigt at opnå *tillid* til andre brugere. Det eneste der er nødvendigt er, at man er sikker på, at disse har *betalt* for oprettelsen. For at sikre at dette er tilfældet foregår oprettelse i systemet på følgende måde. Først kontaktes registreringsserveren, hvor der ikke kræves autentifikation. Registreringsserveren sikrer sig dog, at brugeren har betalt for oprettelse hos en pengeserver vha. protokollen, der beskrives i afsnit 6.8.5. Det eneste registreringsserveren ellers skal kontrollere, før denne udsteder et certifikat til brugeren er, at den (eller de) emailadresser, der ønskes registreret til brug i systemet, rent faktisk tilhører brugeren og at disse ikke allerede er registreret. At emailadressen tilhører brugeren sikres ved at afsende en email til emailadressen. I denne email vil brugeren skulle trykke på et aktiverings-link. Når det registreres at linket har været aktiveret, kan registreringsproceduren afsluttes.

Registreringsserveren *udsteder* altså udelukkende certifikater. Tilbagetrækning af disse (*revocation*) står blacklistserveren for ved at opretholde en blackliste over certifikater, der ikke længere er gyldige. Disse to servere udgør altså tilsammen SpamCash-systemets certifikatmekanisme. Systemet sikrer at certifikater, der misbruges bliver blacklistet. Modtages derfor email sendt med et certifikat, der ikke er på blacklisten, opfattes dette

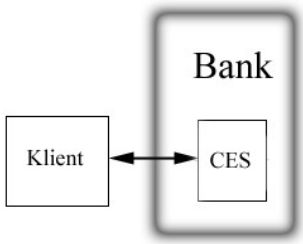
som gyldigt, *uden* at kontakte en certifikatautoritet. At det ikke er nødvendigt at kontakte en certifikatautoritet hver gang en email modtages, kræver dog at blacklisten er opdateret. Opdateres denne hver gang en email modtages, vil der genereres samme trafik som ved direkte kontakt til en certifikat-autoritet. Der vil i afsnit 8.2 blive argumenteret for, hvorfor det ikke er nødvendigt at opdatere blacklisten så ofte, og dermed at servernes arbejde kan begrænses væsentligt, ved at benytte ovenstående model i stedet for den traditionelle.

6.2.2 Pengeoverførsler

Pengeserveren skal give mulighed for at hæve penge og indsætte penge. I det endelige system skal autentifikation håndteres af en netbankserver. Sammenhængen pengeserveren skal indgå i kan ses på Figur 9. Her befinder pengeserveren sig i banken, hvor en netbankserver på den ene side står for autentifikation af og kommunikation med brugeren via et webinterface og bankens centrale kontoserver på den anden side står for håndtering af kundernes konti.



Figur 9 – Pengeserverens rolle i det endelige system.



Figur 10 – Pengeserverens rolle i det simplificerede system.

Da systemet ikke integreres med netbank i denne rapport, ser situationen i stedet ud som på Figur 10. Her er ingen netbankserver til at håndtere autentifikation og ingen kontoserver til at håndtere kundernes konti. For at demonstrere, at systemet kan virke i praksis, indføres derfor en simpel udgave af denne funktionalitet i pengeserveren. Kontakt til pengeserveren kræver i systemet blot angivelse af korrekt brugernavn og password. Altså en relativt svag autentifikation, der reelt set ikke er stærk nok til overførsel af penge. I det endelige system vil dette dog, som nævnt, blive erstattet af netbankens autentifikationsmekanisme, som må antages at være tilstrækkelig sikker.

6.3 Design af elektroniske penge

I dette afsnit vil det blive beskrevet hvorledes de elektroniske penge designes. Først vil mekanismen, der benyttes til overførsel af penge blive beskrevet. Herefter vil møntens struktur og indhold blive fastlagt. Til sidst vil vi redegøre for, hvordan mønten er sikret mod misbrug.

6.3.1 Betaling

I afsnit 5.2 blev den grundlæggende ide præsenteret og herunder blev mekanismen, der benyttes til betaling for email, beskrevet. En betaling vedhæftes altså enhver email og modtager kan vælge at indkassere betalingen indenfor et vist tidsrum. Overskrides denne tidsfrist, vil betalingen blive tilbageført til afsender. Hvordan dette gøres konkret, vil blive klargjort i det følgende.

For at en betaling (en 'mønt') kan tilbageføres fra modtager til afsender, *uden* at skulle sende en email til afsender, er det nødvendigt, at afsender beholder en kopi af mønten ved afsendelse. Hvis modtager ønsker at beholde mønten, afsendes en email fra dennes proxy til afsenders proxy med besked om at destruere møntkopien. For at undgå at afsender skal danne en kopi, kunne det være nærliggende blot at sende mønten tilbage når der ikke er tale om spam. Antaget at systemet virker efter hensigten og begrænser spam, vil langt de fleste emails ikke være spam. Derfor er det hensigtsmæssigt kun at sende en email tilbage, når der *er* tale om spam. Derfor benyttes denne løsning.

For i praksis at kunne håndtere (og diskutere) denne overførsel af mønter, indføres følgende terminologi og regelsæt. En mønt kan befinde sig i tre tilstande, som vi har valgt at kalde: A, B og C. Følgende karakteriserer de tre tilstande:

A: Mønten har altid værdi
B: Mønten har værdi *indtil* et vist udløbstidspunkt
C: Mønten har værdi *fra* den tilhørende B-mønts udløbstidspunkt

Det er let at ændre tilstanden for en mønt. I systemet er det dog kun tilladt at behandle mønttilstande efter følgende tre regler:

1. A → B, C
2. B → A
3. C → A

Kort forklaret omdannes efter første regel en A-mønt til en B- *samt* en C-mønt. Efter de 2 sidste regler omdannes hhv. en B- og en C-mønt til en A-mønt. Kun under visse omstændigheder må ovenstående tre regler benyttes.

Når en mønt købes af en pengeserver, er denne i tilstand A. Når en email skal sendes, må regel 1 benyttes: A-mønten omdannes til en B-mønt, som påhæftes emailen og en C-mønt som gemmes. Opdelingen af A-mønten giver mulighed for at returnere værdien til afsender uden reelt at sende denne retur. Dette gøres ved at modtager destruerer B-mønten på udløbstidspunktet og afsender samtidig omdanner sin C-mønt til en A-mønt (regel 3). Hvis modtager kategoriserer emailen som spam, omdanner denne sin B-mønt til en A-mønt (regel 2) og giver afsender besked om, at dennes C-mønt skal destrueres. Afsender får denne besked, ved at modtager genererer og sender en såkaldt *notifikationsmail* til denne. Modtagelse af en notifikationsmail vil altså få brugeren til at slette sin kopi af mønten. Dette kunne forsøges misbrugt af en angriber, hvorfor det er nødvendigt at sikre, at notifikationsmailen ikke kan genereres af en sådan. Dette er gjort

ved at lade notifikationsmailen indeholde modtagers signatur af møntens serienummer samt afsenders emailadresse. Dermed sikres at ingen andre end modtager kan sende en gyldig notifikationsmail. Indholdet af det signerede (serienummer, afsenderadresse) sikrer endvidere, at notifikationsmailen kun kan bruges til at slette den *ene* mønt hos den *ene* bruger. Dermed beskyttes mod angreb, hvor en man-in-the-middle opsnapper notifikationsmails og forsøger at misbruge disse til f.eks. at få slettet andre mønter eller samme mønt hos en senere ejer.

Skulle det ske, at notifikationsmailen ikke når frem, vil modtager og afsender begge omdanne hhv. deres B- og C-mønt til en A-mønt. Dermed vil de hver især kunne sende email med to identiske kopier af samme mønt. Dette tilfælde vil (desværre) ikke kunne skelnes fra en møntkopiering foretaget af afsender. Dette vil betyde, at (enkelte) uskyldige brugere vil blive blacklistet. For at undgå dette kunne det antages at mindst 99% af disse emails når frem, og dermed lade systemet tolerere møntkopiering i et omfang, der ligger under 1%.

En alternativ løsning på dette problem kunne være, at der *ikke* afsendes notifikationsmails. I stedet kontrollerer afsender, når mønten er udløbet, om denne er blevet indløst ved at kontakte banken. Hvis ikke, krediteres afsenders konto. Et af problemerne med denne løsning er dog, at banken får kendskab til, at brugeren har brugt den pågældende mønt til at sende med. Dette har betydning for brugernes anonymitet, som beskrevet i (slutningen af) afsnit 5.4.2.b. Derudover går der længere tid, inden klienten får besked om, at mønten blev indkasseret af modtager. Pengeserverne vil desuden blive belastet mere, da de så skal udføre arbejde som ellers påtages af klienterne. Derfor benyttes dette alternativ ikke.

At bryde ovenstående regler for omdannelse af mønter er relativt let og møntkopiering, med henblik på misbrug, kan let foretages med udgangspunkt i ovenstående. Møntkopiering vil dog under alle omstændigheder være muligt (det er f.eks. let at kopiere de lokalt opbevarede penge på harddisken). Hvis de, på den ene eller anden måde, kopierede mønter benyttes til afsendelse af email, vil dette blive håndteret af systemet. Ovenstående møntfunktionalitet benyttes blot for at gøre det lettere at holde styr på de forskellige 'stadier' e-mønterne kan befinde sig i og vil altså *ikke* introducere nye sikkerhedsproblemer i systemet.

6.3.2 Møntstruktur

Enhver mønt i SpamCash-systemet indeholder:

- Værdi
- Serienummer (SN)
- Udløbsdato
- Tilladte antal ejerskift (TE)
- Én signatur af beløb, SN, udløbsdato og TE dannet af den udstedende pengeserver.
- Transaktionsliste

Serienummeret benyttes til detektion af møntkopiering. TE angiver det maksimale antal gange den enkelte mønt må skifte ejer, før den skal indløses. Da der kun tilføjes transaktionselementer til mønten ved hvert ejerskift, er TE således med til at sikre et øvre loft for størrelsen på den enkelte mønt. Samtidig sikres det, at kopierede mønter gennemsnitligt bliver detekteret hurtigere, end hvis det var tilladt at mønterne kunne skifte ejer et ubegrænset antal gange. Udløbsdato og værdi fastsættes af den pågældende pengeserver. Udløbsdatoen kan f.eks. sættes til den dato, hvor pengeserveren af sikkerhedsmæssige grunde udskifter sit nøglepar. Signaturen i mønten dannes ved først at beregne en hashværdi af værdi, SN, udløbsdato og TE og herefter signere denne hashværdi. Pengeserverens signatur er nødvendig for at sikre, at ingen af ovennævnte værdier kan ændres udetekteret. Det sidste element i mønten, transaktionslisten, indgår i mønten for at kunne understøtte transferability og samtidig sikre imod misbrug. Denne indgår ikke i pengeserverens signatur, da listen ændres når mønten bruges. Hvordan listen sammenkædes med resten af mønten og sikres mod misbrug beskrives i følgende afsnit.

6.3.2.a Transaktionslisten

Transaktionslisten er en liste, der viser hvilke overførsler mønten har indgået i. Listen angiver altså, hvem der ejer mønten og de tidligere ejere. Når mønten udstedes, vil denne indeholde en transaktionsliste med netop ét element. Dette element angiver, at mønten har været overført fra pengeserveren til køber. Elementet indeholder pengeserverens signatur af modtagers (købers) emailadresse-hash samt serienummeret på mønten. Samme type signatur dannes, når mønten overføres mellem to brugere af systemet, og vi har derfor givet denne betegnelsen *overførselssignatur*. Herunder ses strukturen af en overførselssignatur, hvor Sig_{user} skal udskiftes med $Sig_{CES,}$ hvis der er tale om første element i transaktionslisten:

Overførselssignatur (Type 1):
- Sig_{user}(Hash(Serienummer på mønten, modtagers emailadresse-hash))

Da serienummeret indgår i signaturen, kan modtager bevise, at afsender har afsendt netop den pågældende mønt (*non-repudiation*). Derudover kan transaktionslistens første element ikke adskilles fra resten af mønten, da serienummeret indgår, og det derudover kræves at signaturen i første element er dannet af en pengeserver. I det følgende beskrives hvordan de resterende elementer i transaktionslisten bindes sammen med det første element.

Før modtager (eller køber) kan benytte e-mønten til at sende email, kontrollerer denne transaktionslistens integritet og tilføjer et element bestående af sit certifikat samt en signatur af en hashværdi af hele listen. Denne signatur har vi givet betegnelsen *integritetssignatur*:

Integritetssignatur (Type 2):
- Sig_{user}(Hash(Transaktionslisten))
- $Cert_{user}$

Denne signatur sikrer, at modtager ikke senere kan påstå, at transaktionlistens integritet ikke var bevaret, da denne overtog mønten. Første gang en integritetssignatur tilføjes transaktionslisten er, når mønten udstedes og overføres fra pengeserveren til køber. Her kontrollerer køber transaktionslistens integritet og signerer listen (som i dette tilfælde altså kun indeholder ét element). Denne signatur tilføjes så til transaktionslisten sammen med købers certifikat. For hvert ejerskift vil der således blive tilføjet 2 elementer til transaktionslisten. Sættes det tilladte antal ejerskift f.eks. til 10, vil det betyde, at en liste maksimalt vil kunne indeholde 20 elementer.

Som nævnt ovenfor findes to grundlæggende typer af elementer i transaktionslisten. I Tabel 2 har vi angivet strukturen af transaktionslisten.

Indeks	**Transaktionsliste**	**Elementtype**
1	$Sig_{CES}(SN,Email_A)$	1
2	Sig_A (Transaktionsliste$_{[1-1]}$), $Cert_A$	2
3	$Sig_A(SN,Email_B)$	1
4	Sig_B(Transaktionsliste$_{[1-3]}$), $Cert_B$	2
5	$Sig_B(SN,Email_C)$	1
6	Sig_C(Transaktionsliste$_{[1-5]}$), $Cert_C$	2
7	$Sig_C(SN,Email_D)$	1
8	Sig_D(Transaktionsliste$_{[1-7]}$), $Cert_D$	2
...	...	...

Tabel 2 – Her ses strukturen af transaktionslisten. Notationen Sig_X(*data*) betyder, at en hashværdi af *data* er signeret med den private nøgle tilhørende X. Transaktionsliste$_{[x-y]}$ angiver et udsnit af transaktionslisten, nemlig fra og med indeks *x* til og med indeks *y*. $Email_X$ angiver en hashværdi af en emailadresse, som tilhører X.

Er det tilfældet at ethvert element er bundet sammen med det forrige element, vil listens integritet være sikret. Da listen alternerer mellem to typer af elementer, skal elementer af den ene type være bundet til elementer af den anden type og omvendt. At dette er tilfældet ses ved at bemærke følgende egenskaber ved transaktionslisten:

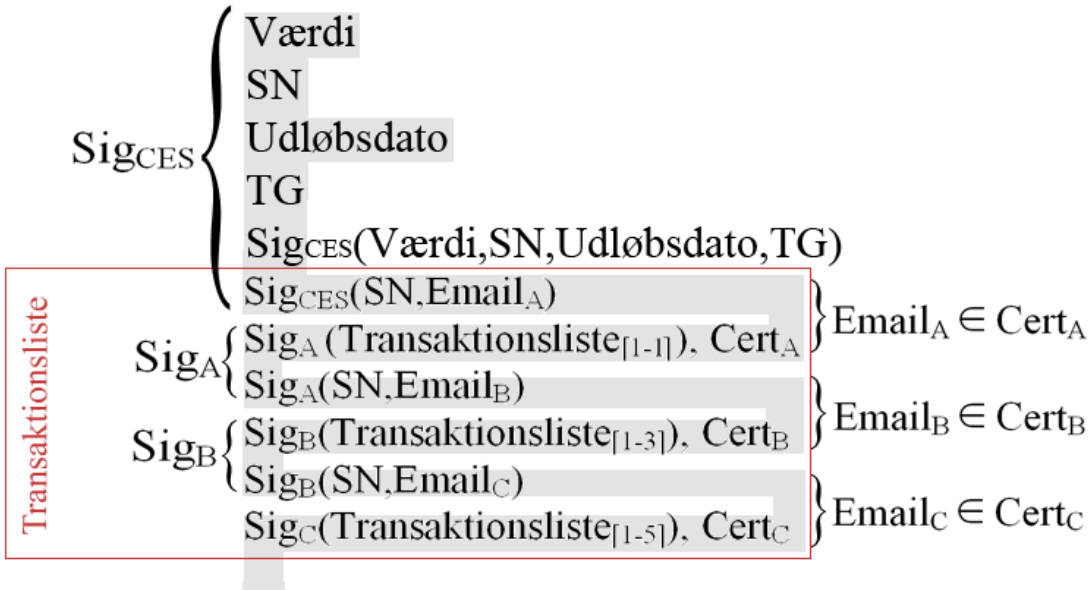
- Det første element er bundet sammen med resten af mønten med pengeserverens signatur af serienummeret og købers emailadresse.
- Elementer med *lige* indeks (type 2) er sammenkædet med forrige element, idet det kan kontrolleres, at certifikatet i elementet indeholder emailadressen (hashværdien af denne) fra forrige element.
- Elementer med *ulige* indeks (type 1) er sammenkædet med forrige element, idet overførselssignaturen kan verificeres vha. certifikatet fra forrige element.

I transaktionslisten findes altså en kæde af signaturer fra pengeserver til nuværende ejer af mønten. Hvordan dette, samt møntens design i øvrigt, sikrer mod økonomisk misbrug, vil blive forklaret i det følgende afsnit.

6.3.3 Økonomisk misbrug

6.3.3.a Modifikation

Som nævnt ovenfor er transaktionslisten på ethvert tidspunkt bundet sammen af en kæde af signaturer fra pengeserver til nuværende ejer. Grundet brug af signaturer, er det ikke muligt, at modificere de enkelte elementer uden at dette kan detekteres. Integritets-signaturerne og emailadresserne i certifikaterne sikrer, at elementer ikke kan byttes rundt eller kopieres og tilføjes til listen. Derudover er transaktionslisten bundet til resten af mønten vha. pengeserverens signatur. På Figur 11 er vist, hvordan møntens indhold er sammenkædet. På figuren symboliserer den grå markering kæden af signaturer.



Figur 11 – Møntens indhold og sikring af dennes integritet vha. signaturer. Den røde kasse afgrænser transaktionslisten og den grå markering viser kæden af signaturen.

Det er dog muligt, at modificere listen på én måde, således at dennes integritet er bevaret, nemlig ved at fjerne elementer fra slutningen af listen. Dette kan forsøges udnyttet af en angriber, som f.eks. opsnapper en mønt i transit mellem to brugere. Hvis angriberen tidligere har været ejer af mønten, kan denne fjerne de elementer, som er blevet tilføjet siden angriberen var indehaver af mønten. Dermed vil angriberen igen fremstå som ejer af mønten, og møntens integritet vil være bevaret. Dette svarer dog blot til, at angriberen tog en kopi af mønten, da denne oprindeligt var ejer af mønten. Desuden kan angriberen ikke forhindre, at afsender benytter mønten igen, når denne efter noget tid omdannes til en A-mønt. Dette skyldes at angriberen *ikke* kan sende en gyldig notifikations-mail, som får afsender til at slette sin tilhørende C-mønt (som beskrevet i afsnit 6.3.1). Hvis angriberen benytter mønten, vil denne altså fremstå som møntkopist med udelukkelse til følge (møntkopiering og tyveri af mønter behandles i næste afsnit). Hvis angriberen *ikke* tidligere har været ejer af mønten, kan denne *ikke* opnå noget ved at fjerne elementer.

Angriberen vil nemlig aldrig kunne fortsætte kæden i transaktionslisten, da denne ikke kender de private nøgler tilhørende de certifikater som findes i den pågældende transaktionsliste. Dermed kan mønten hverken indløses eller benyttes til at sende emails (dette uddybes i næste afsnit).

Selv om det er muligt at fjerne elementer fra slutningen af transaktionslisten, giver dette altså *ikke* mulighed for misbrug til skade for systemet eller dets brugere.

6.3.3.b Tyveri

Som forklaret i afsnit 6.3.2.a findes der i gyldige mønter en transaktionsliste indeholdende en ubrudt kæde af signaturer. Når der sendes emails, vil der altid være risiko for at påhæftede mønter kan stjæles undervejs, da systemet bygger på det eksisterende email-system. Mønterne er tilmed ikke krypterede og kan derfor frit læses af enhver, som på den ene eller anden måde opsnapper en email med påhæftet mønt. Rent sikkerhedsmæssigt er dette dog ikke et problem i SpamCash-systemet.

I det følgende beskrives det, hvordan systemet er sikret mod økonomisk misbrug, herunder tyveri. Bliver en mønt stjålet eller kopieret undervejs fra afsender til modtager, vil angriberen have følgende muligheder:

b. Mønten sælges videre eller angriberen forsøger selv at benytte denne.
c. Mønten forsøges indløst med det samme.
d. Mønten destrueres / gemmes i håb om at afsender lider økonomisk tab.
e. Mønten kopieres, og mønt eller kopier forsøges benyttet til gratis afsendelse af email.

Hvorfor en angriber ikke kan opnå noget i nogen af disse scenarier forklares i de følgende fire afsnit, der beskriver scenarierne i ovennævnte rækkefølge.

Salg eller brug af stjålne mønter

Fordi transaktionslisten indeholder en ubrudt kæde af signaturer, vil en angriber ikke kunne sælge mønten (da køber ikke kan bruge den). Køber vil nemlig *ikke* kunne fortsætte kæden af signaturer, da emailadressen i sidste element i transaktionslisten ikke befinder sig i købers certifikat. Af samme grund kan angriberen heller ikke *selv* bruge mønten. Kun hvis angriberen opsnapper en privat nøgle på ét af de certifikater, som befinder sig i transaktionslisten, vil denne kunne benytte mønten. Dette er dog straks en sværere opgave. Skal en sådan fremskaffes, kræver det nemlig, at angriberen bryder den lokale beskyttelse af den private nøgle på en af de pågældende klientmaskiner. Forudsat at brugere vælger passwords med omhu, vil dette være en uoverkommelig opgave for en angriber. Systemet er således generelt sikret mod salg og brug af stjålne mønter.

Indløsning af stjålne mønter

Når mønter indløses hos en pengeserver (CES), vil denne aflevere disse videre til en verifikationsserver (VS) sammen med certifikatet tilhørende brugeren (CES får kendskab til brugeres certifikater i forbindelse med oprettelse i systemet). VS vil herefter dekryptere transaktionslisten og udføre et integritetscheck på denne (på nøjagtig samme måde som klienter i systemet gør det ved modtagelse af email). Såfremt dette integritetscheck er succesfuldt, kontrolleres det, om indløseren er den retmæssige ejer af mønten, ved at sammenholde indløserens certifikat med certifikatet sidst i transaktionslisten. Stemmer disse overens, kan mønten indløses. I modsat fald får indløser ingen penge for mønten, da denne har forsøgt at indløse en mønt, der ikke tilhører denne.

Destruktion af stjålne mønter

Destrueres eller gemmes mønten på ubestemt tid af angriberen, vil dette blot resultere i, at afsender efter noget tid vil få refunderet sin mønt, som beskrevet i afsnit 6.3.1. Ved at destruere mønter kan en angriber altså ikke opnå noget.

Kopiering af stjålne mønter

Kopieres mønten efter tyveriet, vil de kopierede mønter umiddelbart kunne benyttes til at sende emails til den oprindelige modtager (og *kun* til denne). Angriberen vil ikke kunne sende til andre end den oprindelige modtager med den pågældende mønt, da denne ikke kan danne de nødvendige overførselssignaturer. Derfor har vi valgt at håndtere denne situation i klientproxyprogrammet ved at opretholde en database over serienumre på modtagne mønter. Det er dog ikke nok blot at afvise emails, hvor påhæftede mønter er set før, da dette sagtens kan forekomme legitimt (grundet transferability). Derfor holdes der desuden øje med *hvor mange gange* mønten er set før. Hvordan dette helt konkret gøres, forklares i afsnit 6.4.2.b omkring detektion.

I ovenstående er det blevet forklaret, hvordan systemet håndterer tyveri og modifikation af mønter. I det følgende afsnit vises det hvordan svindel i form af kopiering af *egne* mønter detekteres, og hvordan systemet identificerer og udelukker (blacklister) kopister.

6.4 Blacklisting

Det er nu blevet klargjort, hvorledes en transaktionsliste kan indgå i mønten, mens det samtidig sikres at mønten ikke kan misbruges. Kopiering kan ikke forhindres i et offline system med transferability som beskrevet i afsnit 5.4.1. Formålet med transaktionslisten er, at bestemme *hvem* der har kopieret en given mønt. Når pengeserveren eller en klient opdager to mønter med samme serienummer (en mønt har altså været kopieret), sendes mønterne til blacklistserveren. Herefter skal blacklistserveren kunne opklare 'forbrydelsen' vha. transaktionslisterne. Brugerne identificeres vha. certifikater. Opgaven går altså ud på, at finde ud af hvilket certifikat, og dermed hvilken bruger, der har kopieret mønten. Opdages mere end to eksemplarer af en mønt, indsendes disse blot parvis til blacklistserveren for at gøre dennes arbejde enklere. I nedenstående afsnit beskrives hvorledes, blacklistserveren kan afgøre hvilket certifikat, der skal tilføjes

blacklisten. Herefter følger et afsnit, der beskriver hvordan det sikres, at mønter der kopieres, bliver sendt til blacklistserveren i tide.

6.4.1 Opdatering af blacklist

Følgende procedure udføres af blacklistserveren, når denne modtager to mønter med samme serienummer.

I begge transaktionslister verificeres signaturkæden fra start til slut. De tre punkter til sidst i afsnit 6.3.2.a kontrolleres altså i hele listen. Kan dette ikke lade sig gøre er bevismaterialet ødelagt og ingen tilføjes blacklisten.

Verificeres transaktionslisterne i begge mønter succesfuldt, sammenlignes transaktionslisterne i de to mønter efterfølgende. Da mønterne har været kopieret, vil listernes begyndelse være identiske. I første element fra oven hvor listerne afviger vil kopistens certifikat være at finde. Kopisten fældes dermed ved først at opstille listerne som søjler i en matrix og dernæst gennemløbe disse fra oven. Det første element, der *ikke* er identisk i de to lister, vil altid være et overførselselement, da listernes integritet kunne verificeres[2]. Derfor gennemløbes listernes overførselselementer fra oven, efter følgende meget simple procedure, hvor det antages, at et NIL-element er tilføjet som sidste element i hver liste:

Hvis elementerne i en række er ens, og forskellige fra NIL, betragtes næste række. Ellers tilføjes certifikatet fra forrige række blacklisten og proceduren afsluttes.

I ovenstående angiver 'forrige række' integritetselementet lige inden de forskellige overførselselementer. Dette vil altid indeholde et certifikat, såfremt listens integritet er bevaret.

Ved denne procedure vil certifikatet tilhørende kopisten blive blacklistet. Kopieres f.eks. en mønt, hvor den ene indløses, og den anden benyttes til at sende en email, vil den ene transaktionsliste være identisk med en del af den anden. På Figur 12 ses netop denne situation, hvor person C har kopieret en mønt. Her vil proceduren korrekt kunne finde C, idet et *nil*-element optræder i rækken efter C. Den anden mulighed, som en kopist sandsynligvis oftere vil benytte, er at kopiere mønter og sende med alle kopierne. En sådan situation kan ses på Figur 13 (dog hvor blot to af mønterne er medtaget). Her har B kopieret en mønt, og derfor varierer listen netop efter B.

[2] Dette kan indses ved at huske på at en emailadresse i systemet findes i netop ét certifikat. Optræder den samme emailadresse i den *k*'te overførselssignatur i de to lister, vil det *k+1*'te element i begge lister *altid* indeholde identiske integritetssignaturer.

Række	Mønt 1	Mønt 2
1	A	A
2	B	B
3	C	C
4	NIL	D
5		E
6		NIL

Figur 12 – C kopierer en mønt og indløser den ene kopi, mens den anden benyttes til at sende med.

Række	Mønt 1	Mønt 2
1	A	A
2	B	B
3	C	D
4	E	F
5	NIL	G
6		NIL

Figur 13 – B kopierer en mønt og sender email med begge kopier.

Mere kompliceret bliver det, når kopier af kopierede mønter er i omløb. Det simpleste eksempel på dette kan ses på Figur 14. Her har B dannet én kopi af en mønt. Begge eksemplarer er senere blevet kopieret af hhv. F og T. Der er altså tre kopister involveret i denne situation.

Række	Mønt 1	Mønt 2	Mønt 3	Mønt 4
1	A	A	A	A
2	B	B	B	B
3	C	C	E	E
4	D	D	T	T
5	F	F	S	Q
6	H	G	NIL	NIL
7	NIL	I		
8		NIL		

Figur 14 – Dobbeltkopiering. B har dannet en kopi af en mønt, herefter har hhv. F og T kopieret disse kopier.

Alle fire mønter har samme serienummer og dette vil blive opdaget før eller senere. Indsendes mønt 1 og 2 vil F blive blacklistet, men ikke B. Indsendes herefter mønt 3 og 4 vil T blive blacklistet, men stadig ikke B. At mønterne indløses i en rækkefølge, der skaber en situation, hvor ikke alle kopisterne bliver blacklistet, er dog *meget* usandsynligt, især hvis antallet af kopier er stort. Den rækkefølge mønterne vil blive indsendt til blacklistserveren, er bestemt af rækkefølgen klienterne indløser mønter. Det er altså en fuldstændig tilfældig rækkefølge, der ikke kan kontrolleres af kopisterne. Dermed er det ligegyldigt, hvordan pengeserveren vælger at indsende kopier, når der forekommer mere end to. Modtager pengeserveren en mønt, hvor mange kopier tidligere har været indløst, kan denne blot vælge en mønt blandt de tidligere indsendte på en

entydig måde, f.eks. ved at indsende den sidst indløste og den næstsidst indløste. Det er altså ikke nødvendigt at udvælge disse tilfældigt for at fælde dobbeltkopister.

6.4.2 Detektion af kopier

Det er nu blevet beskrevet, hvordan blacklistserveren kan tilføje rette vedkommende til blacklisten, når denne får tilsendt to mønter. I det følgende vil blive beskrevet, hvorledes systemet sikrer, at blacklistserveren får tilsendt de kopierede mønter. Der er to måder hvorpå møntkopiering kan opdages: Af pengeserveren ved indløsning eller af en klient ved modtagelse af email.

6.4.2.a Detektion i pengeserver

Pengeserveren vil opdage kopierede mønter, når disse forsøges indløst. Serveren opretholder en liste med alle mønter, der har været indløst. Når to mønter med samme serienummer opdages, indsendes disse til blacklistserveren.

6.4.2.b Detektion i klient

Klienten kunne som pengeserveren blot huske alle serienumre og sende mønter til blacklistserveren, når to mønter med samme serienummer optræder. Da systemet understøtter transferability, kan der dog forekomme 'overførselsløkker', hvor emails hverken skal afvises eller sendes til blacklistserveren. Derudover skal emails, der sendes i flere kopier afvises, men ikke medføre blacklisting, da det både kan være afsender og en angriber på mailserveren, der sender disse. Derfor skal klienten i stedet beskyttes vha. den mere avancerede teknik, der beskrives herunder.

Klienten skal opretholde to lister: *EarnedCoins* og *InboxCoins*. Disse lister skal benyttes til at detektere møntkopiering og beskytte klienten mod emails med identiske mønter. *InboxCoins* er en liste med serienumre på mønter hørende til emails, som pt. befinder sig i indbakken. Modtages en email med en mønt, hvor serienummeret på denne findes i *InboxCoins*, skal denne afvises med det samme. Dette beskytter klienten mod angreb, hvor samme mønt benyttes af samme afsender til at 'bombardere' klienten med emails. Denne løsning er valgt, fordi det ikke er muligt at straffe afsender for dette, da det ikke kan afgøres, om det er afsender eller en man-in-the-middle angriber som foretager angrebet. Så snart mønter indløses eller føres tilbage til afsender, fjernes serienummeret fra denne liste, da der herefter er mulighed for at mønten igen kan bruges til at sende til klienten 'på lovlig vis' (grundet transferability). Såfremt mønten opkræves, placeres denne på listen *EarnedCoins*, som indeholder elementer på formen *(Coin, TimesEarned)*, og *TimesEarned* sættes til 1. Såfremt den pågældende mønt allerede eksisterer i *EarnedCoins*, tælles *TimesEarned* i stedet blot én op. *EarnedCoins* indeholder således samtlige mønter, som hidtil er blevet opkrævet og *TimesEarned* angiver, hvor mange gange den pågældende mønt har været opkrævet af klienten. Dette giver klienten mulighed for at kunne detektere og afvise ældre kopier af mønter, som denne tidligere har været ejer af. Når en mønt, som befinder sig i *EarnedCoins* modtages, kan klienten nemlig undersøge, hvor mange gange klientens certifikat findes i transaktionslisten. Hvis alt er i orden, skulle dette antal gerne være lig *TimesEarned*, da klienten tilføjes til en mønts transaktionsliste én gang, når denne opkræves. Hvis antallet af gange klienten optræder på transaktionslisten er mindre end *TimesEarned*, må der være tale om en ældre

kopi af mønten, eller en mønt hvor transaktionslisten har været modificeret (afkortet). Modtages en email vedhæftet en sådan mønt, skal denne derfor blot afvises. Bemærk at antallet af gange klienten forekommer på transaktionslisten, umuligt kan være større end *TimesEarned*, med mindre en softwarefejl er sket i klienten (så *TimesEarned* fejlagtigt er blevet talt op) eller en angriber har fået kendskab til klientens private nøgle. Klienten skal således afvise alle emails, hvor bare ét af nedenstående punkter er opfyldt:

- Møntens serienummer findes i *InboxCoins*
- Møntens serienummer findes i *EarnedCoins*, og *TimesEarned* er forskellig fra antallet af gange man optræder på transaktionslisten.

Klienten skal derudover afvise emails, hvor verifikationen af møntens integritet eller gyldighed fejler (verifikation beskrives i detaljer i afsnit 6.9.3).

Udover de mekanismer, der er beskrevet i de to foregående afsnit, vil stikprøvekontrol udført af både klienter og pengeservere yderligere være med til at gøre det formålsløst at sende spam med kopierede mønter. Dette foregår ved at klienter og pengeservere indsender tilfældigt udvalgte mønter til blacklistserveren. Hvordan denne stikprøve-kontrol udføres, beskrives i afsnit 8.2 (justering af systemet).

6.5 Anonymitet

Indledningsvis skal det understreges, at systemet ikke introducerer nye muligheder for at læse andres emails, da disse sendes nøjagtig som i det eksisterende email-system. Betalings-systemer, anvendt i emailsammenhæng, kan dog give mulighed for at opnå information om, hvem der sender til hvem i systemet. I det følgende redegøres for, hvordan det er sikret, at brug af SpamCash-systemet er forbundet med anonymitet i denne sammenhæng. Ved anonymitet forstås, at brugere er anonyme overfor hinanden samt overfor systemet (serverne).

6.5.1 Anonymitet overfor systemet

Første gang en bruger synliggør sig overfor systemet er under oprettelsen. Under denne proces vil såvel pengeserveren som registreringsserveren kende emailadresse (-rne) og det certifikat, der genereres til brugeren, som ønsker at oprette sig. Det skal her bemærkes, at der ikke stilles krav til hvilke emailadresser, der kan benyttes (anonyme adresser som f.eks. Hotmail kan altså godt benyttes). Registreringsserveren vil kende certifikatet, da det er her det udstedes. Emailadressen vil kendes, da en hashværdi af denne placeres i certifikatet (og denne i øvrigt benyttes i forbindelse med afsendelse af aktiveringsmail til brugeren). Registreringsserveren vil dog ikke kende identiteten på brugeren og vil i øvrigt kun komme i kontakt med denne i forbindelse med oprettelsen. Brugerens personlige informationer og emailtrafik er således skjult for registreringsserveren.

Certifikatet indeholder ingen personlige oplysninger, men altså udelukkende en hashværdi for hver af brugerens emailadresser. Pengeserveren vil komme til at kende brugerens certifikat i forbindelse med oprettelse af brugeren (dette er nødvendigt, for at

sikre at stjålne mønter ikke kan indløses). Pengeserveren vil i forvejen kende identiteten på brugeren, da vi antager at brugeren kontakter pengeserveren gennem et netbankinterface. Pengeserveren vil altså kende identiteten på de brugere i systemet, som er tilknyttet den pågældende pengeserver. Dette er ikke umiddelbart noget problem, så længe serveren ikke kan udlede noget om trafikken i systemet, altså hvilke brugere der sender til hvem, eller hvor meget de sender. Dette sikres ved at opdele information om brugeren som beskrevet i afsnit 5.4.2.b. Når en bruger hæver penge, foregår dette vha. blinde signaturer (som beskrevet i afsnit 4.4). Dette betyder, at pengeserveren *ikke* kender serienumre på mønter under udstedelse. Når mønterne på et tidspunkt indløses, vil pengeserveren derfor ikke vide hvem, der i sin tid købte mønterne. Information om de transaktioner, den enkelte mønt har været igennem, står i transaktionslisten. Denne er krypteret med en privat nøgle (blacklist- / verifikationsserverens) som ingen pengeserver kender. Derfor har pengeserveren ingen mulighed for at vide noget om, hvilke brugere mønten har været ejet af (på nær indløser). Den eneste information pengeserveren har, er altså hvor mange mønter hver enkelt bruger har hævet og indløst, hvilket ikke kan bruges til at udlede information om emailtrafikken i systemet.

Opdages møntkopiering (der indløses to mønter med samme serienummer) sender pengeserveren de pågældende mønter til blacklistserveren, som har den nødvendige private nøgle for at kunne dekryptere transaktionslisten. Blacklistserveren alene kan ikke udlede information om brugerne eller deres adfærd, da denne ingen identiteter kender, men kun ser mønter med transaktionslister indeholdende certifikater. Disse indeholder som nævnt kun hashede emailadresser. I princippet kan blacklistserveren dog udlede information ved at have en stor database over hashede emailadresser og forsøge at matche disse med hashværdierne i certifikaterne. Det er tvivlsomt om dette vil give noget brugbart og under alle omstændigheder vil blacklistserveren, kun modtage en forsvindende andel af mønterne i omløb (såfremt systemet virker som tilsigtet). Så længe blacklistserveren og pengeservere således holdes adskilt, vil brugere af systemet generelt være anonyme på den måde, at systemet ikke kan udlede hvor mange emails de sender, til hvem de sender, eller fra hvem de modtager.

Verifikationsserverne er, som blacklistserveren, ikke i besiddelse af brugernes personlige informationer. Derudover modtager verifikationsserverne kun et lille udsnit af indløste mønter. Derfor vil verifikationsserverne ikke kunne udlede information hverken om brugerne eller emailtrafikken i systemet.

6.5.2 Anonymitet overfor andre brugere

Klienter har altid mulighed for at se transaktionslisten på modtagne mønter, hvor der altid vil befinde sig ét eller flere certifikater. I praksis vil det være umuligt, at kortlægge hvilke ejere en mønt har haft, da det som sagt kun er hashværdier af emailadresser, der befinder sig i certifikaterne. En situation hvor man dog kan være ’heldig’ og udlede information kunne være som følger: Man kender f.eks. sin kærestes email-adresse og kender dermed også dennes emailhashværdi. Kender man også adressen på en person, han eller hun muligvis har en affære med, kan man undersøge alle mønter man modtager med henblik på at se om kæresten har sendt emails direkte til vedkommende. Ser man dog på antallet af mønter i omløb, vil sandsynligheden for, at en sådan ’undersøgelse’ er mulig, være

meget tæt på 0. I praksis er en bruger anonym overfor andre brugere i systemet, så snart antallet af brugere har nået en vis størrelse.

6.6 Use-cases

Nu er de centrale designaspekter blevet beskrevet. I resten af dette kapitel vil designet blive behandlet mere detaljeret.

I det følgende er beskrevet en række *use-cases* for systemet. Use-cases dækker over interaktionen mellem systemet og dets brugere. Titlen på hver use-case angiver, hvad brugeren gerne vil opnå og den enkelte use-case viser, hvorledes dette opnås.

For nedenstående use-cases gælder det generelt, at *første gang* systemet har brug for at tilgå den lokale e-mønt-beholdning (betegnes herefter *pengeskabet*), vil systemet vise en dialog-boks. Her skal brugeren angive placering af nøglefil, samt indtaste personligt password. Herefter vil programmet, indtil dette lukkes, kunne tilgå pengeskabet uden interaktion fra brugerens side. For at undgå unødig kompleksitet i nedenstående use-cases antages det derfor, at brugeren allerede har angivet placering af nøglefil, samt indtastet password (eller at denne gør det, når der bedes om det).

6.6.1 Use Case 1a: Start program

Scenario: Registrering har endnu ikke fundet sted, og brugeren indtaster korrekte login-oplysninger.

Brugerens handling	**Systemrespons**
Brugeren starter programmet.	Systemet viser en dialogboks, hvor brugeren kan indtaste brugernavn og password til pengeserveren, samt et personligt password. Derudover indtastes den eller de emailadresser, der ønskes registreret i systemet.
Brugeren indtaster korrekt brugernavn og password til pengeserveren og vælger et personligt password. Herefter trykker brugeren på en 'Register'-knap i dialogboksen så registreringsprocessen kan starte.	Systemet gennemfører registreringen og genererer alle nødvendige filer (herunder certifikatet og den private nøgle) og viser et vindue, der informerer brugeren om, at et antal e-mønter succesfuldt er blevet hævet fra pengeserveren.
Brugeren trykker på OK-knappen.	Systemet viser programmets hovedvindue, som indikerer at systemet nu er klar til brug.

Giver brugeren forkerte login-oplysninger eller opstår der fejl, når de forskellige servere kontaktes, skal der vises en fejlmeddelelse og brugeren gives mulighed for et nyt forsøg.

6.6.2 Use Case 1b: Start program

Scenario: Registrering *har* fundet sted.

Brugerens handling	**Systemrespons**
Brugeren starter programmet.	Systemet viser programmets hovedvindue, som indikerer at systemet er klar til brug.

6.6.3 Use Case 2: Konfigurer program

Scenario: Brugeren indtaster korrekt information i alle felter.
Udgangspunkt: Brugeren har startet programmet (Use Case 1)

Brugerens handling	**Systemrespons**
Brugeren vælger fanebladet 'Configuration' i hovedvinduet.	Der vises to felter i hovedvinduet, hvor der kan indtastes hhv. indgående og udgående mailserver-adresse. Der er endvidere en 'Update'-knap i fanebladet.
Brugeren indtaster indgående og udgående mailserver-adresse og trykker herefter på 'Update'-knappen	Systemet gemmer konfigurationen.

Indtaster brugeren ikke en IP, eller et DNS-navn, der umiddelbart kan oversættes til en IP-adresse, skal systemet give brugeren en fejlmeddelelse når der trykkes på 'Update'-knappen.

6.6.4 Use Case 3: Penge-status

Udgangspunkt: Brugeren har startet programmet (Use Case 1)

Brugerens handling	**Systemrespons**
Brugeren klikker på knappen 'Money-status' i hovedvinduet.	Systemet viser en dialogboks med information om den aktuelle e-mønt-beholdning samt statistik fra tidligere pengeoverførsler.
Brugeren trykker på knappen 'Ok' for at lukke dialogboksen	Systemet lukker dialogboksen og viser igen hovedvinduet.

6.6.5 Use Case 4a: Afsend email

Scenario: Brugeren *er* i besiddelse af e-mønter.
Udgangspunkt: Brugeren har startet programmet (Use Case 1) og i programmet er indtastet korrekte mailserver-oplysninger (Use Case 2)

Brugerens handling	**Systemrespons**
Brugeren afsender en email med sin foretrukne emailklient.	Et statusvindue vises i nederste højre hjørne, som angiver status for afsendelsen. Når afsendelsen er sket succesfuldt forsvinder dette igen.

Er der ikke indtastet korrekte mailserver-oplysninger giver *mailklienten* en fejlmeddelelse om at serverne ikke kan kontaktes.

6.6.6 Use Case 4b: Afsend email

Scenario: Brugeren *er ikke* i besiddelse af e-mønter:
Udgangspunkt: Brugeren har startet programmet (Use Case 1) og i programmet er indtastet korrekte mailserver-oplysninger (Use Case 2)

Brugerens handling	**Systemrespons**
Brugeren afsender en email med sin foretrukne emailklient.	Der vises en fejlmeddelelse som angiver at brugeren er løbet tør for e-mønter. Der opfordres til at denne hæver flere mønter eller indløser modtagen spam for igen at kunne sende emails med systemet.
Brugeren trykker 'Ok'-knappen i dialogboksen	Fejlbeskeden forsvinder.

6.6.7 Use Case 5: Modtag email

Udgangspunkt: Brugeren har startet programmet (Use Case 1), og i programmet er indtastet korrekte mailserver-oplysninger (Use Case 2)

Brugerens handling	**Systemrespons**
Brugeren modtager en eller flere nyankomne emails vha. sin foretrukne emailklient.	Forudsat at gyldige mønter er påhæftet de modtagne emails, videregives disse til emailklienten efter at mønterne er opsnappet og gemt. Alle andre emails filtreres og slettes af proxyprogrammet.

6.6.8 Use Case 6: Indkasser ”SpamCash”

Scenario: Brugeren har modtaget spam i sin indbakke (med påført betaling naturligvis).
Udgangspunkt: Brugeren har startet programmet (Use Case 1) og i programmet er indtastet korrekte mailserver-oplysninger (Use Case 2)

Brugerens handling	**Systemrespons**
Brugeren trykker på fanebladet ’Spam Withdraw’ i hovedvinduet.	I systemets hovedvindue vises en oversigt over modtagne emails.
Brugeren markerer de emails der betragtes som spam, og trykker på ’Get coin’-knappen.	Systemet indkasserer de pågældende mønter. Disse er hermed klar til brug for klienten og kan indløses eller benyttes til afsendelse. En notifikationsmail sendes til afsenderne af de pågældende emails, som fortæller at deres emails blev klassificeret som spam af modtager.

6.6.9 Use Case 7: Hæv Penge

Udgangspunkt: Brugeren har startet programmet (Use Case 1)

Brugerens handling	**Systemrespons**
Brugeren ønsker at hæve penge og trykker på fanebladet ’Bank Withdraw’ i hovedvinduet.	Systemet viser nu tre felter som skal udfyldes af brugeren samt en ’Withdraw’-knap. I det øverste felt skal angives det beløb, brugeren ønsker at veksle til e-mønter. I de to nederste felter skal brugeren indtaste brugernavn og password til pengeserveren.
Brugeren indtaster det beløb, der ønskes vekslet til e-mønter samt brugernavn og password. Derefter trykkes på knappen ’Withdraw’.	Pengene overføres og systemet viser et vindue med en OK-knap, som informerer brugeren om at transaktionen er foregået succesfuldt. Derudover vises hvor mange e-mønter der er blevet tilføjet pengeskabet.

Indtaster brugeren forkerte login-oplysninger eller angives et for højt beløb (mere end der er dækning for på kontoen) vises en fejlmeddelelse.

6.6.10 Use Case 8: Indløs penge

Udgangspunkt: Brugeren har startet programmet (Use Case 1)

Brugerens handling	**Systemrespons**
Brugeren ønsker at indløse e-penge og trykker på fanebladet ’Bank Deposit’ i hovedvinduet.	Systemet viser nu tre felter, som skal udfyldes af brugeren samt en ’Deposit’-knap. I det øverste felt skal angives det antal mønter, brugeren ønsker at indløse. I de to nederste felter skal brugeren indtaste brugernavn og password for at logge på en pengeserver.
Brugeren indtaster det beløb, der ønskes indløst samt brugernavn og password, og trykker på knappen ’Deposit’.	Mønterne indløses og systemet viser et vindue med en OK-knap, som informerer brugeren om, at transaktionen er foregået succesfuldt.

Indtaster brugeren forkerte login-oplysninger eller angives et for højt møntantal (højere end antallet af e-mønterne på harddisken), giver systemet en fejlmeddelelse.

6.6.11 Use Case 9: Tilføj email til whitelist

Udgangspunkt: Brugeren har startet programmet (Use Case 1)

Brugerens handling	**Systemrespons**
I hovedvinduet vælger brugeren fanebladet ’Whitelist’	Systemet viser en (muligvis tom) liste over brugere, som allerede findes på Whitelisten. Der vises endvidere et indtastningsfelt, hvor brugeren kan indtaste en emailadresse.
Brugeren indtaster emailadressen på vedkommende, som ønskes whitelistet og trykker på ’Add’-knappen	Forudsat at brugeren har indtastet en email-adresse med korrekt format, tilføjes denne til whitelisten. Den opdaterede liste vises.

6.6.12 Use Case 10: Fjern email fra whitelist

Udgangspunkt: Brugeren har startet programmet (Use Case 1)

Brugerens handling	**Systemrespons**
I hovedvinduet vælger brugeren fanebladet ’Whitelist’	Systemet viser en (muligvis tom) liste over brugere, som allerede findes på Whitelisten. Herunder vises en ’Remove’-knap.
Brugeren markerer en emailadresse i listen og trykker på ’Remove’-knappen	Emailadressen fjernes fra listen og den opdaterede liste vises.

6.6.13 Use Case 11: Luk program

Udgangspunkt: Brugeren har startet programmet (Use Case 1)

Brugerens handling	Systemrespons
Brugeren trykker på knappen 'Exit' i hovedvinduet.	Systemet gemmer al relevant data og lukker ned.

6.7 Sikkerhed

I dette afsnit beskrives først hvordan klienten opbevarer den private nøgle og de elektroniske penge lokalt. Derefter beskrives netværkssikkerheden i systemet herunder valg af nøglestørrelser.

6.7.1 Lokal sikkerhed

Hver enkelt klient har tilknyttet et certifikat med tilhørende privat nøgle samt et antal e-mønter, som denne benytter ved afsendelse af email. Certifikatet er offentligt og bliver sendt med hver gang klienten sender email og det behøver derfor ikke sikres. E-mønter er rent designmæssigt sikret mod tyveri (som beskrevet i afsnit 6.3.3.b). Dette er også grunden til, at mønterne kan påhæftes og sendes med email i ukrypteret form. Når vi har valgt at kryptere det lokale elektroniske pengeskab på harddisken, er det altså ikke pga. risikoen for tyveri, men af grunde, der vil blive redegjort for i det følgende. Hvis pengeskabet *ikke* var krypteret, kunne dette udnyttes af en angriber til at få ejeren af mønterne blacklistet. Dette kunne f.eks. ske ved at kopiere ejerens mønter én eller flere gange og lægge de kopierede mønter sammen med de originale mønter i ejerens pengeskab. Dermed kunne ejeren uforvarende komme til at benytte de kopierede mønter og derved risikere at blive beskyldt for møntkopiering med udelukkelse af systemet til følge. Ved at kryptere pengeskabet sikres integriteten af dette og dermed vil denne eller lignende situationer ikke kunne opstå.

Klientens private nøgle sikres på samme måde som det lokale pengeskab, altså vha. kryptering både med password og nøglefil (som det kendes fra f.eks. netbanker). Dermed kan nøglefilen gemmes på f.eks. en usb-nøgle, floppydisk eller et passende sted på den lokale harddisk. Kun hvis denne findes, *og* det korrekte password indtastes, kan den private nøgle tilgås. Det er nødvendigt med en streng sikkerhed omkring den private nøgle. Hvis denne nemlig opsnappes eller stjæles af en angriber, vil dette åbne mulighed for at sende med klientens mønter (hvis disse f.eks. opsnappes i transit). Hvis vedkommende, som har opsnappet klientens private nøgle, få rådighed over bare én af klientens mønter, kan denne kopieres og benyttes til at sende spam. Dette vil i sidste ende medføre lukning af klientens konto, som dermed vil være tvunget til at oprette sig på ny (og naturligvis skulle betale for dette). Hvis den private nøgle kompromitteres, vil det altså med al sandsynlighed betyde en lukning af kontoen, hvorfor det er vigtigt, at denne holdes privat.

6.7.2 Netværkssikkerhed

I det følgende beskrives det, hvordan der i systemet sørges for, at kommunikation mellem servere og klienter foregår sikkert.

I systemet er hver enkelt server blevet tildelt et certifikat med tilhørende privat nøgle (BLS og VS deler dog et certifikat-nøgle sæt). Distributionen af disse certifikater skal i en kommerciel version af systemet foregå ved at benytte en certifikatautoritet som f.eks. Verisign. Når systemet skal implementeres distribueres disse certifikater og private nøgler dog manuelt. Når en klient (eller server) ønsker at kommunikere med en server i systemet, opnås sikker kommunikation ved at benytte den pågældende servers certifikat til at oprette en kryptografisk session.

Først genererer klienten en symmetrisk sessionsnøgle, hvorefter denne krypteres med den pågældende servers offentlige nøgle. Sessionen oprettes ved at klienten sender den krypterede sessionsnøgle til serveren, hvorefter denne efterfølgende kan benyttes i en fastlagt tidsperiode til sikker kommunikation mellem server og klient. Når tidsperioden udløber, kræver serveren at en ny session (med en ny sessionsnøgle) oprettes, for at kommunikationen kan fortsætte. I systemet er det valgt, at en session er gyldig i 20 minutter. Dette skyldes, at klienter meget sjældent vil have brug for at kommunikere med en server i mere end 20 minutter og at sessionsnøglen kan betragtes som værende sikker i minimum 20 minutter.

6.7.3 Nøglestørrelser

Når størrelsen for de forskellige kryptografiske nøgler i systemet skal vælges, er det vigtigt, at der er sammenhæng i sikkerhedsniveauerne. Nøglerne kan betragtes som en kæde, hvor det svageste led dikterer sikkerhedsniveauet i systemet. I den forbindelse findes fem størrelser, der skal fastlægges i systemet.

- Antal bits i
    - Servernes asymmetriske nøgler
    - Sessionernes symmetriske nøgler
    - Klienternes asymmetriske nøgler
    - Klienternes symmetriske nøgler til lokal kryptering
    - Hashfunktionerne anvendt til digitale signaturer

Servernes asymmetriske nøgler vil blive udskiftet én gang om året. Dette giver en angriber et år til at bryde disse. Sessionsnøglerne er kun gyldige (og dermed brugbare) i 20 min. Størrelsen på disse nøgler kan derfor vælges relativt lille uden at gå på kompromis med sikkerheden. Idet vi anvender de anerkendte algoritmer RSA (asymmetrisk) og AES (symmetrisk), vil nøglestørrelserne afhænge af disse algoritmers egenskaber.

Det er altid vigtigt at vurdere, i hvor høj grad det krypterede data skal holdes hemmeligt. Er der f.eks. tale om fru Jensens indkøbsseddel, vil denne skulle beskyttes i mindre grad end information om militærets nyeste teknologi. Et af kravene til systemet er, at det ikke

skal være muligt at misbruge dette økonomisk. Dette betyder, at de kryptografiske nøgler skal være sikre mindst et årti ud i fremtiden og derfor vælges en asymmetrisk nøglestørrelse på 2048 bit til serverne. Som beskrevet i *Key management guideline* udarbejdet af NIST (National Institute of Standards and Technology) i USA [41], anses dette som værende sikkert mindst 10 år ud i fremtiden. Ud fra dette skal de andre størrelser i systemet vælges, så det samlede sikkerhedsnivea bevares. Nøglestørrelsen på de symmetriske nøgler vælges til 128 bit. Anvendes AES-algoritmen, vil der dermed opnås mindst samme sikkerhedsniveau som ved anvendelse af 2048 bit i RSA algoritmen ifølge NIST [41]. Det vil teoretisk set vare mindst et århundrede, før en 128 bit AES nøgle kan brydes indenfor rimelig tid, forudsat at der ikke kommer nye revolutionerende kryptoanalytiske teknikker [42, 43 (s. 167)]. Derudover anvendes en SHA-256 hashfunktion, da sikkerhedsniveauet for denne svarer til sikkerhedsniveauet for en 128 bit AES nøgle [41].

Den sidste nøglestørrelse, der skal fastlægges, er størrelsen på klienternes asymmetriske nøgler. Størrelsen af disse nøgler skal sammenlignes med sikkerheden hos klienterne. Den private nøgle opbevares med dobbeltkryptering, altså krypteret med både password og nøglefil, hos klienten. Nøglefilen indeholder en AES-nøgle, der ligeledes vælges til 128 bit. Lærer en angriber en klients nøgle at kende, vil denne kunne misbruge dennes certifikat, indtil dette blacklistes. Det er op til klienten, hvor godt denne beskytter sit password, sin computer og nøglefilen. Værktøjet til et relativt højt sikkerhedsniveau er til stede, men det er dog ofte tilfældet, at brugere ikke anvender dette fornuftigt. Skulle det ske, at brugerens system kompromitteres, vil dette dog kun gå ud over brugeren selv. Det sikkerhedsniveau, der opretholdes af brugeren, skal svare til sikkerheden i nøglen i dennes certifikat. Ud fra denne betragtning, og fordi en bruger skal kunne anvende systemet med samme nøgle i mindst et par år, vælges en asymmetrisk nøglestørrelse på 1024 bit. Levetiden på et klient-certifikat skal altså være noget længere end en servers certifikat, men der er alligevel valgt en mindre nøglestørrelse. Dette skyldes, at angribere med al sandsynlighed vil bruge langt mere energi på at afsløre servernes private nøgler end en tilfældig klients. Systemets sikkerhed vil nemlig lide et langt større knæk (og mulighed for misbrug vil være væsentlig større), hvis en servers private nøgle afsløres end hvis det samme sker for en klient. Derudover vil den økonomiske gevinst ved at afsløre en klients nøgle være meget begrænset. Hermed er nøglestørrelserne i systemet fastlagt. I Tabel 3 herunder gives et overblik over disse:

Nøgle	Algoritme	Antal bit
Servernes asymmetriske nøgler	RSA	2048
Klienternes asymmetriske nøgler (digital signatur)	RSA	1024
Sessionerne symmetriske nøgler	AES	128
Klienternes symmetriske nøgler til lokal kryptering	AES	128
Hashfunktionerne anvendt til digitale signaturer	SHA-256	256

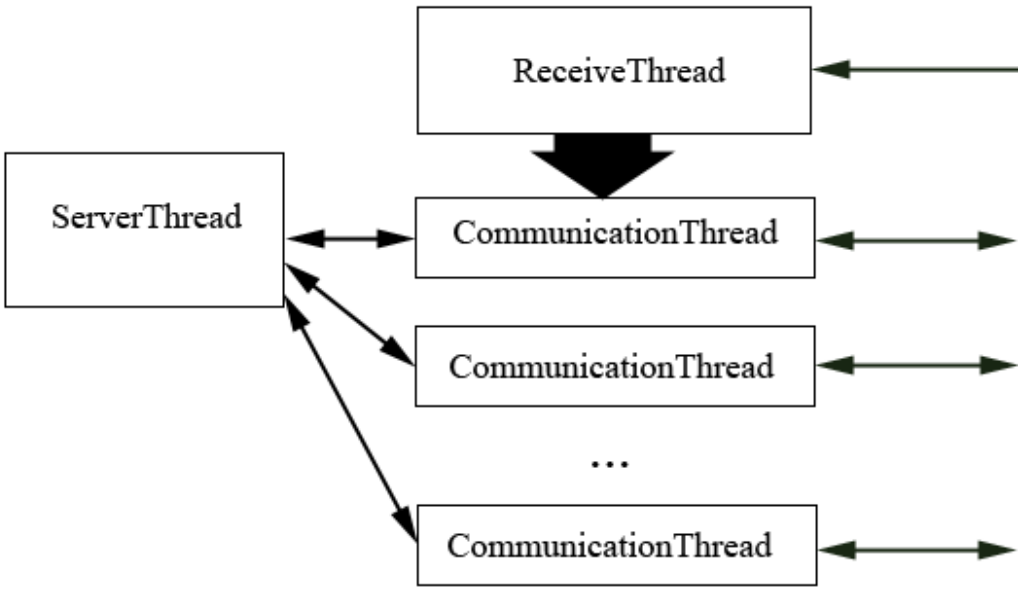
Tabel 3 – De anvendte algoritmer og nøglestørrelser.

6.8 Kommunikationsprotokoller og serverdesign

I det følgende vil kommunikationsprotokollerne i systemet blive gennemgået med udgangspunkt i serverne. Når servernes funktionalitet er fastlagt, og protokollerne er blevet beskrevet vha. sekvensdiagrammer, vil den mere komplekse klientapplikation blive beskrevet.

6.8.1 Servere

Serverne i systemet har en række fællestræk. De skal kunne behandle et stort antal forbindelser samtidig og den service, der ydes, skal altid være tilgængelig. Derfor er serverne opbygget som vist på diagrammet på Figur 15.



Figur 15 – Intern struktur af serverapplikationer. Kasserne symboliserer trådene i applikationen. De tynde pile viser kommunikation. Den tykke pil illustrerer at 'ReceiveThread' danner en 'CommunicationThread' og videregiver forbindelsen til klienten.

Der findes tre typer af tråde i serverne. Modtagetråden (ReceiveThread) lytter på serverens tildelte port og modtager forbindelser. Når en klient kontakter serveren, vil modtagetråden sørge for oprettelse af en kryptografisk session. Kan dette gøres successfuldt, danner modtagetråden en kommunikationstråd (CommunicationThread). Kommunikationstråden vil herefter kommunikere med klienten og selve servertråden (ServerThread). Afbrydes forbindelsen eller afslutter klienten sine forespørgsler, vil kommunikationstråden selvdestruere. I en serverapplikation findes altså altid én servertråd, én modtagetråd og et variabelt antal kommunikationstråde afhængig af antallet af forbundne klienter. Under opstart af en server vil denne starte en modtagetråd. Modtagetråden vil så, efterhånden som forbindelser oprettes, danne kommunikationstråde.

Fordelen ved at strukturere serverne på denne måde er, at flere klienter (i princippet uendeligt mange) kan håndteres samtidig. Derudover opnås det, at modtagetråden *altid* er klar til at modtage en forbindelse fra en ny klient. Servicen, der leveres, er altså altid tilgængelig. Det er dog nødvendigt, at indføre et loft på antallet af tråde for at reducere risikoen for et succesfuldt DOS-angreb.

Serverne er altså meget klientvenlige, idet de altid er tilgængelige. Til gengæld skal det understreges, at hvis en server oplever nogen form for problemer, såsom ulæseligt data eller protokolfejl, vil forbindelsen til klienten blive afbrudt øjeblikkeligt. Herefter kan klienten starte forfra, hvis det ønskes.

I de følgende afsnit beskrives de enkelte serveres funktionalitet, altså virkemåden af servertråden i de forskellige servere. Der gives derudover et overblik over protokollerne ved hjælp af sekvensdiagrammer. På diagrammerne er vist applikationerne i systemet og hvilke netværkspakker, der sendes mellem disse.

6.8.2 Blacklistserver

Brugere autentificeres ikke, når disse kontakter blacklistserveren. Dette skyldes, at blacklisten er offentlig tilgængelig og at serverens arbejde på denne måde begrænses. Skulle serveren autentificere brugerne, ville det kræve mindst lige så meget trafik og arbejde, at autentificere (og evt. afvise) brugere, som hvis disse blot får tilsendt blacklisten uden autentifikation. Grundet blacklistserverens 'knudepunktsrolle' i systemet, er dennes arbejde forsøgt reduceret mest muligt. Derfor indeholder serveren udelukkende følgende variable og funktionalitet.

Blacklistserveren indeholder følgende variable:

- **BLS_certificate** – Blacklistserverens certifikat
- **PrivateKey** – Privat nøgle hørende til **BLS_certificate**
- **CES_certificate** – Pengeserverens certifikat
- **RS_certificate** – Registreringsserverens certifikat (rodcertifikatet)
- **Blacklist** – En liste med hashværdier af klientcertifikater

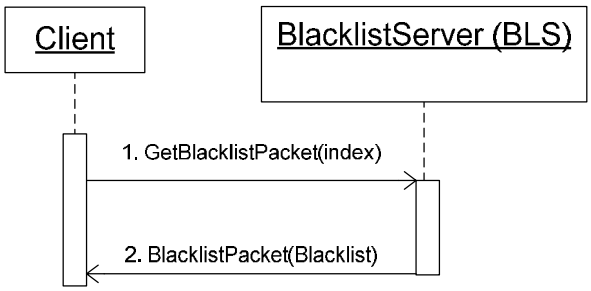
Herunder er beskrevet serverens input og output. Ved hvert punkt er vist hvilken netværkspakke, der modtages og derefter de resulterende handlinger og tilbagesendte pakker. Denne opstilling vil også blive benyttet i de følgende afsnit.

1. *ReportCoinsPacket(Coin a, Coin b)*. Medfører intet svar, men blacklistserveren sørger for korrekt opdatering af blacklisten. Dette gøres ved at benytte proceduren beskrevet i afsnit 6.4.

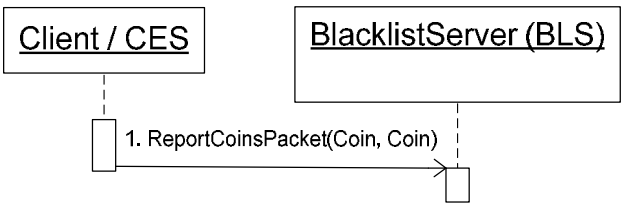
2. *GetBlacklistPacket(Index k)*. Blacklistserveren returnerer en *BlackListPacket(BlackList)* med de certifikat-hash-værdier, der er blevet tilføjet blacklisten efter indeks *k*.

Ved at give klienterne mulighed for kun at hente den del af blacklisten, de ikke allerede har, spares båndbredde. Listen vil i princippet vokse i det uendelige, men ved at klienterne blot henter den del af listen de mangler, vil det kun sjældent være hele listen, der skal sendes. En anden måde at begrænse listens størrelse på kunne være, at lade meget gamle elementer udgå fra listen. Dette ville give mulighed for at misbruge de pågældende certifikater igen. Men kan det sandsynliggøres, at certifikatet ikke findes mere, vil det være acceptabelt at gøre dette.

Herunder ses sekvensdiagrammer for protokollerne, der anvendes, når der kommunikeres med blacklistserveren (se Figur 16 og Figur 17). Det ses at kommunikationen er meget simpel og det kan bemærkes, at der ikke kommer noget svar fra serveren, når mønter indrapporteres på Figur 17. Dette skyldes, at blacklistserverens arbejde på denne måde reduceres. De to diagrammer kan virke meget simple (og dermed overflødige), men er medtaget for fuldstændighedens skyld.



Figur 16 – En klient downloader blacklisten



Figur 17 – En klient eller CES indrapporterer kopierede mønter.

For at reducere arbejdet i blacklistserveren yderligere, kunne det være en mulighed at distribuere dennes arbejdsbyrde. Én server kunne håndtere indrapportering, og et hierarki af andre blacklistservere kunne håndtere upload af listen til klienterne. På denne måde ville båndbreddeforbruget kunne fordeles på en række servere. Derudover ville serveren, der står for opdatering, også kunne benytte en række arbejdsservere, der kunne finde det korrekte certifikat ud fra to mønter (altså proceduren i afsnit 6.4). Hermed ville den centrale blacklistserver blot skulle opretholde en liste, mens de to krævende opgaver (upload og opdatering) ville blive fordelt på andre servere.

6.8.3 Verifikationsserver

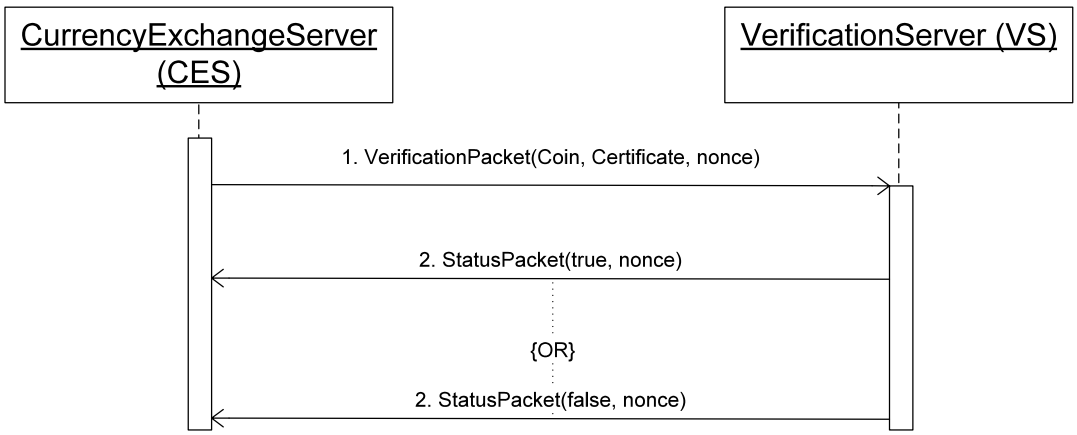
Blacklistserveren kunne i princippet *også* håndtere verifikationsserverens opgaver, da serverne deler samme nøglepar. Da blacklistserverens arbejde dog som nævnt skal holdes på et minimum, er arbejdsbyrden fordelt på de to. Verifikationsserveren indeholder ingen variable udover certifikater og nøgler, som det fremgår af listen herunder:

- **BLS_certificate** – Blacklistserverens certifikat
- **PrivateKey** – Privat nøgle hørende til **BLS_certificate**
- **CES_certificate** – Pengeserverens certifikat
- **RS_certificate** – Registreringsserverens certifikat (rodcertifikatet)

Den eneste opgave verifikationsserveren skal udføre, er verifikation af en mønts gyldighed og ejerskab. Herunder er beskrevet serverens input og output.

1. *VerificationPacket(Coin a, Certificate c, nonce).* Verifikationsserveren verificerer møntens integritet, og kontrollerer at sidste element i transaktionslisten indeholder certifikatet *c*. I bekræftende fald sendes en *StatusPacket(true, nonce),* og ellers en *StatusPacket(false, nonce).*

Transaktionslisten i mønten kan dekrypteres med blacklistserverens nøgle. Kontrollen af listen og mønten i øvrigt foregår på præcis samme måde som i klientapplikationen, når en bruger modtager en email. Derfor henvises her blot til afsnit 6.9.3, hvor denne procedure er beskrevet. Herunder ses sekvensdiagrammet for situationen, hvor en pengeserver vil have verificeret en mønt.



Figur 18 – Pengeserveren sender en mønt og et certifikat til verifikationsserveren der verificerer ejerskabet af mønten.

Selvom det udelukkende er pengeservere (CES), der skal kunne benytte verifikationsserveren, foregår ingen autentifikation. Dette skyldes, at serverens funktionalitet ikke umiddelbart giver anledning til misbrug. Serverens funktionalitet adskiller sig nemlig næsten ikke fra klientens. Til gengæld er det vigtigt, at denne er sikret mod man-in-the-middle angreb, hvor en angriber udgiver sig for at være verifikationsserveren og f.eks. svarer *true* på alle forespørgsler. Dette sikres der imod på

to måder. Pengeserveren opretter forbindelse til verifikationsserveren ved at kryptere en sessionsnøgle med dennes offentlige nøgle og dermed kan pengeserveren være sikker på, at det *er* en verifikationsserver, der kommunikeres med. Derudover sendes et nonce (altså et tilfældigt tal) med i hver pakke fra pengeserver til verifikationsserver, og verifikationsserveren returnerer et par bestående af true/false og det samme nonce. Dermed sikres imod replay-angreb, idet pakker ikke kan gentages eller ombyttes af en angriber.

Et DOS angreb mod verifikationsserveren vil være nærliggende for en angriber, idet mønters ejerskab dermed ikke vil kunne kontrolleres. Derfor er det vigtigt, at pengeserveren ikke giver mulighed for at indløse mønter, så længe en verifikationsserver ikke kan kontaktes.

6.8.4 Pengeserver

Kommunikationstrådene i pengeserveren sikrer at brugeren er logget ind, før der videresendes forespørgsler til servertråden. Når login-information modtages, vil denne blive kontrolleret af servertråden. Når et login er gennemført successfuldt, husker kommunikationstråden brugernavnet for brugeren, og videregiver dette til servertråden ved de efterfølgende forespørgsler.

Pengeserveren indeholder følgende variable:

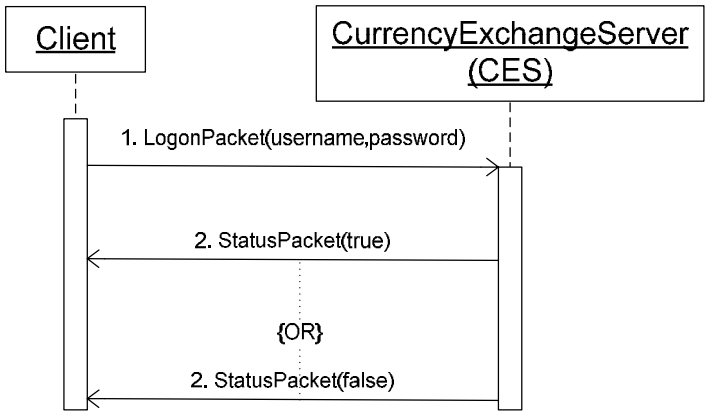
- **CES_certificate** – Pengeserverens certifikat
- **PrivateKey** – Privat nøgle hørende til **CES_certificate**
- **BLS_certificate** – Blacklistserverens certifikat
- **RS_certificate** – Registreringsserverens certifikat (rodcertifikatet)
- **UsedCoins** – Liste med alle mønter der har været indløst
- **UsedSerialNumbers** – Liste med serienumre på alle mønter der har været indløst
- **Accounts** – Liste med konti for bankens brugere
- **CoinSets** – Liste med møntsæt.

Variablen 'UsedSerialNumbers' indeholder i princippet overflødig information (da denne er indeholdt i UsedCoins). For hurtigt at kunne undersøge om et givent serienummer har været anvendt før, er serienumrene dog gemt for sig i denne variabel. Derudover indeholder pengeserveren funktionalitet til at opretholde simple konti for brugerne.
I det følgende beskrives serverens funktioner.

6.8.4.a Login

Det første der foregår, under en hvilken som helst interaktion med pengeserveren, er login. Herunder er beskrevet, hvorledes dette foregår og på Figur 19 ses det tilhørende sekvensdiagram.

1. *logonPacket(username, password)*: CES kontrollerer login informationen, og sender en *logonResponsePacket(ok = true/false)* tilbage. Er *ok=true* sættes *loggedIn* til *true* i den pågældende kommunikationstråd.



Figur 19 – Login-procedure for pengeserveren.

I det følgende beskrives pengeserverens øvrige funktioner. For at gøre diagrammerne mere overskuelige er login-pakker her ikke medtaget. I alle tilfælde foregår der dog login som vist i Figur 19, inden nogen anden kommunikation med pengeserveren kan finde sted.

6.8.4.b Hæv penge

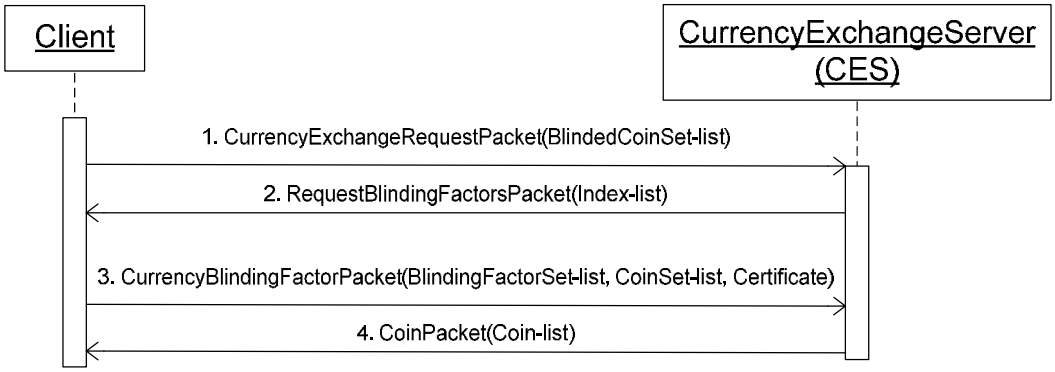
Når en klient ønsker at hæve penge, foregår dette ved at benytte en 'blind signature'-protokol. Protokollen bygger på blinde signaturer i RSA-algoritmen, som beskrevet i afsnit 4.4. Pengeserveren kan naturligvis ikke signere penge blindt uden at kende værdien af de mønter, der signeres. Derfor kan de blinde signaturer f.eks. indgå i følgende protokol for at være brugbare [43]:

a. For hver e-mønt klienten ønsker udstedt, genereres et *sæt* e-mønter, der alle blindes med hver sin blinding-faktor (Alle mønter har forskellige serienumre) Disse sendes til CES.
b. CES udvælger tilfældigt én mønt i hvert sæt. Herefter sendes en besked til klienten om, at denne skal fremsende blinding-faktorer på alle på nær de udvalgte mønter.
c. Klienten sender blinding-faktorer samt sit certifikat til CES.
d. CES kontrollerer gyldigheden af certifikatet samt at alle mønter fra hvert sæt, på nær den udvalgte, er struktureret korrekt (og har samme værdi). Den ene udvalgte mønt fra hvert sæt signeres herefter blindt og de signerede mønters værdi hæves på brugerens bankkonto. De signerede mønter sendes til sidst til klienten.

I ovenstående protokol er det klienterne, der genererer mønterne. Disse er værdiløse uden pengeserverens signatur. Fordelen ved ovenstående protokol, er, at CES ikke kender serienummeret på de mønter, der udstedes. Hvis det kræves, at der findes 100 mønter i et sæt, vil sandsynligheden, for at en bruger snyder pengeserveren, være en hundrededel[3]. Da hver e-mønts værdi er relativt lille (under 1 krone), vil sandsynligheden for at slippe af sted med et større pengebeløb (f.eks. 100 kr.) være meget tæt på 0 (0.01^{100}). Derfor

[3] Hvor mange mønter der skal være i et sæt, mønternes værdi og andre parametre fastlægges i afsnit 8.2

synes det rimeligt at antage, at klienterne ikke vil forsøge at snyde pengeserveren. Opdages svindel, kan klienterne straffes af banken. Samtidig kan klienterne være sikre på, at pengeserveren ikke kender serienumrene på de mønter, der er blevet udstedt, da det udelukkende er klienten, der kender blinding-faktorerne hørende til de udstedte mønter. For yderligere detaljer omkring denne protokol henvises til Bruce Schneiers *Applied Cryptography* [43]. Nedenfor ses et sekvensdiagram, der viser interaktion mellem pengeserver og klient ved udstedelse af penge.



Figur 20 – Udstedelse af mønter.

Herunder ses en beskrivelse af, hvordan pengeserveren håndterer de forskellige netværkspakker i protokollen. Nummereringen fortsættes fra forrige liste med input (Login).

2. *CurrencyExchangeRequestPacket(BlindedCoinSet-list)*: CES gemmer *BlindedCoinSet-list*. CES udvælger et tilfældigt indeks i hvert *BlindedCoinSet* i listen. Herefter sendes disse indeks tilbage til klienten i en *requestBlindingFactorsPacket(Index-list).*
3. *CurrencyBlindingFactorPacket(blindingFactorSet-list, CoinSet-list, Certificate)*: CES kontrollerer gyldigheden af certifikatet, samt at alle mønter fra hvert sæt på nær den udvalgte har samme værdi, og er struktureret korrekt (at mønterne sendes her, skyldes implementeringstekniske forhold). Er dette tilfældet, signeres den ene ukontrollerede mønt fra hvert sæt blindt. Beløbet hæves på brugerens bankkonto (har brugeren e-mønter til gode grundet oprettelse, hæves disses værdi ikke) og de signerede mønter sendes til klienten i en *CoinPacket(Coin-list)*.

Pengeserveren gemmer de enkelte *BlindedCoinSets* under interaktionen med klienten og disse knyttes til klientens brugernavn. Hermed er det kun den korrekte klient, der kan få tilsendt mønterne senere. En angriber kan altså ikke ’bryde ind’ i protokollen. Selvom en angriber opfanger de krypterede mønter i trin 4, vil denne ikke kunne bruge disse til noget, ud over at forhindre at klienten modtager dem. I dette tilfælde vil pengeserveren dog kunne sende klienten endnu en kopi af mønterne på et senere tidspunkt. Her vil pengeserveren altså udlevere kopier af mønter. Hvis klienten misbruger disse mønter, vil

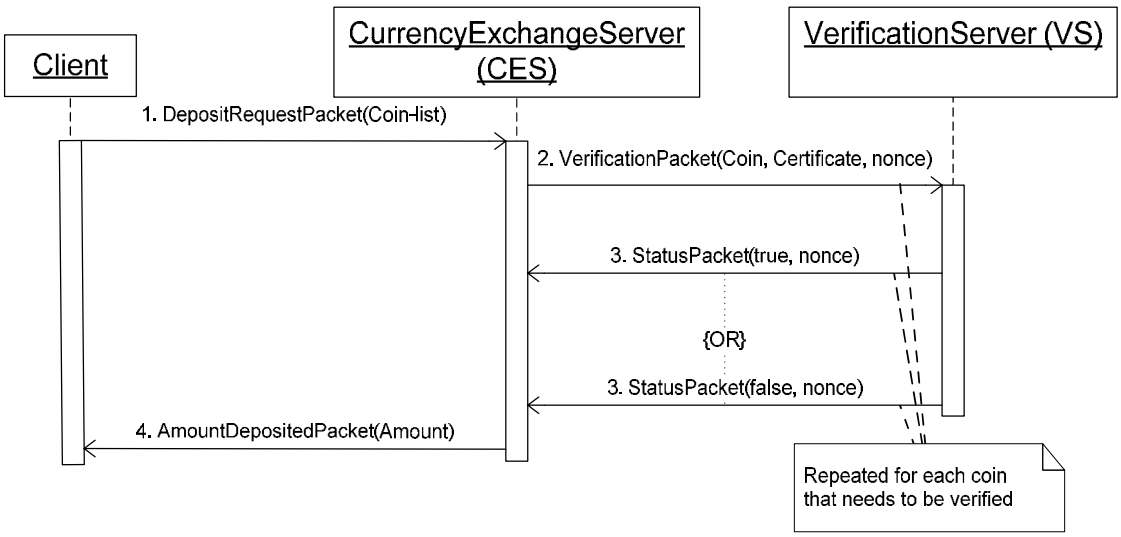
dette dog håndteres præcist, som hvis denne selv havde kopieret dem. Derfor er der intet sikkerhedsproblem i dette.

En anden situation, hvor der også hæves penge på brugerens konto, er under oprettelsen. Oprettelsesproceduren beskrives i detaljer i afsnit 6.8.5, der omhandler registrering. Derfor vil pengeserverens rolle i registreringen blot blive vist uden videre forklaring her, mens der henvises til afsnit 6.8.5. Under registrering udfører pengeserveren følgende:

4. *RegistrationInfoPacket(activationNumber)*: CES hæver oprettelsesbeløbet på klientens konto og det markeres, at denne har e-mønter for dette beløb til gode. Herefter sendes et *activationNumber* inkl. signatur af dette til RS i en *ActivationPacket(activationNumber, Sig_{CES}(activationNumber))*. Pengeserveren venter nu et øjeblik på, at certifikatet er udstedt og henter dette fra registreringsserveren. Herefter sendes en *StatusPacket* til klienten, der fortæller denne, enten at certifikatet er klar til afhentning hos registreringsserveren, eller at en fejl er opstået.

6.8.4.c Indløsning af e-mønter

Når en bruger ønsker at indsætte penge, udføres den på Figur 21 viste kommunikation



Figur 21 – Indløsning af mønter.

Mere konkret udfører pengeserveren følgende opgaver:

5. *DepositRequestPacket(Coin-list)*: CES kontrollerer ejerskabet af mønter ved at sende disse til verifikationsserveren. Derudover kontrolleres det, at der ikke er tale om kopier af tidligere indløste mønter. Herefter krediteres brugerens konto med beløbet for de mønter, som klarer ovennævnte kontrol. CES gemmer serienumrene på indløste mønter i listen *UsedSerialNumbers* og selve mønterne i *UsedCoins*. Der indrapporteres til BLS, hvis der opdages kopier som beskrevet i afsnit 6.8.2. Herefter sendes en *AmountDepositedPacket(Amount)*, der fortæller klienten værdien af det indløste (nogle mønter har muligvis været kopieret eller modificeret og er derfor uden værdi).

En angriber kan opfange pakker i trin 1 for f.eks. at forsøge at destruere klientens penge. Derfor beholder klienten en kopi af mønterne indtil trin 4 er gennemført. Standses pakken i trin 1, vil klienten kunne forsøge at indløse mønterne igen. I denne situation (og ved replay-angreb) kan det ske, at serveren modtager den samme mønt flere gange fra en klient. Serveren indrapporterer derfor ikke mønter, som er identiske til BLS. Kopister, der anvender kopierede mønter fældes dog stadig, da transaktionslisterne vil være forskellige i denne situation.

I trin 2 og 3 medsendes et nonce for at sikre mod man-in-the-middle-angreb (replay-angreb og angreb ved ombytning af pakker).

Pengeserveren behøver ikke at sende alle mønter til VS. Denne kan nøjes med at verificere et udvalg af de indløste mønter. Er straffen, for at forsøge at indløse stjålne mønter, høj nok, vil verifikation af et lille udvalg af mønter være nok til, at kun få vil forsøge at svindle. Samme princip benyttes under udstedelse af mønter, hvor det teoretisk også er muligt at snyde pengeserveren. De færreste vil dog forsøge dette, hvis chancen for at slippe afsted med svindlen er tilstrækkelig lille og straffen høj nok.

Verifikationsserverens arbejdsbyrde begrænses på denne måde. Verifikationsserveren kan optimeres ved at tillade at denne kan modtage lister med mønter i stedet for blot enkelte mønter. Hermed vil antallet af forbindelser og netværkspakker reduceres. For at holde serveren så simpel som muligt, har vi dog valgt at medtage denne optimering.

6.8.5 Registrationserver

Brugere der kontakter registreringsserveren autentificeres ikke. Kommunikationen med en given klient foregår dog naturligvis krypteret for at sikre mod angreb på protokollen. Registreringsserveren indeholder følgende:

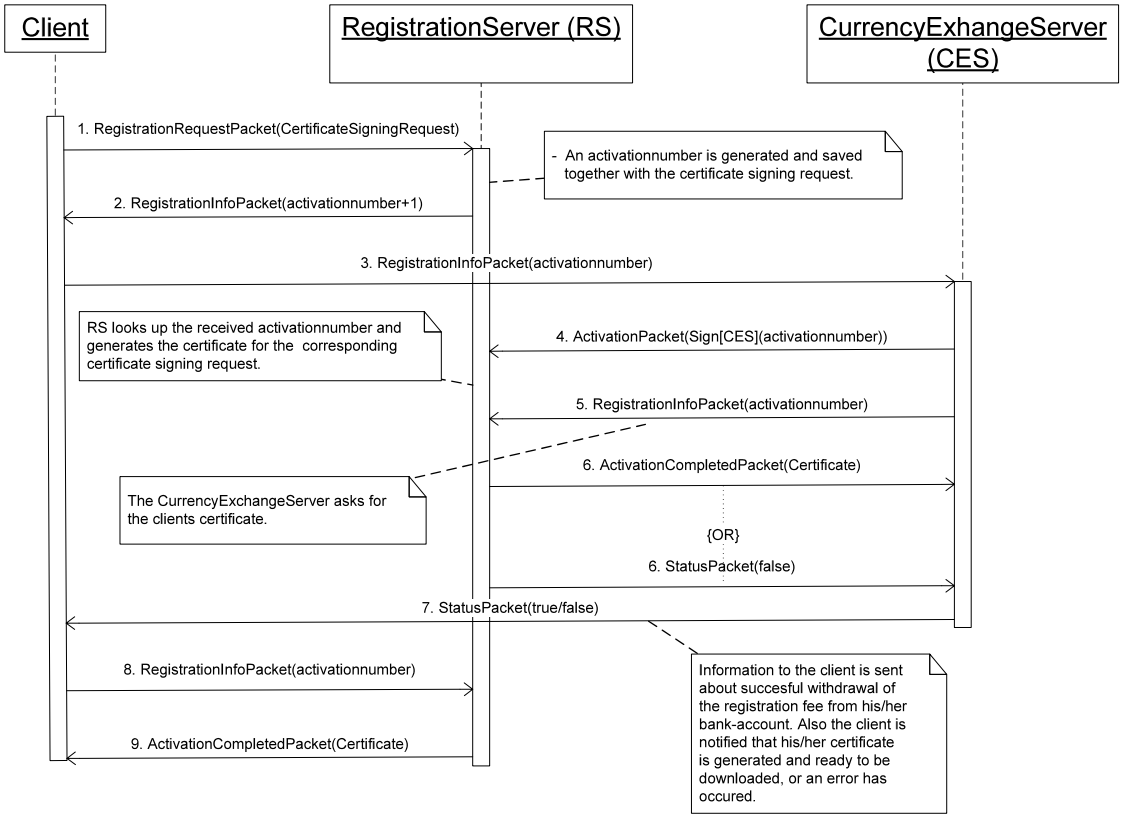
- **RS_certificate** – Registreringsserverens certifikat (rodcertifikatet)
- **PrivateKey** – Privat nøgle hørende til **RS_certificate**
- **CES_certificate** – Pengeserverens certifikat
- **PendingAccounts** – Liste med PendingAccount-elementer:
    - **PendingAccount –** Indeholder:
        - ActivationNumber – Sendes fra CES til RS når klienten har betalt
        - CertificateSigningRequest – Genereres af klienten ved oprettelse

Variablen *PendingAccounts* indeholder et element for hver klient, der er blevet oprettet for nylig (og muligvis endnu ikke har fået tilsendt sit certifikat). Variablen *ActivationNumber* benyttes til at skelne mellem oprettelser. Da klienter ikke autentificeres, vil angribere kunne forsøge at gætte et *activationNumber* med henblik på at få tilsendt en andens certifikat. Der er dog ikke noget sikkerhedsproblem i dette, da certifikaterne er offentlige. Certifikatet må ikke slettes fra serveren, så snart klienten har hentet dette, da serveren ikke kan vide, om det vitterlig *er* klienten og ikke en angriber, der har hentet dette. Hvis certifikatet blev slettet, ville en angriber kunne hente den eneste kopi af dette. Hermed vil klienten, der forsøger at oprette sig, umiddelbart miste sit oprettelsesgebyr. Vi vælger derfor at lade RS beholde en kopi af certifikatet i en begrænset tidsperiode (indtil det må antages, at klienten har hentet dette).

På Figur 22 ses et sekvensdiagram, der viser oprettelsesproceduren og herunder ses registreringsserverens opgaver undervejs. Der modtages en:

1. *RegistrationRequestPacket(registrationRequest)* (trin 1 i protokollen) RS udfører nu følgende:
    a. Genererer et stort tilfældigt tal: *activationNumber*.
    b. Opretter et element i *PendingAccounts*-listen indeholdende *activationNumber* samt det medsendte *CertificateSigningRequest*.
    c. RS sender en *RegistrationInfoPacket* til klienten indeholdende: *activationNumber+1*

2. *ActivationPacket(activationNumber)* (trin 4 i protokollen) RS verificerer signaturen af *activationNumber* og kontrollerer, at denne stammer fra en gyldig pengeserver. Hvis (og kun hvis) dette er tilfældet, finder RS *activationNumber* i *pendingAccounts*-listen og genererer det tilhørende certifikat.
3. *RegistrationInfoPacket(activationNumber)* (trin 5 i protokollen) RS slår *activationNumber* op i *pendingAccounts*-listen og sender det tilhørende certifikat til klienten i en *ActivationCompletedPacket(Certificate)*. Opstod der fejl i forbindelse med udstedelse af certifikatet, sendes en *StatusPacket(false)* til klienten. Elementer i *PendingAccounts*, der er mere end én måned gamle slettes.



Figur 22 - Oprettelsesproceduren, hvor både klient, pengeserver og oprettelsesserver er involveret.

I ovenstående protokol benyttes pakken *RegistrationInfopacket* fire gange nemlig i trin 2, 3, 5 og 8. Her vil det som angriber være oplagt med et replay-angreb (der kan foregå på trods af den kryptografiske session). Da pakkerne i trin 2, 3 og 5 sendes i forskellige kryptografiske sessioner, vil det her ikke være muligt at ombytte pakker. Til gengæld vil pakken fra trin 2 kunne sendes af en angriber i trin 8, uden at registreringsserveren kan se forskel. Derfor sendes i trin 2 en pakke, hvor der er lagt 1 til aktiveringsnummeret. Klienten kan ved modtagelse trække 1 fra og have det korrekte aktiveringsnummer. Netværkspakkerne i trin 2 og 8 vil dermed være fuldstændig forskellige pga. kryptografien og dermed være sikret mod dette replay-angreb. Det skal dog her bemærkes, at en angriber egentlig ikke vil kunne opnå noget ved dette angreb, da certifikatet denne kan tilegne sig er offentligt og derudover krypteret.

Et andet sted, hvor en angriber kan benytte et replay-angreb, er i trin 3. Her kan en angriber gentagne gange sende denne netværkspakke til CES i håb om, at der hver gang hæves et oprettelsesbeløb på brugerens konto. For at sikre imod dette skal pengeserveren kun hæve oprettelsesbeløbet én gang for hvert aktiveringsnummer (og kun kontakte registreringsserveren én gang). Modtages det samme aktiveringsnummer igen, skal serveren ikke foretage sig andet end at sende en *StatusPacket(true)* tilbage. På denne måde vil der være sikret mod denne type angreb.

I trin 6 i protokollen sender RS enten certifikatet eller en fejlbesked til pengeserveren. Fejl kan f.eks. opstå, hvis klienten forsøger at registrere en email-adresse, som allerede er

registreret i systemet. I trin 8 og 9 er certifikatet udstedt succesfuldt og sendes til klienten efter dennes forespørgsel. Ved at pengeserveren (i trin 5) henter brugerens certifikat, *ved* denne i fremtiden, at certifikatet tilhører netop denne klient. Denne information udnyttes under indløsning af mønter, hvor ejerskab af mønter kontrolleres af VS. At CES kan sammenkæde en brugers identitet med dennes certifikat, betyder ikke noget for anonymiteten for den enkelte bruger i systemet, som forklaret i afsnit 6.5.

6.9 Klientdesign

Protokollerne, der benyttes når klienten skal kommunikere med serverne, er nu blevet gennemgået. Derudover er serversiden blevet beskrevet i detaljer. Herefter følger en beskrivelse af klientapplikationen. Klienten er bl.a. en emailproxy og det beskrives indledningsvis, hvordan proxyen understøtter de mest gængse protokoller til afsendelse og modtagelse af email. Derefter beskrives programmets interne struktur og hvordan proxyfunktionalitet, brugerinteraktion og serverkommunikation integreres i klienten.

6.9.1 Protokoller

I det følgende beskrives protokollerne for overførsel af email, som proxyprogrammet understøtter. Til afsendelse af email drejer det sig om SMTP (*Simple Mail Transfer Protocol*) og til modtagelse om POP3 (*Post Office Protocol*) og IMAP (*Internet Message Access Protocol*).

6.9.1.a Afsendelse af email

SMTP

Den mest populære protokol til afsendelse af email i dag er SMTP (beskrevet i RFC 821). Protokollen er designet, så denne relativt nemt kan benyttes af både maskiner og mennesker. Protokollen understøtter kun én forbindelse mellem server og klient. En server vil aldrig foretage sig noget uden en forudgående forespørgsel fra klienten. Hver forespørgsel fra klienten besvares af serveren med en trecifret talkode, som angiver om operationen gik godt eller ej. Talkoden efterfølges af valgfri tekst, som udelukkende tjener som letlæselig information for mennesker. I proxyprogrammet ignoreres denne tekst derfor. Kun talkoderne benyttes, når det skal afgøres hvilket sted i protokollen klient og server er nået til.

I det følgende beskrives, hvordan en typisk afsendelse af email foregår vha. SMTP. Når en klient kontakter en SMTP-server, kan klienten angive afsenderadressen efter at have modtaget talkoden 220 (efterfulgt af en velkomstbesked). Godkendes afsenderadressen, vil serveren returnere et svar indeholdende talkoden 250 til klienten. Klienten sender nu modtageradressen og SMTP-serveren vil igen svare med talkoden 250, såfremt modtageradressen accepteres. Afvises enten afsenderadresse eller modtageradresse, vil serveren sende talkoden *5xy*, hvor værdien af *x* og *y* afhænger af årsagen til afvisningen. Nedenfor er vist et eksempel på, hvordan en succesfuld emailafsendelse kan foregå:

```
>telnet mail.ishoejby.dk 25
220-Welcome!
>MAIL FROM: jacobi@ishoejby.dk
250 OK
>RCPT TO: krabo@ishoejby.dk
250 ACCEPTED
```

Serveren har her godkendt afsender- og modtageradresse og er klar til at modtage indholdet af emailen fra klienten i MIME-format (*Multipurpose Internet Mail Extensions*, jvf. RFC 2045). Klienten kan påbegynde afsendelse af denne, efter at følgende kommando er sendt til mailserveren:

```
>DATA
354 Enter message, ending with "." on a line by itself
```

Proxyprogrammet vil lade trafik mellem klient og server passere uhindret indtil netop dette punkt i protokollen. Så snart talkoden 354 modtages fra serveren ved proxyprogrammet, at det næste serveren forventer som input, er selve emailen. Proxyprogrammet overtager her midlertidigt kommunikationen med klienten og lader mailserveren vente. Proxyprogrammet kommunikerer med klienten, indtil hele beskeden er modtaget. Dette er sket, når klienten sender et punktum på en linie for sig selv, altså:

`<CRLF>.<CRLF>` (hvor <CRLF> står for Carriage Return Line Feed)

Når proxyprogrammet har modtaget e-mailen fra klienten, vedhæftes en mønt på denne. Email, inklusiv vedhæftning, sendes til serveren i henhold til protokollen. Når emailen er succesfuldt modtaget af serveren, vil denne give et svar begyndende med talkoden 250. Dette svar videresendes til mailklienten, således at brugeren informeres om den succesfulde afsendelse af emailen. Når dette er sket, får klient og server igen lov at kommunikere frit, så en eventuel ny email kan afsendes.

Proxyprogrammet understøtter således alle SMTP-servere og klienter, som opererer i overensstemmelse med de retningslinier, der er angivet i RFC 821.

6.9.1.b Modtagelse af email

POP3

POP3 (Post Office Protocol vers. 3) er en simpel protokol, der tillader email-klienter at hente emails fra en enkelt elektronisk postkasse (*mailboks*). Klienten kontakter en POP3-server og kan herefter spørge efter en liste over modtagne emails, hente og slette disse fra postkassen på serveren. POP3 er en meget udbredt protokol f.eks. hos internetudbydere, der benytter denne til levering af email til de enkelte brugere.

Protokollen er som SMTP en 'forespørgsel-svar'-protokol, men hvor hver forespørgsel besvares med *+OK*, *-ERR* eller en emailbesked i MIME-format. At protokollen er meget simpel betyder dog, at den har en række begrænsninger. Eksempelvis er det ikke muligt kun at hente udvalgte dele af en email (f.eks. headerne). Dette betyder, at al mailhåndtering foregår, når posten *er* hentet ned på klientmaksinen. POP3-protokollen

understøtter heller ikke 'ny mail'-notifikation, altså at serveren kontakter klienten, når der ankommer emails på serveren. Klienten er således nødt til at spørge (poll'e) POP3-serveren med jævne mellemrum, hvis det er nødvendigt, at nyankomne emails læses hurtigt. I det følgende beskrives det, hvordan en typisk modtagelse af email kan foregå vha. POP3.

Brugeren er autentificeret overfor mailserveren, når denne har sendt hhv. brugernavn og password til denne og fået svar indeholdende "+OK" tilbage (jvf. RFC 1081):

```
>telnet mail.ishoejby.dk 110
+OK <4c7143b63f83fbb10a78c77cd198f95c@cust-serv-02-nic.arrownet.dk>
USER krabo@ishoejby.dk
+OK Tell me your password.
PASS *******
+OK Welcome aboard! You have exactly one message.
```

Brugeren har her autentificeret sig succesfuldt overfor serveren og serveren meddeler, at brugeren har én ny email i den elektroniske postkasse. Emailen kan herefter hentes og til sidst slettes fra serveren på følgende måde:

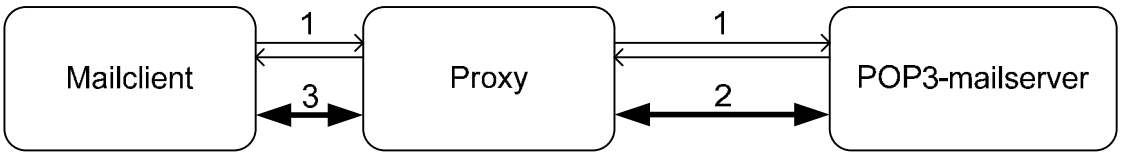
```
RETR 1
+OK Message follows
Message-ID: <000801c50e95$6521ff50$540010ac@krabo1>
From: "Jacob" <jacobi@ishoejby.dk>
To: <krabo@ishoejby.dk>
Subject: SpamCash
MIME-Version: 1.0
Content-Type: text/plain;
        format=flowed;
        charset="iso-8859-1";
        reply-type=original
Content-Transfer-Encoding: 8bit

Hej Lasse.
Har du hørt om den helt nye antispam-løsning, SpamCash.
Jeg tror den holder.
Mvh.
Jacob
.

dele 1
+OK Done.
quit
+OK Done
Connection to host lost.
```

Proxyprogrammet vil lade trafik mellem klient og server passere uhindret, indtil autentifikation af mailklienten har fundet sted (altså til umiddelbart efter at serveren har godkendt password). Når dette sker, overtager proxyprogrammet kommunikationen med mailserveren og lader klienten vente. Efter overtagelse sørger proxyprogrammet for at hente samtlige emails fra mailserveren og filtrere dem. E-mønterne i de emails, der passerer filteret gemmes. Når dette er sket, vil proxyprogrammet afslutte forbindelsen

med mailserveren og overtage dennes rolle. Mailklienten kan herefter hente emails fra proxyen vha. POP3. Proxyen agerer altså herefter POP3-server overfor klienten, så denne kan afhente emails. Ovenstående protokol for afhentning af emails er illustreret på Figur 23.



Figur 23 – Afhentning af emails via proxy. Tallene angiver i hvilken rækkefølge kommunikationen foregår. De tynde pile (1) angiver autentifikationsprocessen mellem mailklient og mailserver. De tykke pile angiver kommunikationen i forbindelse med overdragelse af emails fra hhv. mailserver til proxy (2) og fra proxy til mailklient (3).

Proxyprogrammet understøtter således alle POP3-servere og klienter, som opererer i overensstemmelse med de retningslinier, der er angivet i RFC 1081. Der introduceres altså ingen begrænsninger ved indførelsen af proxyen (udover en smule forsinkelse).

Når POP3 benyttes, vil mailhåndtering foregå *offline.* Når emails er hentet ned, er det nemlig ikke længere nødvendigt at være forbundet til mailserveren. Dette kan både være til fordel og til ulempe. Skal de pågældende emails kun kunne læses på én computer af én bruger, er det umiddelbart bekvemt, men kan blive et problem i et system, hvor flere deler samme postkasse. I sidstnævnte tilfælde vil en central, *online* mailhåndtering være at foretrække. Denne funktionalitet tilbydes i IMAP og denne protokol gennemgås i følgende afsnit.

IMAP

IMAP4 (*Internet Message Access Protocol vers. 4*) tilbyder i modsætning til POP3, at email kan håndteres direkte på serveren (*online*). Mailhåndtering kan således finde sted, uden at emails behøver blive sendt frem og tilbage mellem server og klient. IMAP tilbyder desuden, at brugeren kan have en række 'foldere' på serveren, hvor email kan placeres. Dette er nyttigt, hvis man f.eks. ønsker at sortere indkomne emails i forskellige kategorier alt efter indhold. IMAP er især nyttigt, hvis man ønsker at tilgå sine emails fra flere forskellige computere, da alle emails altid er placeret centralt på serveren. Dette giver også mulighed, for at flere brugere kan tilgå postkassen, uden at de hver især er nødsaget til at hente emails ned lokalt. Kommer der nye emails på serveren, vil IMAP-servere give tilsluttede klienter besked om dette. En ulempe ved IMAP er dog, at det er nødvendigt for klienterne at være online, for at kunne tilgå deres emails. Dette er et mindre problem i dag, hvor mange efterhånden har ADSL-forbindelser eller tilsvarende.

I modsætning til POP3, hvor det altid er klienten, der giver en forespørgsel og afventer svar, kan det i IMAP forekomme, at det er serveren, der kontakter klienten. Derudover kan udføres parallelle aktiviteter ved at klienten opretter flere forbindelser til serveren. Dette håndteres af proxyen ved at lytte efter forespørgsler både hos klienten og serveren

på alle forbindelser samtidig. Vi ønsker at filtrere alle emails på serveren, før klienten får adgang til denne. Derfor gribes der ind i protokollen to steder.

- Når serveren giver besked om ny email
- Når klienten ønsker adgang til en folder på serveren (hvor der muligvis er ny email)

Først behandles tilfældet, hvor serveren giver klienten besked om nyankomne emails. Da SpamCash-systemet skal være transparent for brugeren, skal en sådan besked blokeres af proxyprogrammet og undersøges nærmere. Dette skyldes, at de nyankomne emails *kunne* være spammails. I så fald skal disse slettes fra serveren og klienten skal *ikke* underrettes. RFC 3501 beskriver IMAP (vers.4 rev.1) og i afsnit 7.0 er det beskrevet hvordan IMAP-servere skal reagere på nyankomne emails. Nedenfor er angivet et uddrag:

```
If new messages are found, the server sends untagged EXISTS and RECENT
responses reflecting the new size of the mailbox.  Server
implementations that offer multiple simultaneous access to the same
mailbox SHOULD also send appropriate unilateral untagged FETCH and
EXPUNGE responses if another agent changes the state of any message
flags or expunges any messages.
```

Derfor lyttes der efter, om serveren sender EXISTS og RECENT. Sker dette, mens forbindelsen er i IDLE tilstand (dvs. beskederne er ikke en del af en klientforespørgsel), sendes kommandoerne som sagt *ikke* direkte til klienten. I stedet filtreres Inboks-folderen for eventuelle spammails. Efter filtrering er foretaget, åbnes der igen for kommunikationen gennem proxyen. I forbindelse med filtreringen vil proxyen have sendt en EXPUNGE kommando til serveren for at fjerne eventuelle slettede mails fra Inboks-folderen. Dette vil betyde (jvf. RFC 3501), at IMAP-serveren sender en 'untagged' FETCH eller EXPUNGE-kommando til klienten. Klienten vil dermed opdage, at der er kommet ny mail. Skulle det ske, at IMAP-serveren ikke helt overholder RFC3501, og altså ikke får sendt en FETCH eller EXPUNGE til klienten efter opdatering af folderen, vil dette blot betyde, at klienten først får besked om nye emails på et senere tidspunkt.

På denne måde vil proxyprogrammet være transparent overfor brugeren, når serveren giver meddelelse om nyankomne emails. Den anden mulighed er, at klienten selv spørger serveren om nyankomne emails. Dette sker ved, at brugeren vælger at se indholdet af en ny folder på serveren. Her skal proxyprogrammet (ligeledes transparent) sørge for at filtrere indholdet af denne, før emailklienten får adgang.

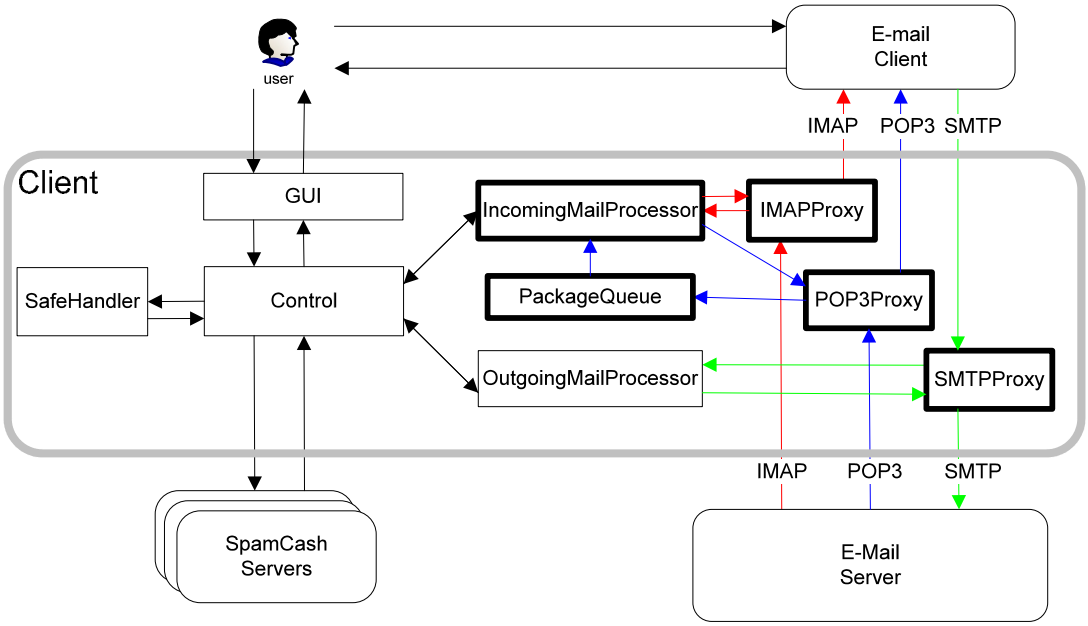
Dette foregår ved at der lyttes til trafikken mellem klient og server, indtil en 'SELECT <*folder*>' sendes fra klienten (hvor <*folder*> er en folder på serveren). Denne og eventuelle efterfølgende kommandoer vil blive holdt tilbage, indtil proxyprogrammet har filtreret den pågældende folder. Filtreringen foregår ved udelukkende at hente mønt, afsenderadresse og Message-ID på hver email. Ud fra mønt og afsenderadresse afgøres det, om emailen skal slettes eller ej. På de emails, der ikke slettes, gemmes Message-ID, så der ikke udføres ovenstående kontrol på disse, næste gang folderen skal filtreres. Så snart folderen er filtreret, får "Select <folder>"-kommandoen (og eventuelle efterfølgende kommandoer) lov at passere til serveren og klienten vil så få indholdet af den filtrerede folder at se. Det bemærkes her, at der er en minimal risiko for, at spam får

lov at slippe igennem. Skulle det ske, at en spammail lander på serveren i det meget lille tidsrum, der går fra filtreringen er afsluttet til proxyprogrammet lader kommandoer fra klienten slippe igennem til serveren, vil denne modtages af klienten. Dette har en angriber dog ikke mulighed for at udnytte, da denne aldrig vil kunne vide, hvornår en klient vælger at se indholdet af sine foldere. En angriber vil derfor ikke kunne 'time' sit angreb. Af denne grund har vi valgt at se bort fra denne forsvindende risiko. For en god ordens skyld skal det siges, at dette problem ikke er til stede, når der benyttes POP3.

De indgreb i protokollen, der foretages af proxyprogrammet, sker altså transparent overfor brugeren (mailklienten). Ovenstående indgreb er samtidig de eneste, der foretages af proxyprogrammet. Al anden kommunikation får lov at passere uhindret mellem klient og server. Den samlede funktionalitet af proxyprogrammet vil dermed være transparent overfor brugeren, *også* når der kommunikeres med IMAP-servere. Proxyprogrammet understøtter derfor alle IMAP4-servere og klienter, der overholder retningslinierne angivet i RFC 3501.

6.9.2 Klientens struktur

Det er nu blevet fastlagt, hvorledes emailhåndtering skal fungere i de pågældende protokoller. I dette afsnit vil der blive fokuseret på klientapplikationens struktur. Indkommende og udgående emails håndteres af objekterne *IncomingMailProcessor* og *OutgoingMailProcessor*. Denne centralisering giver en simplere struktur og gør behandling af email uafhængig af hvilken protokol, der anvendes. For at programmet skal kunne håndtere forespørgsler i alle protokoller samtidig, skal programmet indeholde tre proxytråde. Til at sammenbinde alle dele i klientapplikationen haves objektet Control. Dette objekt er centrum i klientprogrammet og indeholder variable og kontrollen med andre tråde. På Figur 24 ses programmets interne struktur illustreret.



Figur 24 – Klientapplikationens interne struktur. Boksene med runde hjørner symboliserer applikationer, og boksene med skarpe hjørner klientapplikationens interne objekter. Pilene viser kommunikation. En emails vej gennem proxyen er vist med hhv. rød, blå og grøn for IMAP, POP3 og SMTP. Objekterne markeret med **fed** udgør selvstændige tråde.

På figuren ses det, at al emailtrafik løber gennem objekterne *IncomingMailProcessor* og *OutgoingMailProcessor*. Emails, der hentes vha. POP3, videregives til *IncomingMailProcessor* via en kø, *PackageQueue*. Dette er fordelagtigt, pga. måden, hvorpå proxyprogrammet arbejder med POP3-protokollen. Ved at indføre en kø er det nemlig muligt at hente én email *samtidig* med, at *IncomingMailProcessor* behandler en tidligere hentet email. Er der tale om mange og/eller store emails, kan det ske at proxyen ikke når at hente og behandle alle emails, før der opstår timeout i klienten. Her vil brugeren blot skulle trykke på 'Hent mail' igen, hvorefter emailen leveres. Hastigheden hvorpå proxyen henter email fra serveren, vil her afgøre hvor mange emails, der når frem i første forsøg. Køen optimerer hastigheden og sikrer, at flest mulige emails når ud til klienten i første forsøg. Når klienten læser email vha. IMAP, vil proxyen ikke hente hele emails ved filtrering, men blot headere. Da filtreringshastigheden hermed er relativt høj, virker en kø overflødig og benyttes derfor ikke her. Når klienten sender emails vha. SMTP, vil proxyen behandle disse én ad gangen og der er i princippet ingen begrænsning på overførselshastigheden mellem klient og proxy. Derfor findes heller ingen kø her. Det skal her nævnes, at denne struktur kræver to identiske instanser af *IncomingMailProcessor*: Én til at modtage fra *PackageQueue* og én til at modtage kald fra *IMAPProxy* (se Figur 24).

Control-objektet har udover mailhåndteringsklasserne også forbindelse til systemets servere over netværket og til brugeren via et GUI-objekt. Derudover håndteres e-penge vha. et *SafeHandler*-objekt, der håndterer elektroniske penge og gemmer disse på harddisken i en 'pengeskabs-fil' (Safe). På diagrammet på Figur 24 er kun medtaget de

vigtigste objekter for at give et overblik. Mange af objekterne indeholder flere dele til at håndtere forskellig funktionalitet.

Når det skal besluttes, om en opgave skal udføres i en separat tråd eller ej, er følgende betragtning vigtig at have for øje:

- Jo flere tråde, desto sværere testfase og dermed større risiko for fejl.

Det er vigtigt at holde antallet af tråde på et minimum. Dette skyldes, at dét at arbejde med flere tråde samtidig medfører kritiske sektioner og risiko for løkker på tværs af tråde. For at kunne fastslå det minimale antal nødvendige tråde i klienten, kan følgende betragtning benyttes. Er det *nødvendigt*, at to aktiviteter foregår parallelt, kræver dette to tråde, ellers ikke. Det er f.eks. nødvendigt at kunne håndtere emailforespørgsler i de forskellige protokoller samtidig. Derfor findes en tråd for hver protokol, som det også fremgår af Figur 24, hvor kanten, på objekter der kører som separate tråde, er **fed**[4]. Udover protokol-trådene kræver kø-mekanismen, at *PackageQueue* og *IncomingMailProcessor* (den instans der benyttes til POP3) kører i separate tråde. Til sidst skal den grafiske bruger grænseflade køre uafhængigt af de herover nævnte tråde. Brugergrænsefladen kan derfor indgå som en del af programmets hovedtråd, der også indbefatter *Control* og de resterende objekter, der er forbundet til denne.

Klientapplikationens grundlæggende struktur er nu fastlagt. Hvordan de enkelte programdele samarbejder, vil blive udspecificeret i det følgende.

6.9.3 Klientens funktionalitet

Klientens funktionalitet vil i dette afsnit blive udspecificeret yderligere. Dette gøres vha. sekvensdiagrammer. Der tages udgangspunkt i de centrale funktioner, der blev gennemgået ved udarbejdelse af use cases (se afsnit 6.6). Der er dannet diagrammer ud fra de use-cases, der ikke er umiddelbare at implementere uden yderligere specifikation. Følgende liste angiver hvilke diagram-afsnit, der findes i det følgende. Efter titlen er angivet hvilket use-case nummer diagrammet hører til:

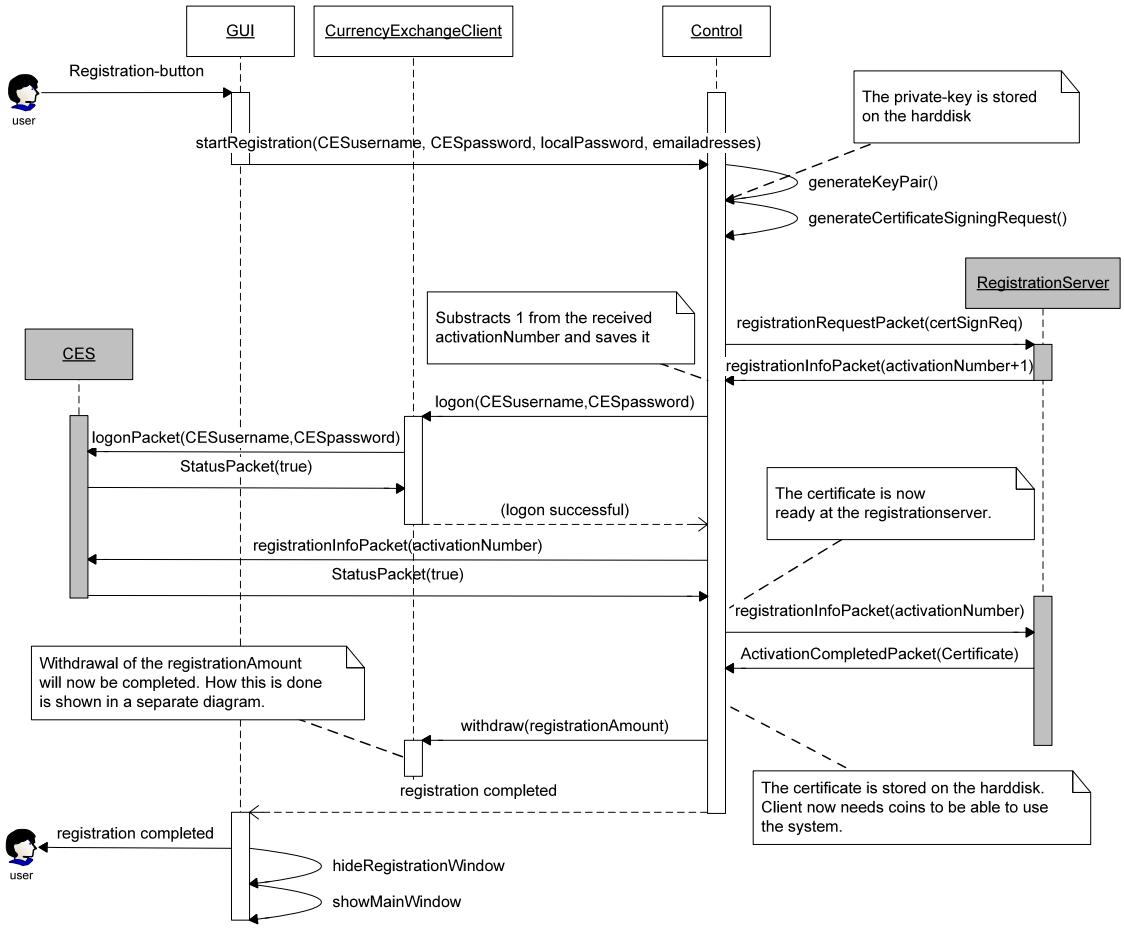
- Registrering (Use Case 1)
- Se pengestatus (Use Case 3)
- Afsendelse af email (Use Case 4)
- Modtagelse af email (Use Case 5)
- Indkasser “SpamCash” (Use Case 6)
- Hæv penge (Use Case 7)
- Indsæt penge (Use Case 8)

Use case 2, 9, 10 og 11, der omhandler hhv. konfiguration af IP-adresser, opdatering af whitelist og programnedlukning, beskriver relativt simple funktioner og er derfor ikke beskrevet yderligere i dette afsnit.

[4] Internt i IMAPProxy benyttes dog, hvis det bliver nødvendigt flere tråde til forbindelser fra klient til server

6.9.3.a Registrering (Use Case 1)

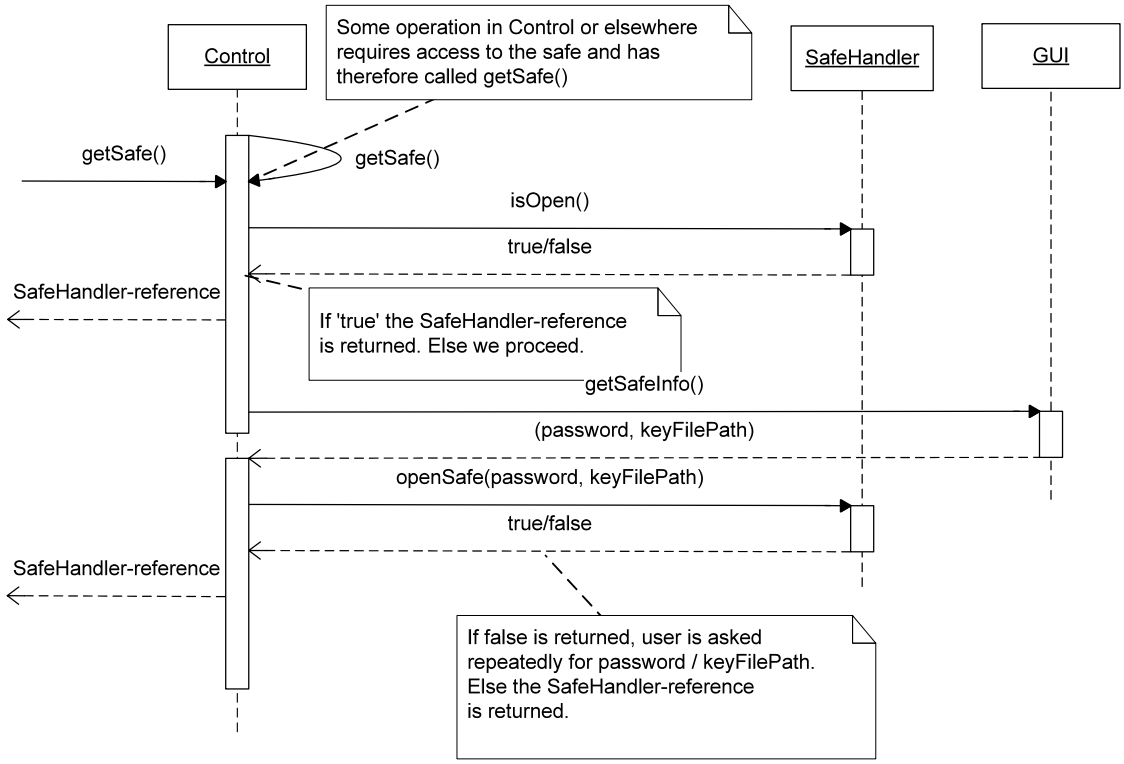
På Figur 25 ses sekvensdiagrammet for oprettelse i systemet.



Figur 25 – I diagrammet er det vist, hvilke funktionskald der udføres og hvilke pakker som sendes i systemet under registreringsprocessen. Udover de tre objekter i selve klienten (Control, GUI og CurrencyExchangeClient) er der medtaget to servere i diagrammet. Disse er markeret med grå baggrund.

Pengeskabsadgang

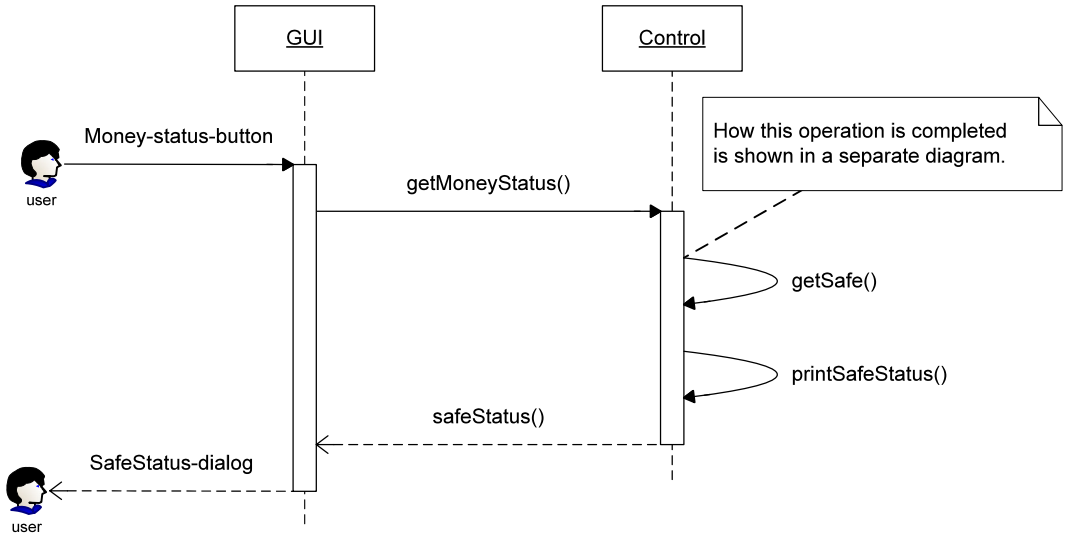
Tilgang til pengeskabs-filen på brugerens maskine sker via getSafe()- funktionen. Funktionen benyttes i de følgende diagrammer, men vil for overskuelighedens skyld blot være angivet med en pil i disse, som så dækker over den funktionalitet, der er angivet i nedenstående diagram (Figur 26).



Figur 26 – getSafe()-funktionen. Denne returnerer en reference til et SafeHandler-objekt forudsat at korrekt nøglesti og password angives. Hvis ikke pengeskabs-filen (Safe) allerede er åben, vises en indtastningsboks (getSafeInfo()) til brugeren via GUI.

6.9.3.b Se pengestatus (Use Case 3)

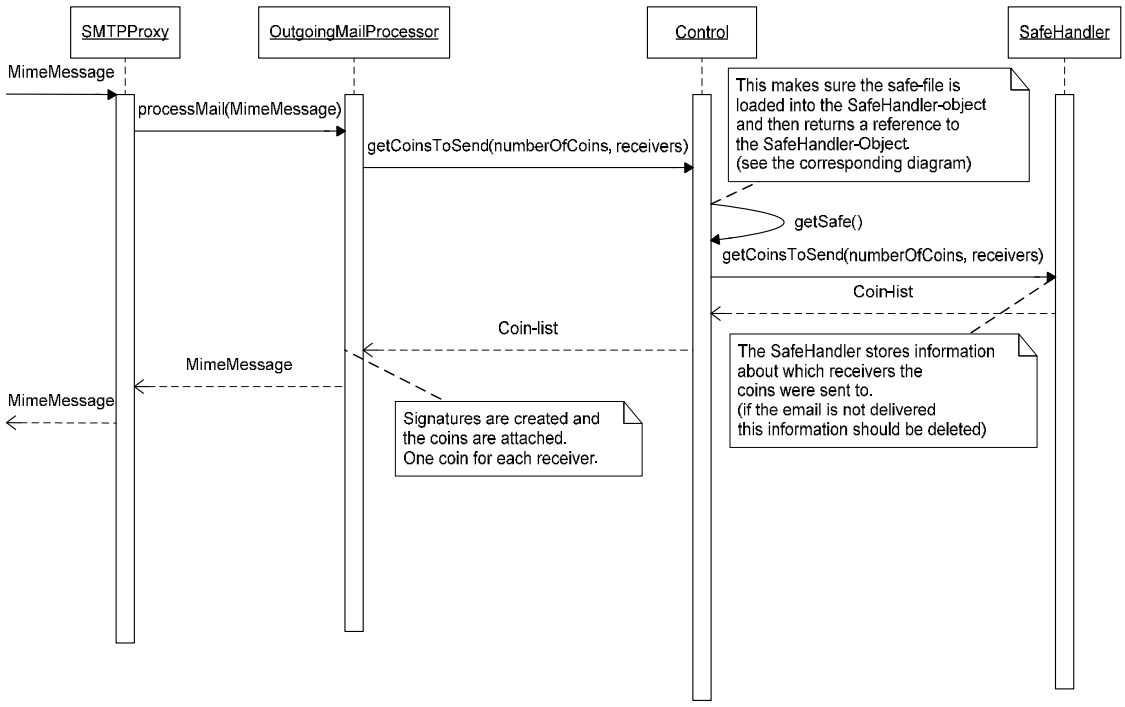
På Figur 27 ses sekvensdiagrammet for situationen, hvor brugeren ønsker at se indholdet af pengeskabet og derfor trykker på ’Money-status’-knappen.
Bemærk at getSafe()-funktionen benyttes i forbindelse med dette.



Figur 27 - Diagrammet viser hvad der sker, når brugeren trykker på moneystatus-knappen for at få en status over sin (elektroniske) pengebeholdning.

6.9.3.c Send email (Use Case 4)

På Figur 28 ses sekvensdiagrammet for afsendelse af email.



Figur 28- Diagrammet viser hvordan vedhæftning af signaturer og mønter foregår, når der sendes email gennem proxyen.

Det uddybes nu præcis hvilke signaturer, der dannes af *OutgoingMailProcessor* inden email afsendes, samt hvordan mønthåndteringen foregår i *SafeHandler*. Begreberne *overførselssignatur* og *integritetssignatur* vil blive nævnt i det følgende. Disse dækker over de signaturtyper under samme navn, som blev indført i afsnit 6.3.2.a. Udover disse indføres en ny type signatur kaldet *emailsignatur*. Emailsignaturen sikrer integriteten af emailens indhold ved at knytte dette sammen med den vedhæftede mønt:

Emailsignatur: $Sig_{Afsender}$(emailindhold, mønt)

Dermed sikres det, at en angriber ikke kan udskifte indholdet af en email i transit (med f.eks. en spam-meddelelse), uden at dette opdages af modtager. Her er emailindholdet ikke hele MIME-beskeden, da dennes struktur og visse dele af indholdet varierer under overførsler. Emailindholdet, der benyttes i signaturen, er i stedet udvalgte felter såsom *emne, fra, til, dato* og *beskedteksten*. Hvilke felter, der medtages i denne signatur bør besluttes under implementation.

Når en email afsendes sker følgende:

- En A-mønt omdannes til hhv. en B- og en C-mønt.
- C-mønten gemmes i pengeskabet.
- Overførselssignatur dannes (se evt. afsnit 6.3.2.a).
- Emailsignatur dannes.
- Overførselssignatur, emailsignatur, certifikat og B-mønt vedhæftes emailen, som herefter er klar til afsendelse.

6.9.3.d Modtagelse af email (Use Case 5)

Når det skal afgøres, om en email skal have lov at passere gennem proxyen ud til mailklienten gøres dette vha. funktionen *keepMail(MimeMessage)*, som kaldes af *IncomingMailProcessor*. *KeepMail*-funktionen udfører følgende kontroller og verifikationer:

- Det kontrolleres, at afsenders certifikat ikke findes på den opdaterede blackliste.
- Det kontrolleres, at mønten ikke har skiftet ejer flere end det tilladte antal gange.
- Det kontrolleres, at mønten er udstedt af en pengeserver ved at verificere dennes signatur af mønten.
- Det kontrolleres, at mønten ikke har været kopieret ud fra kriterierne i afsnit 6.4.2.b.
- Det medsendte certifikat verificeres.
- Den vedhæftede overførselssignatur verificeres vha. medsendte certifikat.
- Den vedhæftede emailsignatur verificeres vha. medsendte certifikat.
- Transaktionslistens integritet verificeres.
- Er alle ovenstående punkter gennemført succesfuldt, opdateres transaktionslisten i overensstemmelse med beskrivelsen i afsnit 6.3.2.a:
    - Overførselssignatur tilføjes (type 1)
    - Integritetssignatur dannes og tilføjes (type 2)

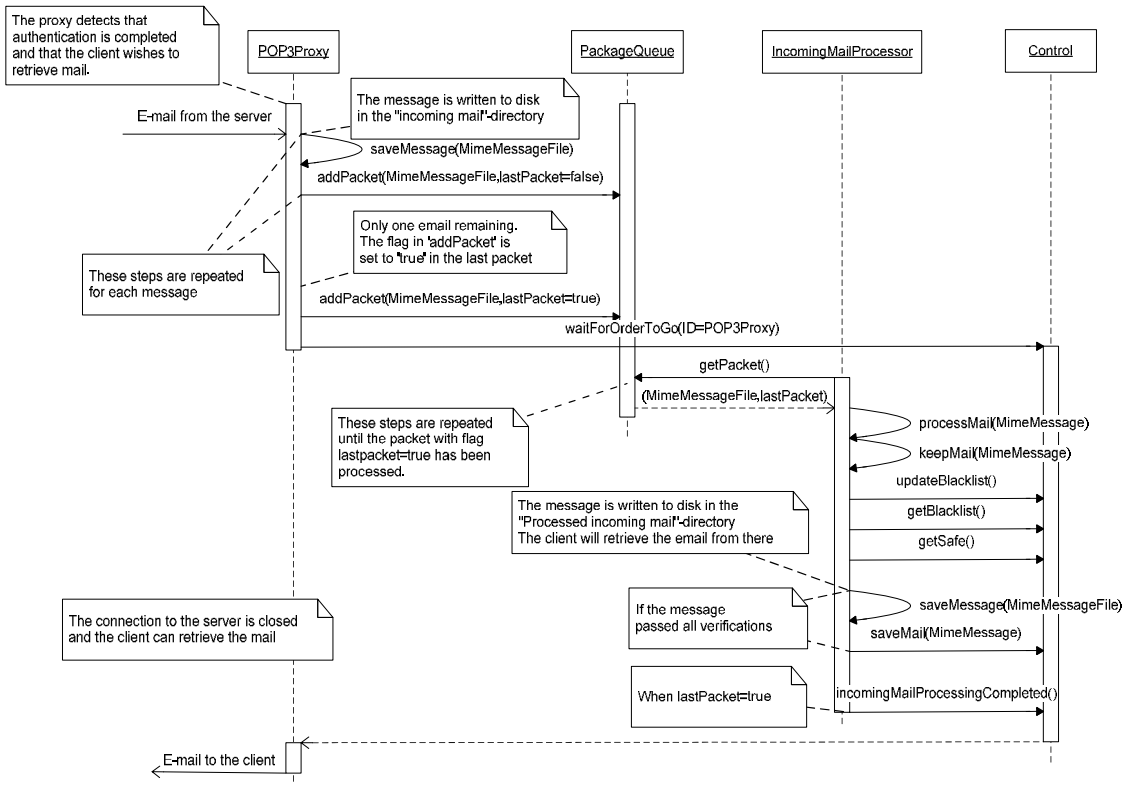
Ovenstående fremgangsmåde sikrer, at kun gyldige mønter accepteres ved modtagelse samt at transaktionslisten opdateres korrekt efter fremgangsmåden beskrevet i afsnit 6.3.2.a.

En modtaget email leveres således kun til brugerens indbakke, hvis mindst ét af følgende punkter er opfyldt:

- Keepmail-funktionen gennemføres med succes (Alle kontroller og verifikationer var succesfulde)
- Afsenders emailadresse er på whitelisten.

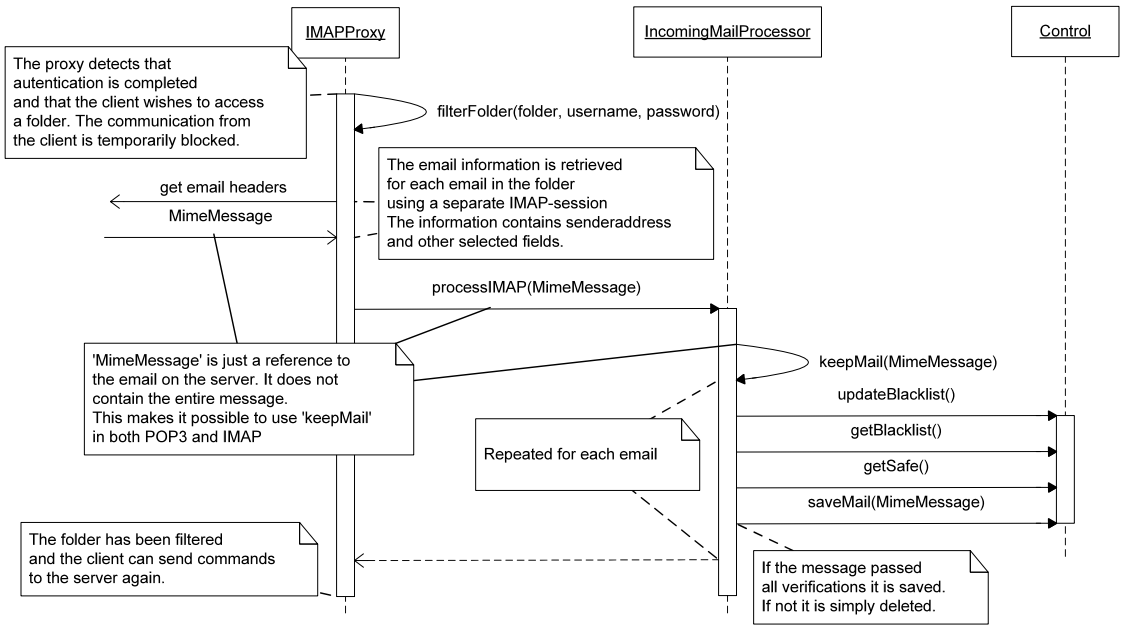
KeepMail-funktionen bruges både når modtagelse af email foregår vha. IMAP, og når den foregår vha. POP3. Diagrammerne for modtagelse vha. POP3 og IMAP er angivet i de følgende to afsnit.

Modtagelse af email i POP3 (Use Case 5)



Figur 29 – Diagrammet viser hvordan filtrering af indkomne emails foregår (i POP3). Det ses at *keepMail* som beskrevet ovenfor, kaldes af *IncomingMailProcessor*-objektet. De tre pile angivet efter dette kald foregår i selve *keepMail*-funktionen, da *keepMail* skal bruge en opdateret blacklist og have adgang til pengeskabet (hvor mønter gemmes undervejs).

Modtagelse af email i IMAP (Use Case 5)

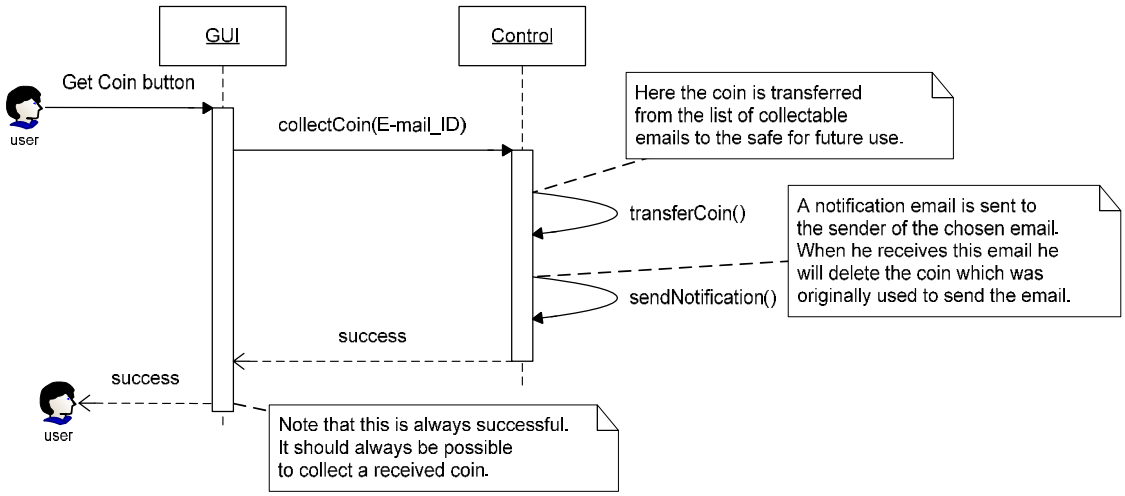


Figur 30 – Diagrammet viser hvordan filtrering af indkomne emails foregår (i IMAP). Det ses at *keepMail* som beskrevet ovenfor kaldes af *IncomingMailProcessor*-objektet. De tre pile angivet efter dette kald foregår i selve *keepMail*-funktionen, da *keepMail* skal bruge en opdateret blacklist og have adgang til pengeskabet (hvor mønter gemmes undervejs).

Funktionen *updateBlacklist()* kaldes for hver modtagen email. *Control* holder dog øje med, hvornår denne sidst blev opdateret, og *kun* hvis der er gået et vist tidsrum, hentes en opdateret blacklist fra blacklistserveren.

6.9.3.e Indkasser “SpamCash” (Use Case 6)

På Figur 31 ses sekvensdiagrammet for situationen, hvor en bruger opkræver betaling for en email.



Figur 31 – Indløsning af e-penge på modtagne spammails. Der sendes en besked til afsender om, at dennes mønt indløses og at kopien hos afsender derfor skal slettes.

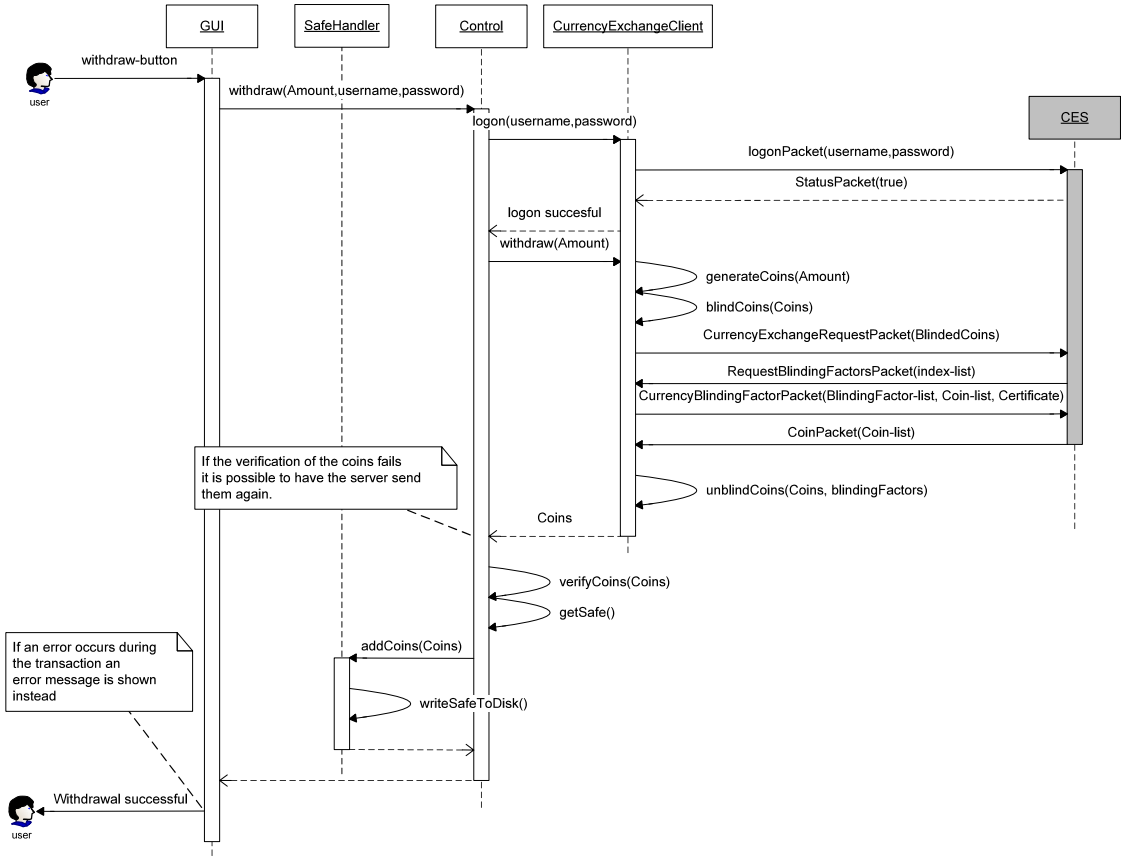
Der sker følgende, når brugeren trykker på ’Get coin’-knappen.

- B-mønten i mailen omdannes til en A-mønt og kan hermed indløses eller benyttes til afsendelse af email.
- En notifikationsmail sendes til afsender, for at fortælle at pengene indløses (og dermed at dennes C-mønt er værdiløs og skal slettes). Notifikationsmailen indeholder en header med en signatur af møntens serienummer og afsenders emailadresse.

Er det ikke muligt at afsende notifikationsmailen af den ene eller den anden grund, vil første punkt stadig gennemføres, da brugeren *skal* kunne opkræve betaling.

6.9.3.f Hæv penge (Use Case 7)

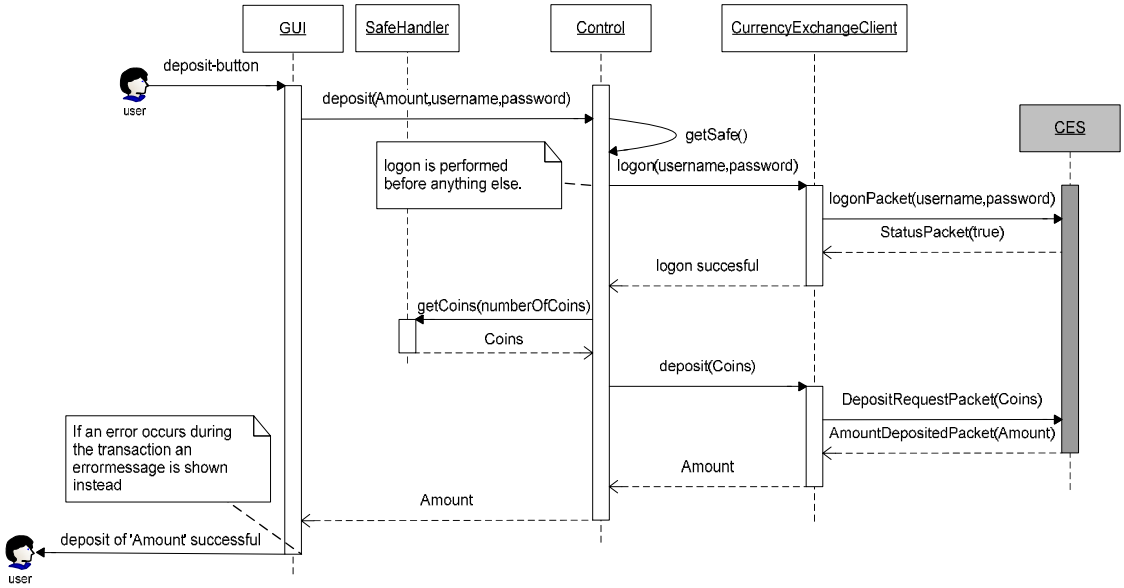
På Figur 32 ses et sekvensdiagram, der viser klientprogrammets opførsel, når brugeren ønsker at hæve penge. Her er, udover objekterne i klientprogrammet, medtaget CES for at vise kommunikationen med denne.



Figur 32 – Diagrammet viser, hvordan klienter hæver elektroniske penge. CES er markeret med grå baggrund, da denne ikke er en del af klienten (proxyprogrammet).

6.9.3.g Indsæt penge (Use Case 8)

På Figur 33 ses et sekvensdiagram, der viser klientprogrammets opførsel, når brugeren ønsker at indsætte penge.



Figur 33 – Diagrammet viser hvordan indsættelse af elektroniske penge foregår. CES er markeret med grå baggrund, da denne ikke er en del af klienten (proxyprogrammet).

6.10 Trusselshåndtering

I dette afsnit vil der blive givet en opsummering af, hvordan truslerne i afsnit 5.6 håndteres af systemet.

Kryptoanalyse

Det antages at de anvendte algoritmer (AES, RSA og SHA) er sikre (se evt. afsnit 6.7.2). Er dette tilfældet, vil det ikke være muligt at foretage succesfulde kryptoanalytiske angreb på systemet.

Reverse engineering

Systemets sikkerhed afhænger ikke af hemmeligholdelsen af proxyprogrammets kildekode. Reverse engineering af programmet og modifikation af dette, vil dermed være formålsløst med henblik på misbrug. Sikkerheden afhænger udelukkende af hemmeligholdelsen af private nøgler og passwords (se evt. afsnit om lokal sikkerhed, 6.7.1).

6.10.1 Økonomisk misbrug

Fabrikation af penge

For at kunne fabrikere e-mønter kræver dette en pengeservers signatur (se evt. afsnit om møntstruktur, 6.3.2). Da vi antager at de anvendte algoritmer er sikre, vil det ikke være muligt for en angriber at danne en sådan, så længe pengeserverens private nøgle holdes hemmelig.

Kopiering af penge

Det vil altid være muligt at kopiere elektroniske penge. Da systemet er offline og understøtter transferability, er det også muligt at *anvende* kopier. Blacklisting-mekanismen sikrer dog udelukkelse og økonomisk straf til kopister som beskrevet i afsnit 6.4.

Hamstring af mønter

Da systemets økonomi er åben, er hamstring af mønter ikke et problem.

Grådige brugere

Grådige brugere, der indløser alle modtagne mønter, vil stå i dårligt lys overfor afsenderne. Dette skyldes at afsender får besked om opkrævning af betalingen via en notifikationsmail (se evt. afsnit 6.3). Dette vil motivere de fleste brugere til kun at opkræve betaling, når der *er* tale om spammail. Et problem er dog, at brugerne kan forhindre notifikationsmails i at blive afsendt, hvorved afsender ikke bliver informeret om opkrævningen. Derudover risikerer afsender at blive blacklistet. Dette problem er ikke løst, men en delvis løsning kunne være at tolerere møntkopiering i mindre omfang.

Møntfiskeri

Dette er et problem, der introduceres så snart der indføres betaling på email. Da der er tale om relativt små pengebeløb, må det antages, at brugerne vil opleve dette som et mindre problem. I afsnit 5.3.1 blev beskrevet hvordan channels kunne bruges til at løse problemet delvist. Det er svært at sige hvor stort problemet vil være, når systemet skal anvendes, men problemet afhænger primært af, hvor kritiske brugerne er i deres brug af emailsystemet.

Tyveri af mønter

Systemet er sikret mod tyveri af mønter. Dette er beskrevet i afsnit 6.3.3.b.

Salg af kopierede mønter

Mønter kan ikke overdrages fra en bruger til en anden udenom systemet som beskrevet i afsnit 6.3.3.b. Foregår overdragelsen af kopier ved emailoverførsel i SpamCash-systemet, vil kopieringen blive opdaget, når mønterne indløses (eller ved stikprøvekontrol) og sælger/afsender udelukkes. Derfor er salg af kopierede mønter ikke et problem.

Modifikation af mønter

Det er ikke muligt at modificere mønter med henblik på misbrug i systemet, som beskrevet i afsnit 6.3.3.a

6.10.2 Netværkstrusler

I de følgende afsnit vil hver enkelt trusselstype blive diskuteret både i forbindelse med emailoverførsel og kommunikation mellem klient og servere i systemet.

Man in the middle

At en angriber opfanger emails i transit mellem to brugere er ikke et problem. Angriberen vil ikke kunne benytte mønten i emailen (afsnit 6.3.3.b). Angriberen vil heller ikke kunne opnå at få afsender blacklistet ved at gensende emailen (afsnit 6.4.2.b). Derudover vil angriberen ikke kunne erstatte indholdet af emailen med f.eks. en spam-meddelelse (afsnit 6.9.3.c).

Et andet sted et man-in-the-middle angreb kan finde sted, er i protokollerne, der benyttes mellem klienter og servere i systemet. De kryptografiske sessioner begrænser her en angribers muligheder og derudover har en systematisk gennemgang af mulige angreb på protokollerne ikke afsløret sikkerhedsproblemer af nogen art (se afsnit 0).

Eavesdropping

En angriber kan læse indholdet af emails sendt med SpamCash-systemet, såvel som dette er muligt i det sædvanlige emailsystem. Betalingen, der er vedhæftet emailen, kan dog ikke misbruges som beskrevet tidligere.

Trafik mellem klienter og servere i systemet er krypteret hvor det er nødvendigt, hvorfor en angriber ikke vil kunne udlede brugbar information her.

Modifikation

Som beskrevet i afsnittet 'man in the middle' ovenfor er modifikation af emailbeskeder i transit ikke mulig uden at dette opdages. Kryptering af netværkspakker mellem klient og servere sikrer, at disse ikke kan modificeres (integriteten er bevaret). Der foretages dog ikke kryptering, når der kommunikeres med BLS (for at reducere dennes arbejdsbyrde). Dette er dog ikke et problem, da eventuel modifikation af pakker sendt fra blacklistserveren kun påvirker blacklisten hos enkelte klienter.

Replay

Som nævnt under afsnittet 'man in the middle' ovenfor, er systemets protokoller blevet gennemgået systematisk for at sikre, at replay-angreb ikke er mulige (se afsnit 0). Det er heller ikke muligt at opnå noget ved et replay-angreb i forbindelse med emailkommunikation, grundet den modtagende klients beskyttelsesmekanisme beskrevet i afsnit 6.4.2.b.

Denial of Service

Denial of service-angreb (DOS) kan ikke forhindres fuldstændig. Dette skyldes at en angriber kan *spoofe* afsenderadressen (IP og MAC) og dermed er det ikke er muligt at filtrere trafik effektivt på serverniveau.

I SpamCash-systemet skal konsekvenserne, af at hver enkelt server angribes, overvejes. Angribes registreringsserveren eller pengeservere, vil dette ikke være kritisk. Dette vil blot betyde, at brugerne ikke kan indløse/hæve penge og at nye brugere ikke kan oprette sig. Er verifikationsservere ude af drift, vil dette heller ikke være kritisk, idet penge-serverne ikke tillader indløsning af e-mønter, før verifikationsserverne igen kan verificere

mønter. Mere kritisk er det, hvis blacklistserveren angribes og kommer ud af drift. Dette vil betyde frit lejde for spammere (der kopierer mønter), indtil serveren kører igen. Når serveren kører igen, vil de, der har kopieret mønter mens serveren var nede, dog blive udelukket. Blacklistserveren kan dog have en række *mirrors*, der indeholder samme funktionalitet og blacklist. Systemet vil dermed være mere robust overfor denne type angreb.

6.10.3 Fysiske trusler

Serverne i systemet skal sikres mod fysiske trusler såsom brand og tyveri. Serverne skal derudover være placeret geografisk langt fra hinanden. Registreringsserver, verifikationsservere og blacklistserver skal placeres i og ejes af en organisation, der er fuldstændig uafhængig af de banker, der administrerer pengeserverne. Dette er essentielt for at sikre brugernes anonymitet i systemet. Overholdes ovenstående retningslinier, vil systemet være sikret mod fysiske trusler.

7 Implementation

Med udgangspunkt i designafnittet (afsnit 6) vil det nu blive gennemgået, hvordan systemet implementeres. Systemet skal kunne bruges af alle og skal derfor ikke være begrænset til kun at kunne køre på ét operativsystem. Vi har valgt at implementere systemet i JAVA netop pga. dets platformuafhængighed og gode indbyggede værktøjer til netværkskommunikation.

Systemet implementeres, så dette fungerer i overensstemmelse med beskrivelserne i designafsnittet. Derfor har vi i det følgende valgt, at koncentrere os om de mere interessante dele af programmet og bruge mindre energi på de dele af programmet, der er relativt trivielle at implementere ud fra beskrivelserne i designafsnittet. I Appendiks H findes kildekoden til programmet med en oversigt over alle klasser. Derudover findes i Appendiks D klassediagrammer for programmets pakker og klasser. Skal programmet senere modificeres eller opdateres, vil det være svært at overskue kodens struktur udelukkende ud fra følgende implementeringsafsnit. Betragtes derimod, udover dette afsnit, diagrammerne i designafsnittet og de to appendiks, vil læseren have et bedre overblik og kunne arbejde med programmet.

7.1 Programoversigt

Der gives nu en kort oversigt over de pakker (*packages*), der findes i programmet. For hver pakke er angivet hvilke dele af programmets funktionalitet, som den pågældende pakke indeholder. Der henvises igen til Appendiks H for en mere fyldestgørende oversigt, hvor samtlige klasser er medtaget. I punktopstillingen er pakkenavne vist med kursiv. Indrykning angiver underpakker.

- *Client*
    - *GUI* – Grafisk interaktion med brugeren håndteres her.
    - *MailHandler* – Vedhæftning og opsnapning af mønter på emails foregår her. Det er også her indkomne mønters gyldighed kontrolleres. Altså generelt håndtering af emails.
    - *MailProxy* – Styring af kommunikation mellem mailklient og mailserver i IMAP, POP3 og SMTP styres her. I denne pakke findes klasser til at lytte til trafikken mellem klient og server, og gribe ind i protokollerne, når dette er nødvendigt.
    - *Safe* – Klientens 'pengeskab'. Dette inkluderer sikker opbevaring af e-mønter, indkomne emails og klientens private nøgle. Her holdes også styr på, hvilke mønter klienten tidligere har modtaget og brugt (så klienten kan bistå med at pågribe kopister). Alt dette gemmes krypteret på harddisken.
- *Currency* - Her findes de klasser, der tilsammen udgør systemets penge, herunder transaktionslisten. Transaktionslisten findes både i en krypteret og en ikke-krypteret form. Desuden findes i denne pakke en *blinded* mønt, som benyttes ved udstedelse af penge.

- *NetworkPackets* – Samtlige netværkspakker som bruges til kommunikation findes her. Herudover findes en *CryptoPacket*, som bruges når netværkspakker skal sendes i krypteret form.
- *Servers*
    - *BLS* – Blacklistserver, kommunikationstråd samt blacklisten er placeret her.
    - *CES* – Pengeserver, kommunikationstråd samt klasser til håndtering af brugernes konti er placeret her.
    - *RS* – Registreringsserver, kommunikationstråd og klasser til opbevaring af registreringsinformation.
    - *VS* – Verifikationsserver og kommunikationstråd er placeret her.
- *Utilities* – Her er placeret generelle værktøjer og dataklasser, som benyttes af de andre pakker i systemet. F.eks. er alle krypteringsværktøjer placeret i klassen `Cryptotools` (beskrives yderligere i afsnit 7.3.3) og det er også i denne pakke, vi finder klassen `Session`, som står for automatisk oprettelse af kryptografiske sessioner, når dette er nødvendigt (afsnit 7.3.4). Her findes også klassen `Debugger`, et output-værktøj, der bl.a. er en stor hjælp i forbindelse med fejlfinding (behandles særskilt i afsnit 7.3.7).

7.1.1 Tredjeparts-software

Vi har nu gennemgået strukturen af den implementerede software. I det følgende vil den tredjeparts-software, der benyttes i systemet blive beskrevet. Det antages at softwaren er sikker (altså at algoritmerne er korrekt implementerede) og at disse ikke indeholder bagdøre.

7.1.1.a BouncyCastle

Grundet eksport-restriktioner i USA understøtter JAVA2SDK pakken ikke som standard kryptering med en styrke, som benyttes i dette projekt. Derfor benyttes en såkaldt *security provider*. Vi har valgt at benytte *Bouncy Castle*[5] som provider, da de leverer et bibliotek, som understøtter de krypteringsalgoritmer, der benyttes i systemet, nemlig RSA, AES og SHA-1. Derudover er *Bouncy Castle* en anerkendt open source provider, så der er grund til at stole på, at algoritmerne er korrekt implementerede og uden bagdøre. Det skal dog nævnes, at der umiddelbart kan skiftes provider i systemet, ved blot at ændre i klassen `Cryptotools`. Dette ville være oplagt at gøre, hvis algoritmerne på et tidspunkt blev understøttet som standard i Java.

Noget som *ikke* understøttes af BouncyCastle, men som er nødvendigt i systemet, er muligheden for at danne blinde signaturer. Derfor har vi valgt at benytte endnu en provider, som leverer netop denne funktionalitet. Dette uddybes i følgende afsnit.

7.1.1.b Logi

For at understøtte blinde signaturer benyttes et bibliotek udviklet af open source provideren *Logi*[6]. I dette bibliotek er blinde signaturer i RSA implementeret således, at

[5] http://www.bouncycastle.org/
[6] http://www.logi.org/logi.crypto/

protokollen beskrevet i afsnit 4.4 kan benyttes. I afsnit 7.3.9 vises det, hvordan protokollen benyttes i praksis i SpamCash-systemet til udstedelse af e-mønter.

7.1.1.c Javamail

I systemet er det i mange tilfælde nødvendigt at modificere emails, når disse afsendes eller modtages via proxyprogrammet. Dette er f.eks. tilfældet, når der skal vedhæftes eller opsnappes mønter på emails. Derfor har vi valgt at benytte Sun's *javamail*[7], som bl.a. gør det nemt at arbejde med emails i MIME-formatet. I SpamCash-systemet benyttes *Javamail* primært til at påhæfte og fjerne mønter på emails i MIME-format. Derudover er det simpelt at afsende emails vha. *javamail*, hvilket udnyttes i forbindelse med generering og afsendelse af notifikationsmails.

7.1.1.d OpenSSL

Registreringsserveren (RS) står for udstedelse af certifikater, som beskrevet i afsnit 6.2.1.b. I SpamCash-systemet benytter vi OpenSSL[8] til at opnå dette. OpenSSL er en kommandobaseret open source certifikatautoritet. Hvordan OpenSSL installeres og konfigureres er beskrevet i installationsvejledningen i Appendiks A. Da OpenSSL benyttes via en normal kommandoprompt, og altså ikke umiddelbart kan styres via et Java-interface, foregår al kommunikation via systemkald. Dette gøres i Java vha. klassen `Runtime`. Ønskes f.eks. system-kommandoen angivet i strengen `command` eksekveret via et javaprogram, kan dette gøres således:

```
Runtime rt = Runtime.getRuntime();
String command = "openssl ...";
rt.exec(command);
```

Javaprogrammet kan dermed gemme et *certificatesigning-request* på disken, eksekvere kommandoen og til sidst indlæse det færdige certifikat fra disken. Hermed kan RS-serveren udstede certifikater vha. OpenSSL. OpenSSL holder styr på at ingen email-adresser kan registreres to gange. Forsøges dette, vil en fejlmeddelelse blive returneret. Der behøves altså ikke eksplicit at blive kontrolleret for allerede registrerede emailadresser i registreringsserveren under oprettelse.

Generering af rodcertifikat og tilhørende privat nøgle foregår i OpenSSL. Certifikater dannet i OpenSSL dannes i X.509 formatet. Dette understøttes af Javas standardbiblioteker, hvorfor certifikater kan indlæses ad denne vej i SpamCash-applikationerne. De private nøgler genereres i Java, dog ikke nøglen hørende til rodcertifikatet. Dette udgør et mindre problem, idet den private nøgle, benyttet under OpenSSL, er gemt i et format, som ikke umiddelbart understøttes af Java. Derfor skal denne først konverteres. Dette gøres vha. følgende kommando i OpenSSL:

```
openssl pkcs8 -topk8 -inform PEM -outform DER -in priv.pem -nocrypt -out priv.der
```

Efter denne engangskonvertering kan certifikatudstedelse og nøglegenerering foregå fuldautomatisk. Ingen af serverne kræver herefter nogen form for administration. Udover

[7] http://java.sun.com/products/javamail/
[8] http://www.openssl.org/

at kunne udstede certifikater, kan RS nu benytte rodcertifikat samt sin private nøgle til etablering af kryptografiske sessioner.

7.1.1.e Base64

Da mønterne skrives direkte fra Javaprogrammet og ned i en email, er det nødvendigt at Base64-kode data først. Dette skyldes at en email under transit ikke må indeholde rå binær data. Derfor benyttes en Base64-implementation af Robert Harder[9]. I Utilities-pakken findes denne implementation i klassen Base64, der indeholder funktionalitet til at kode og afkode data til og fra Base64 formatet.

7.2 Begrænsninger

Af tidsmæssige årsager har vi været nødsaget til at fravælge at implementere visse features i systemet. Fælles for disse er dog, at de stort set problemfrit vil kunne implementeres senere. I det følgende gennemgås de fravalgte features og det forklares kort hvordan man, i en fremtidig version af programmet, vil kunne implementere disse.

7.2.1 Sletning af certifikater

Certifikater slettes aldrig i registreringsserveren (RS). Dette er en service for brugerne i systemet, da disse altid ville kunne generhverve deres certifikat ved at kontakte RS. Efterhånden som brugerskaren vokser vil dette dog betyde, at RS skal have mange tusind certifikater liggende, hvilket går ud over ydeevnen. En løsning kunne være, at der f.eks. månedligt, bliver slettet certifikater, som er mere end én måned gamle. Dette kan gøres ved blot at se på tidspunktet for oprettelse af filen, hvori certifikatet befinder sig.

7.2.2 Justering af mønters værdi

Det er ikke muligt at justere det beløb, der kræves for at få adgang til mailklientens indbakke. Det er altså det samme faste beløb (én mønt), der kræves for at få adgang til alle brugeres indbakke. I en fremtidig version kunne man forestille sig, at proxyprogrammet ville svare tilbage til klienter, som sender med for lille 'frankering', at de må prøve igen med et større beløb. Dette kunne gøres ved at tillade, at flere mønter vedhæftes i hver email. På denne måde vil brugerne kunne bestemme 'prisen' for adgang til deres indbakke.

7.2.3 Emailadresser under oprettelse

Det er kun muligt at registrere sig med én emailadresse pr. oprettelse. Når man er oprettet i systemet, skal man således både sende og modtage på denne ene adresse. Flere adresser kan nemt indgå i certifikaterne, men dette vil kræve ekstra arbejde hos klienter og verifikationsservere, når disse skal verificere transaktionslister i mønterne.

7.2.4 Ejerskab af emailadresser under oprettelse

For at sikre ejerskab af emailadresse(rne) ved oprettelse skal følgende implementeres. Der skal sendes en 'aktiveringsemail' til den (eller de) angivne emailadresser ved

[9] http://iharder.net/base64

oprettelse. Først når brugeren trykker på et link i disse emails aktiveres SpamCash-kontoen. Dette sikrer nemlig, at brugeren har adgang til den pågældende emailkonto. Denne funktionalitet er ikke implementeret. Det antages derfor blot, at brugerne angiver deres egne emailadresser under oprettelsen.

7.2.5 Afsendelse af emails til flere modtagere

Afsendelse af emails til flere modtagere ved benyttelse af BCC og CC-felter i emailen understøttes ikke. Dette ville kræve ekstra funktionalitet i proxyprogrammet, da denne type emails normalt kopieres på den udgående mailserver, altså udenfor proxyprogrammets domæne.

En løsning kunne være, at proxyprogrammet tilbageholder emailen og sender så mange kopier af sted til serveren, som der er modtagere til denne og påhæfte én modtager og én mønt på hver kopi. Dette ville dog skabe en del ekstra trafik mellem afsender og udgående mailserver og desuden vil modtagerne ikke kunne se, at andre har modtaget samme email.

En anden mere brugbar løsning er, at påhæfte én mønt for hver modtager på samme email og danne en overførselssignatur for hver modtager. Emailen vil så kunne sendes afsted og distribueres til samtlige modtagere af mailserveren på normal vis. Hver modtager vil kun kunne bruge netop én af mønterne, nemlig den, hvor afsender har dannet en overførselssignatur med netop den pågældende modtagers email-adresse. Proxyprogrammet skulle da blot opsnappe den éne brugbare mønt ud og slette resten. Denne løsning vil således ikke udgøre en sikkerhedsrisiko mht. økonomisk misbrug. Til gengæld vil emailens samlede størrelse være relativt stor, da der skal påhæftes én mønt for hver modtager. Med denne løsning er det altså umiddelbart muligt at understøtte brug af CC- og BCC-felterne, vel at mærke uden at ændre på emailsystemet. Dette vil dog ske på bekostning af et øget båndbreddeforbrug.

7.2.6 Stikprøvekontrol

Der foretages ikke stikprøvekontrol hos klienten. En stikprøvekontrol, hvor klienter indsender en procentdel af modtagne (gyldige) e-mønter til blacklistserveren, vil sikre at kopister bliver opdaget hurtigere, end hvis disse blot pågribes under indløsning af e-mønter (som nævnt i afsnit 6.4.2.b). Et forslag til implementering af en stikprøvekontrol går ud på, at klienter indsender enkelte mønter til blacklistserveren, som opretholder en database over disse. Dette vil til gengæld give et øget pres på denne. Dette problem kan dog løses ved i stedet at indsende mønterne til de udstedende pengeservere og lade disse holde øje med kopier. Hvis dette skal være holdbart, skal indsendelsen ske efter forudgående kryptering af transaktionslisten (som det i forvejen sker ved indløsning af mønter) og samtidig være anonym. Hvis der ikke er tale om anonym indsendelse, vil pengeserveren kunne registrere, at indsender har været ejer af den pågældende mønt, når denne modtages. Når mønten senere indløses, vil pengeserveren vide, at mønten i ét eller flere led er kommet fra indsender til indløser. Dette er umiddelbart ikke acceptabelt. Den bedste løsning på problemet vil således være at tillade anonym indsendelse af e-mønter til de udstedende pengeservere og på den måde aflaste blacklistserveren.

7.3 Udspecificering af implementeringsdetaljer

I ovenstående afsnit blev beskrevet de features, der ikke er implementeret i systemet. I dette afsnit vil blive fremhævet nogle centrale dele, der *er* implementeret.

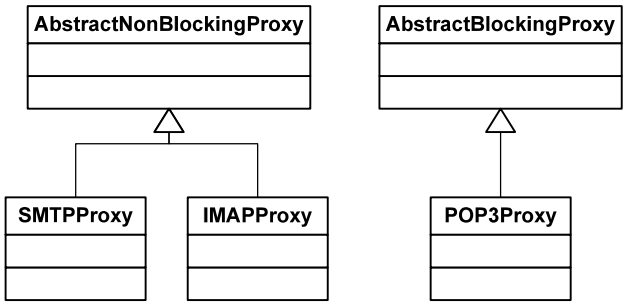
Under implementationen er det generelt sikret, at funktionalitet kun findes ét sted i koden. Dette gør fejlfinding lettere og giver bedre mulighed for at udbygge programmet. Derudover vil programmets størrelse på denne måde blive reduceret. I de følgende afsnit vil blive fremhævet dele af programmet, hvor der er lagt særligt stor vægt på strukturen.

7.3.1 Proxy

Når proxyfunktionaliteten i klienten skal implementeres, skal denne understøtte tre protokoller, nemlig SMTP, POP3 og IMAP. I alle tre tilfælde skal proxyen lade trafikken slippe igennem indtil et bestemt tidspunkt, hvorefter der skal brydes ind i protokollen. POP3 er som tidligere nævnt en forespørgsels-protokol, idet kommunikation foregår ved, at klienten sender en kommando og serveren svarer. Endvidere findes kun én forbindelse mellem klient og server. I IMAP kan både klient og server sende kommandoer og proxyen skal kunne håndtere et vilkårligt antal forbindelser. Derfor findes to grundsten i proxypakken nemlig `AbstractBlockingProxy` og `AbstractNonBlockingProxy`. Disse to klasser benytter, som navnene antyder, hhv. blokerende og ikke-blokerende netværksforbindelser.

POP3-proxyen bygges ved at nedarve fra `AbstractBlockingProxy`. I `AbstractBlockingProxy` findes funktioner til at opretholde én forbindelse til klienten, én til serveren og til at læse og skrive kommandoer til og fra de to. I selve POP3Proxy-klassen findes så funktioner til at styre proxyens reaktioner og til håndtering af kommandoer fra klienten i POP3.

IMAP-proxyen bygger på `AbstractNonBlockingMailProxy`. Denne står for at oprette forbindelser, men da der kan forekomme flere forbindelser, håndteres hver forbindelse af en separat klasse, nemlig `NonBlockingProxyConnection`. Denne klasse læser og skriver kommandoer mellem server og klient. `AbstractNonBlockingMailProxy` benytter ikke-blokerende netværksforbindelser til klient og server. Dette giver mulighed for, at denne ene tråd kan lytte til både skrive- og læseevents på flere netværksforbindelser samtidig. Dette er nødvendigt for at understøtte IMAP-protokollen. Når en kommando sendes af server eller klient, sendes kommandoen gennem proxyklassen, for at denne kan reagere det rigtige sted i protokollen. Med udgangspunkt i disse klasser kan en proxyklasse skrives uden større vanskeligheder. Både `SMTPProxy` og `IMAPProxy` nedarver funktionalitet fra `AbstractNonBlockingMailProxy`, da det her er ønskeligt med mulighed for flere netværksforbindelser. På UML-diagrammet på Figur 34 kan proxyklassernes indbyrdes forhold ses. Her er ikke vist klassernes funktioner, men i Appendiks D kan et komplet UML-klassediagram ses.



Figur 34 – Klassediagram for Proxy-pakken. Her er funktionsnavne ikke medtaget for at give overblik frem for detaljeindsigt.

Ved at lade den funktionalitet, der ikke involverer selve protokollen, blive implementeret særskilt i basisklasserne `AbstractBlockingProxy`, `AbstractNonBlockingProxy` og `NonBlockingProxyConnection`, kan disse umiddelbart benyttes til at implementere en proxy, der understøtter andre protokoller. Er der tale om en forespørgsels-protokol, kan `AbstractBlockingProxy` benyttes. Kræver protokollen flere forbindelser gennem proxyen kan `AbstractNonBlockingProxy` benyttes. Det eneste, der skal implementeres, er funktionalitet til at undersøge kommandoer og gribe ind, når det ønskes.

7.3.2 Emailbehandling

I SpamCash-systemet er det ofte nødvendigt at ændre i modtagne og afsendte emails (f.eks. i forbindelse med vedhæftning og opsnapning af e-mønter). Alle værktøjer til emailhåndtering er placeret ét sted, nemlig i klassen *Mimetools.* I SpamCash-systemet placeres mønter i en 'SpamCash'-header i emailen. Fordelen ved denne fremgangsmåde, frem for f.eks. at placere disse i en attachment, er, at det er langt nemmere at arbejde med headers end attachments. At tilføje en header vil nemlig ikke ændre på strukturen af emailen, mens det at tilføje en attachment kan få emailen (som er i MIME-format) til at ændre type. Derudover lettes arbejdet i forbindelse med understøttelse af IMAP i proxyprogrammet. Ved at placere e-mønten i headeren kan proxyprogrammet blot nøjes med at hente denne og kontrollere at e-mønten er gyldig. Er dette tilfældet, kan klienten blot få emailen at se uden ændringer, da denne ikke vil vise headere til klienten under normal brug. Et mindre problem ved brug af headere er dog, at nogle mailservere kun tillader headere af en vis længde. Derfor understøtter SpamCash-systemet, at e-mønter opdeles i et valgfrit antal headere. Dette gøres ved at en maksimal e-mønt-headerstørrelse angives i klassen `config` (denne behandles i afsnit 7.3.8). Herefter vil `Mimetools` sørge for korrekt opsplitning af e-mønter ved afsendelse samt gendannelse ved modtagelse.

7.3.3 Kryptering

I forbindelse med implementation af kryptografi i SpamCash-systemet benyttes en række biblioteker som beskrevet i afsnit 7.1.1. For at lette implementation af systemets øvrige klasser, hvor kryptografi skal benyttes, er klassen `Cryptotools` blevet implementeret. `Cryptotools` indeholder en lang række funktioner til generering af nøgler, kryptering vha. AES, beregning af hashværdier vha. SHA og kryptering, signering og blind signering vha. RSA. Med denne klasse haves *et* kryptografi-interface, der gør det til en simpel

opgave at benytte kryptografi i programmets øvrige klasser. Derudover er det let at udskifte de programpakker, der benyttes til at udføre de kryptografiske funktioner, idet det udelukkende er funktionerne i `Cryptotools`, der skal modificeres.

Det skal her bemærkes at den benyttede tredjepartssoftware ikke understøtter højere end 160 bit SHA (SHA-1). Derfor anvendes ikke SHA-256 i implementationen, men den noget svagere SHA-1. Skal systemet anvendes i praksis, vil det, for at opfylde designkriterierne, derfor være nødvendigt at implementere eller finde en provider, der understøtter SHA-256.

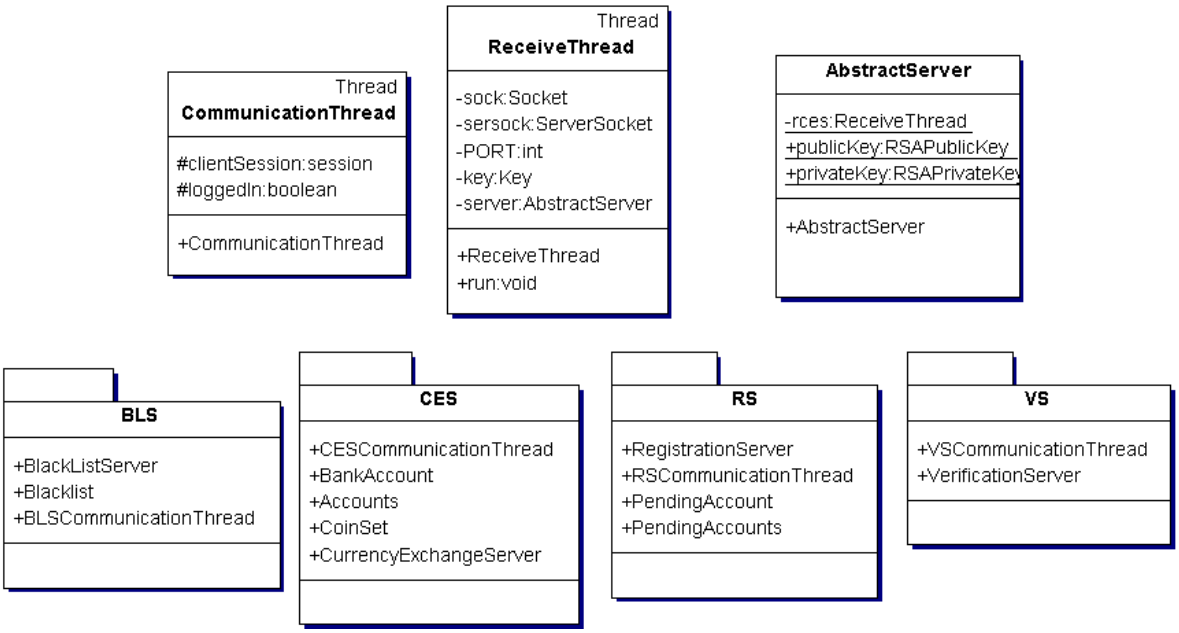
7.3.4 Session

Når kryptografi skal anvendes til at opnå sikker kommunikation over netværket i SpamCash-systemet, oprettes en kryptografisk session. Klassen `Session` sørger for håndtering af kryptografiske sessioner både hos server og klient. Så snart en forbindelse mellem server og klient er oprettet, vil begge oprette et `Session`-objekt. På klientsiden oprettes et `Session`-objekt ved at angive serverens offentlige RSA-nøgle. Herefter vil `Session` generere en sessionsnøgle og sende denne krypteret til serveren. På serversiden dannes et `Session`-objekt ved at angive serverens private nøgle. Herefter vil `Session`-objektet forsøge at modtage en krypteret sessionsnøgle og dekryptere denne. Når dette er sket, vil en `StatusPacket` blive sendt til klienten for at fortælle, at sessionen er oprettet successfuldt.

Når en session er oprettet vha. et `Session`-objekt, kan funktionerne `receive()` og `send(NetPacket)` benyttes. `Session`-objektet vil så sørge for kryptering / dekryptering samt afsendelse / modtagelse af pakker. På denne måde foregår kryptografi transparent for klient og server i systemet. I pakken `NetworkPackets` findes en lang række netværkspakker, der alle nedarver fra `NetPacket` for at lette håndtering af disse (bl.a. i `Session`). Et flag angiver ved oprettelse af et `Session`-objekt, om trafikken skal krypteres eller ej. Angives det at trafikken skal krypteres, vil alle pakker blive krypteret og sendt i en `CryptoPacket`. Ellers sendes pakken blot som den er.

7.3.5 Nedarvning i servere

I server-pakken findes én underpakke for hver af de 4 servere: BLS, CES, RS og VS. Disse har en række fælles egenskaber, der er samlet yderst i Server-pakken. Klassen `AbstractServer` indeholder den funktionalitet, som er fælles for alle serverne. Derudover findes i serverpakken klasserne `CommunicationThread` og `ReceiveThread`, der fungerer som beskrevet i afsnit 6.8.1. De enkelte serveres funktionalitet findes i pakker ved samme navn og hver enkelt server nedarver funktionalitet fra `AbstractServer` og `CommunicationThread`. På denne måde er det påkrævede arbejde ved at udvikle en ny server begrænset. Herunder ses et klassediagram for Server-pakken.



Figur 35 – Klassediagram for serverpakken hvor de tre klasser og fire pakker kan ses.

I `AbstractServer`-klassen er placeret en shutdown-metode, som benytter Java's `ShutdownHook`-klasse. Dette giver mulighed for at eksekvere kode, når serverapplikationerne lukkes ned, hvad enten dette sker korrekt eller 'på den hårde måde' (f.eks. med Ctrl+C). Dette udnyttes af de servere, som har behov for at gemme data til næste opstart. Dette gælder blacklistserveren, som skal gemme blacklisten, og CES, som bl.a. skal gemme kunders kontooplysninger og saldo.

7.3.6 GUI

Serverne i systemet er konsolbaserede, hvorfor der ikke benyttes en GUI (*Graphical User Interface*) her. I klienten findes en GUI til at interagere med brugeren af applikationen. I GUI'en findes udelukkende grafisk funktionalitet. Dette skyldes, at klientprogrammet grundlæggende er opbygget efter Model-View-Control princippet. At det grafiske dermed er fuldstændig adskilt fra resten af programmet, gør det muligt at teste denne del uden at involvere andre komponenter. Derudover skabes et bedre overblik over funktionaliteten. GUI'en er opbygget omkring to hovedvinduer, `RegistrationFrame` og `MainFrame`, der hhv. vises ved registrering og når programmet benyttes efter registrering. Herunder ses en liste med klasserne, der udgør GUI'en i SpamCash. I vinduet `MainFrame` findes et faneblad for hver hovedfunktion. Disse er indrykket og vist under `MainFrame` i nedenstående opstilling:

- **MainFrame** – Hovedvinduet med faneblade til funktionerne
    - **ConfigurationPanel** – Konfiguration af serveradresser
    - **DepositPanel** – Indløsning af e-mønter
    - **WihtdrawPanel** – Hævning af e-mønter
    - **SpamPanel** – Indkassering af 'SpamCash'
    - **WhitelistPanel** –Redigering af whitelisten
- **RegistrationFrame** – Her kan indtastes oprettelsesinformation og registrering startes
- **ErrorDialog** – Dialogboks til at vise fejl- og informationsbeskeder
- **OpenSafeDialog** – Dialogboks hvor brugeren kan intaste password og sti til nøglefil
- **FormPanel** – Indeholder en række felter der kan udfyldes
- **AmountPanel** – Indeholder felt til at indtaste og/eller justere beløb vha. +/- knapper
- **ImageButton** – En knap der benytter GIF-billeder som baggrund.
- **ProgressWindow** – Et status-vindue, der vises i nederste højre hjørne af skærmen
- **SplashScreen** – Opstartsvindue
- **Definitions** – Indeholder definitioner af komponenters farver, fonte og position.

Det skal bemærkes, at (stort set alle) klasserne i GUI'en er navngivet sådan at sidste del af navnet fortæller komponentens type. Dette giver bedre overblik, idet klasser bedre kan sammenkædes med grafisk opførsel. Vi vil ikke komme nærmere ind på detaljerne omkring alle klasserne, da de fleste har rent grafiske formål. Klassernes fremtræden kan ses i brugsvejledningen i Appendiks B, hvor det visuelle indtryk af programmet også kan bedømmes. Derudover opfordres læseren til at køre programmet (evt. også serverne) for selv at vurdere brugervenligheden og brugbarheden af den grafiske brugergrænseflade.

Klassen `Definitions` spiller en speciel rolle i GUI'en. Denne klasse definerer alle farver og fonte i systemet. Dermed kan programmets udseende ændres fuldstændig på få minutter. Udbygges denne klasse en smule, vil programmet have en decideret 'skin'-funktion, som det kendes fra mange andre programmer.

7.3.7 Log

I implementationen er indbygget en log-funktion i form af klassen `Debugger` i pakken `Utilities`. Ingen andre klasser i systemet udskriver information til skærmen. Til gengæld giver alle øvrige klasser beskeder omkring fejl og information om programmets fremskridt til `Debugger`, der derefter kan udskrive informationen, hvis det ønskes. Beskeder om fejl fra klasserne gives til `Debugger` ved at kalde nedenstående metode, hvor det kun er de første tre parametre, der er obligatoriske:

```
Debugger.debug(Class class, int level, String message, Throwable e)
```

Tallet `level` angiver beskedens detaljegrad. Er det en besked, der fortæller om store tilstandsændringer i programmet, er `level` (niveauet) sat til 1. Er det derimod meget detaljeret information, f.eks. vedrørende en løkkes fremskridt, sættes niveauet til 3. I `Debugger` findes en tabel med tupler på formen (pakke/klasse, niveau), hvor det kan angives hvilket informationsniveau, der ønskes for pakker og klasser. Her har angivelse af klassers informationsniveau højeste prioritet. Indtastes f.eks. et element med indholdet (Client, 3), vil *alle beskeder* fra *alle klasser* i pakken `Client` med niveau 1, 2 og 3 blive

udskrevet i terminalen. Findes der samtidig et element med indholdet (Control, 1), vil beskeder med niveau 2 og 3 fra klassen `Control` alligevel ikke blive udskrevet.

Ud fra referencen `class` kan stien til klassen, der har udført funktionskaldet findes. Er det f.eks. et kald fra `BlacklistServer`, vil stien være `SpamCash.Servers.BLS.BlacklistServer`. Ud fra denne sti afgøres det, hvilke pakker klassen er en del af. Dermed kan det afgøres, om beskeden skal udskrives ved at betragte niveauet. Beskeder vises ved angivelse af beskeden med foranstillet klassenavn og niveau, som det kan ses i eksemplet herunder.

```
IMAPProxy (1): Proxy is ready on port 143
```

Ved at have denne avancerede udskrift-klasse opnås flere ting:

- Fejlfinding lettes og det er ikke nødvendigt at fjerne print-statements fra koden, der ikke benyttes, når programmet er færdigt. I stedet ændres blot niveauet for klasser og pakker i `Debugger`.
- Debugger-statements, der findes i stort antal overalt i koden, fungerer som kommentarer til koden.
- Det er muligt at finjustere outputtet under test og efterfølgende tilpasse dette til det færdige program.
- Information fra `Debugger` kan umiddelbart gemmes i log-filer.

I klassen `Debugger` kan et globalt printniveau sættes, ligesom det kan vælges om 'Exception'-information skal vises. Applikationerne i systemet udskriver til skærmen efter kriterierne angivet i `Debugger`. Derudover dannes (hvis det ønskes) følgende logfiler i biblioteket `/log`, der er meget nyttige, når programmets handlinger skal undersøges efterfølgende:

- `Client.log`: Indeholder de beskeder, der vises ud fra kriterierne i tabellen i `Debugger`
- `Client_Level_1.log`: Indeholder alle beskeder med niveau 1.
- `Client_Level_2.log`: Indeholder alle beskeder med niveau 1 og 2.
- `Client_Level_3.log`: Indeholder alle beskeder med niveau 1, 2 og 3.

I ovenstående er kun medtaget log-filerne for klienten. Der genereres 4 tilsvarende filer for hver server. Et eksempel på indholdet af en `Client.log` kan ses i Appendiks E.

7.3.8 Konfiguration

I SpamCash-systemet findes en lang række parametre, der er afgørende for systemets virkemåde. Disse er samlet ét sted, nemlig i klassen `Config`. Her angives blandt andet stier til mapper, filnavne, headernavne, nøglestørrelser, pengeværdi, udløbstider, IP-adresser og portnumre. Dette giver mulighed for at ændre på systemets parametre uden større besvær. I evalueringen vil det blive fastlagt, hvilke værdier disse skal have. Skulle systemet anvendes i praksis, ville de parametre, som brugeren muligvis skal kunne justere, kunne skrives og læses fra en fil. I den nuværende implementation er det kun konfiguration af mailservernes IP-adresser (som også findes i `Config`), der gemmes på disken. Resten er fastlagt ved programstart og kan ikke ændres af klienten.

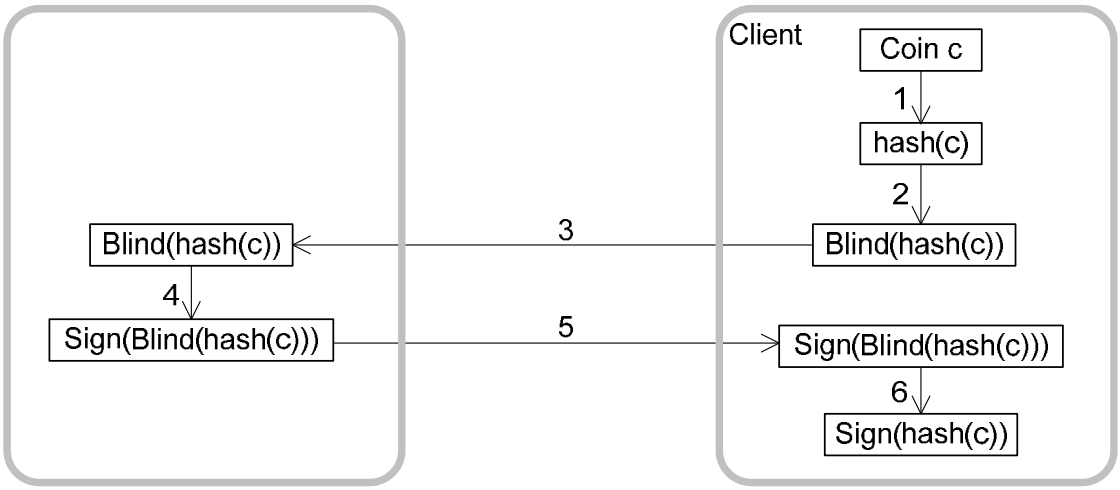
7.3.9 Blinde signaturer

I SpamCash-systemet sker udstedelse af e-mønter uden at banken kender serienumre på udstedte mønter. Der benyttes blinde signaturer, som beskrevet i designafsnit 6.8.4.b, for at opnå dette. Når udstedelsesprotokollen, beskrevet i afsnit 6.8.4.b, skal implementeres, gøres dette ved at benytte *Logi*-pakken til at udføre *blind*- og *unblind*-operationer. Hvordan dette konkret udføres i implementationen gennemgås i det følgende.

Ved udstedelse af emønter kan de mønter, der genereres af klienten gennemgå to forløb, afhængig af om serveren ønsker at kontrollere møntstrukturen eller ej. Først beskrives forløbet, hvor mønten *ikke* kontrolleres (Figur 36).

Blind signering af mønt

Klienten har genereret en e-mønt (*Coin c*) og hasher denne (trin 1). Herefter blindes hashværdien og resultatet sendes til pengeserveren (trin 2 og 3). Pengeserveren signerer og sender denne tilbage til klienten (trin 4 og 5). Klienten kan herefter unblinde denne (trin 6) og dermed råde over signaturen, som gør mønten gyldig (og dermed brugbar i systemet).

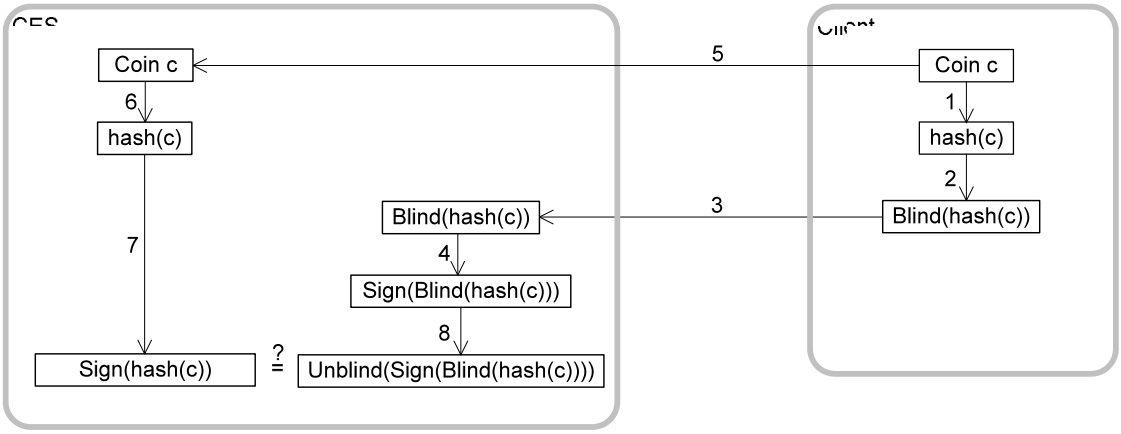


Figur 36 – Blind udstedelse af en e-mønt. Tallene på pilene angiver i hvilken rækkefølge operationer og kommunikation foregår.

Kontrol af mønt

På Figur 37 er vist forløbet hvor pengeserveren ønsker at kontrollere, at en mønt i et givet møntsæt er struktureret korrekt af klienten. Indtil et vist punkt (trin 4), foregår alt som ved signering af mønter (Figur 36). Herefter (trin 5) vil pengeserveren dog bede klienten om at få tilsendt den pågældende e-mønt, samt blindingfaktor for den korresponderende blindede hashværdi, som blev modtaget i trin 3. Pengeserveren kontrollerer efter modtagelse af e-mønten, at denne er struktureret korrekt. Er dette tilfældet hashes og signeres mønten (trin 6 og 7) og blindingfaktoren benyttes til unblinding (trin 8). Det kan nu kontrolleres, at værdien modtaget i trin 3 hører til mønten, der blev modtaget i trin 5. Dette foregår ved en simpel sammenligning af de to signerede hashværdier (som dannes

ved at udføre trin 7 og trin 8). Hvis, og kun hvis, disse er identiske, er kontrollen succesfuld.



Figur 37 – Verifikation af en blinded e-mønt. Tallene på pilene angiver i hvilken rækkefølge operationer og kommunikation foregår.

Umiddelbart ser det ud til, at trin 4 og 7 på Figur 37 kan udelades, da der udføres en identisk operation på det, som senere skal sammenlignes. Operation kan dog ikke udelades, da det er påkrævet at data er signeret, før unblind-operationen kan udføres. Dette kræves i den benyttede implementation af blinde signaturer i RSA.

7.3.10 Filer

I programmet benyttes en filstruktur til at gemme tilstande i de enkelte applikationer, og til at håndtere emails i proxydelen. I dette afsnit vil blive dannet et overblik over, hvilke filer applikationerne benytter. På installations-CD'en (se installationsvejledning i Appendiks A) findes et bibliotek for hver applikation. I hver af disse findes selve jar-filen med programmet og et bibliotek kaldet `data`. I dette bibliotek gemmer applikationerne alt relevant data. Applikationerne indeholder følgende filstruktur, hvor indrykning angiver indhold af mapper, filnavne er markeret med kursiv og mapper er markeret med fed:

- **Data**
    - *Configuration.sc* – Konfigurationsdata i klienten
    - *Safe.sc* – Pengeskabet indeholdende klientens private nøgle og objekterne Safe og pendingMails i krypteret form.
    - *Whitelist.sc* – Indeholder whitelisten i klienten
    - *Blacklist.sc* – Indeholder blacklisten. Benyttes i klient og i blacklistserver
    - *CESdata.sc* – Indeholder pengeserverens data
    - **Images** – Indeholder billeder til GUI'en
    - **Tempincoming** – Emails, der er modtaget men ikke behandlet (POP3)
    - **Incoming** – Emails, der er modtaget og behandlet (POP3)
    - **Outgoing** – Emails, der er ved at blive sendt (SMTP)
    - **Keys** – Nøgler
        - *Safe-key.key* – AES-nøgle til kryptering af filen *Safe.sc* i klienten
        - *BLS_private.key* – Blacklist- og verifikationsserverens private nøgle
        - *CES_private.key* – Pengeserverens private nøgle
        - *RS_private.key* – Registreringserverens private nøgle
    - **Certificates** – Certifikater
        - *Certificate.crt* – Klientens eget certifikat
        - *BLS_certificate.crt* – Blacklist- og verifikationsserverens certifikat
        - *CES_certificate.crt* – Pengeserverens Certifikat
        - *RS_certificate.crt* – Registreringserverens certifikat
    - **Log** – Log-filer

I ovenstående er alle filer medtaget. I den enkelte applikation findes dog kun visse filer. Kun de korrekte private nøgler findes i hver enkelt applikation. De fire øverste biblioteker (startende med **Images**) samt filerne *Configuration.sc*, *Safe-key.sc*, *Safe.sc* og *Whitelist.sc* findes udelukkende i klienten. Generelt er det kun de filer, der er nødvendige, der findes i hver applikation. Derudover kan alle mappe- og filnavne umiddelbart ændres i klassen `Config`.

8 Evaluering

Formålet med at implementere SpamCash-systemet var, at undersøge om designet kunne fungere i praksis. For at afgøre dette er det dog nødvendigt, at afgøre både *om* og *hvor godt* programmet virker. I det følgende vil derfor først blive udført en test af programmets funktionalitet, der skal vise om programmet fungerer efter hensigten. Derefter vil et afsnit diskutere og fastlægge parametre i programmet, for at dette kan fungere bedst muligt. Til sidst vil der blive målt på systemets ydeevne.

8.1 Afprøvning

8.1.1 Testmetode

Der er tale om en demoimplementation, der som nævnt er udarbejdet for at demonstrere at designet virker i praksis. Med dette som udgangspunkt vil testen blive foretaget på relativt højt niveau. Dermed menes, at der ikke vil blive udført en gennemgribende strukturel test af hver enkelt klasse. Dette skyldes hovedsageligt systemets omfang og tiden til rådighed i dette projekt (vi er nu på side 123). Til gengæld vil en grundig funktionel test blive udført. I denne vil de dele af programmet, der har betydning for sikkerheden, blive testet særligt grundigt. Dette betyder at f.eks. metoden til blacklisting af brugere i blacklistserveren *vil* gennemgå en strukturel test. Tilsvarende vil funktionen, der afgør, om en email skal modtages, blive testet særligt grundigt.

8.1.2 Funktionel test

8.1.2.a Klient

I dette og de følgende afsnit vil en systematisk funktionel test blive gennemført. I nedenstående punktopstilling angiver hovedpunkter en overordnet tilstand eller *event*. Ved hver case angives tilstanden i udgangssituationen og den handling, der skal udføres. Derudover beskrives hvilken tilstand, der skal opnås efter udførsel af handlingen, for at den pågældende *case* kan betragtes som succesfuld.

Test udført i SpamCash-klient

I det følgende udføres en test af alle funktioner i SpamCash-klienten, der kan aktiveres af brugeren.

- Opstart af program. Brugeren starter proxyprogrammet og:
    - **Case 1:** Certifikat-filen og pengeskabsfilen findes på disken. Her skal hovedvinduet vises.
    - **Case 2:** Certifikat-filen findes ikke på disken. Her skal registreringsvinduet vises.
    - **Case 3:** Pengeskabs-filen findes ikke på disken. Her skal registreringsvinduet vises.

- Registrering. Brugeren trykker på ’Register’-knappen i registreringsvinduet og:

    - **Case 4:** Klientprogrammet udfører registreringsproceduren. Her skal det kontrolleres, at programmet har gemt det modtagne certifikat på disken. Derudover skal programmet have dannet pengeskabsfilen og en nøglefil til beskyttelse af denne. De to filer til blacklisten og whitelisten skal endvidere være dannet.
    - **Case 5:** Registreringsserveren kan ikke kontaktes. Der skal vises en fejlbesked og brugeren skal kunne forsøge igen.
    - **Case 6:** Pengeserveren kan ikke kontaktes. Der skal vises en fejlbesked og brugeren skal kunne forsøge igen.
    - **Case 7:** Der er ikke penge nok til oprettelse på brugerens konto. Der skal vises en fejlbesked og brugeren skal kunne forsøge igen.

- Konfiguration af mailserveradresser. Brugeren indtaster nye adresser og trykker på 'update'-knappen.
    - **Case 8:** Adresserne gemmes i hukommelsen og på disken. Her kontrolleres det, ved at trykke på 'send og modtag email'-knappen i email-klienten, at de opdaterede adresser benyttes. Ved at genstarte programmet og foretage samme procedure kontrolleres det, at de opdaterede adresser indlæses korrekt fra disk.

- Hæv penge. Brugeren vælger et beløb i 'Withdraw'-fanebladet og trykker på 'Withdraw'-knappen og:
    - **Case 9:** Beløbet vælges lavere end én mønts værdi. Dette skal give en fejlmeddelelse til brugeren.
    - Beløbet vælges større end én mønts værdi og:
        - **Case 10:** Klienten har *ikke* dækning for beløbet på sin konto. Skal give en fejlmeddelelse til brugeren.
        - **Case 11:** Klienten *har* dækning for beløbet på sin konto. Det korrekte antal mønter skal returneres til brugeren og deres værdi debiteres på dennes konto.

- Indløs penge. Brugeren vælger et beløb i 'Deposit'-fanebladet og trykker på 'Deposit'-knappen.
    - **Case 12:** Antal mønter vælges større, end antallet af mønter klienten har til rådighed. Skal give en fejlmeddelelse til brugeren.
    - Antal mønter vælges ikke større end antallet af mønter klienten har til rådighed og:
        - **Case 13:** Ingen af mønterne har været indløst tidligere.
        - **Case 14:** En eller flere mønter har været indløst tidligere. Her skal klienten modtage besked om hvor mange mønter, der kunne indløses, og kun krediteres for disse.

- Indkasser ’SpamCash’. Brugeren markerer et antal emails i ’SpamCash’-fanebladet og trykker på ’Get coin’-knappen.
    - **Case 15:** Brugeren vælger at indkassere betaling for én email. E-mønten skal konverteres til en A-mønt og placeres i pengeskabet. En notifikationsmail skal sendes til afsender.
    - **Case 16:** Brugeren vælger at indkassere betaling for flere emails. Alle tilhørende e-mønter skal konverteres til A-mønter og placeres i pengeskabet. En notifikationsmail skal sendes til afsender.

- Opdater ’whitelist’.
    - **Case 17:** Brugeren indtaster en emailadresse under ’Whitelist’-fanebladet og trykker på ’Add’-knappen. Det kontrolleres, at emailadressen tilføjes korrekt til whitelisten.
    - **Case 18:** Brugeren markerer én eller flere emailadresser under ’Whitelist’-fanebladet og trykker på ’Remove’-knappen. Det kontrolleres at email-adresserne fjernes korrekt fra whitelisten.

- Pengestatus.
    - **Case 19:** Brugeren trykker på ’Moneystatus’-knappen. Her skal vises et vindue med status for pengebeholdningen. I dette skal stå hvor mange mønter, der er tilrådighed, hvor mange mønter der er i venteposition (betalingen er ikke udløbet hos modtager) og hvor mange mønter modtagere har beholdt siden oprettelse.

- Afslut program.
    - **Case 20:** Brugeren trykker på ’Exit’-knappen. Programmet skal gemme data (whitelist og blacklist) korrekt på disken. Det kontrolleres ved at betragte output og ved at starte programmet igen, at data er blevet gemt korrekt. Pengeskabet gemmes på disken ved hver eneste ændring. Derfor er det ikke nødvendigt at dette gemmes ved nedlukning af programmet.

- Åbne pengeskabsfil. Brugeren udfører en handling, der kræver adgang til pengeskabet. Det kontrolleres at der vises en dialogboks, hvor sti til nøglefil og password kan angives (kontrol af oplysningernes rigtighed behandles fra Case 28 til og med Case 32). Følgende handlinger kræver adgang til pengeskabet:
    - **Case 21:** Sender email.
    - Henter email og
        - **Case 22:** Ingen mails går igennem filtreringen. Her skal dialogboksen *ikke* vises.
        - **Case 23:** Én eller flere mails går igennem filtreringen
    - **Case 24:** Hæver penge.
    - **Case 25:** Indkasserer ’SpamCash’.
    - **Case 26:** Indløser penge.
    - **Case 27:** Vælger at se pengestatus.

- Åbne pengeskabsfil. Brugeren får forevist en dialogboks og det kontrolleres, at programmet reagerer efter hensigten på indtastninger i denne:
    - **Case 28:** Brugeren indtaster en sti til en fil, der ikke findes. I dette tilfælde skal pengeskabsfilen ikke indlæses.
    - **Case 29:** Brugeren indtaster en sti til en fil, der ikke er en nøglefil. I dette tilfælde skal pengeskabsfilen ikke indlæses.
    - **Case 30:** Brugeren indtaster en sti til en forkert nøglefil. I dette tilfælde skal pengeskabsfilen ikke indlæses.
    - **Case 31:** Brugeren indtaster en sti til den korrekte nøglefil, men angiver forkert password. I dette tilfælde skal pengeskabsfilen ikke indlæses.
    - **Case 32:** Brugeren indtaster en sti til den korrekte nøglefil og angiver korrekt password. I dette tilfælde skal pengeskabsfilen indlæses.

Ud over ovenstående testcases er det blevet testet, at det ikke er muligt at indtaste ulovlige værdier i den grafiske brugergrænseflade. Der er altså forsøgt indtastet forkerte eller mangelfulde værdier i alle felter, og det er efterfølgende blevet kontrolleret at en passende fejlbesked er blevet vist.

Test udført i mailklient

Test af proxyfunktionaliteten i klientapplikationen foregår ved at benytte en mailklient på en maskine, hvor klientapplikationen kører. For at udføre nedenstående tests har vi benyttet Microsoft's Outlook Express som mailklient og Alt-N's Mdaemon mailserver. Før testen er mailklient og SpamCash-klient blevet konfigureret som beskrevet i brugsvejledningen (se Appendiks B) og startet på samme maskine. Mailserveren er startet på en anden maskine. Båndbredden mellem mailserver og klientmaskine var under testen 512 Kbit, hvilket gjorde det nødvendigt at benytte emails på op til 1-2 MB for at fremprovokere timeout i emailklienten. Dermed er det også blevet testet, at proxyprogrammet kan håndtere emails af denne størrelse.

Test af emailhåndtering

For at teste om SpamCash-klienten håndterer emails korrekt ved afsendelse og modtagelse, er følgende tests blevet udført.

- Modtag email. Brugeren trykker "Hent email" i mailklienten.
    - **Case 33:** Opdatering til blacklisten hentes fra blacklistserveren.
    - **Case 34:** Opdatering hentes ikke, fordi blacklisten er opdateret.
    - **Case 35:** Afsender er på whitelisten. Emailen leveres til mailklienten uden videre kontrol af indhold eller eventuel mønt.
    - **Case 36:** Afsender er ikke på whitelist. Det kontrolleres at verifikation af e-mønten fortsætter korrekt.

I *alle* følgende ”Modtag email”-cases skal emailen afvises, såfremt afsenderadressen ikke er på whitelisten. Det skal altså kontrolleres, at denne slettes korrekt:

- **Case 37:** Emailen indeholder ingen e-mønt.
- **Case 38:** Afsenders certifikat findes i blacklisten.
- **Case 39:** Det tilladte antal ejerskift er overskredet (transaktionslisten er for lang).
- **Case 40:** Mønten er i ’pendingMails’. Den har altså været modtaget kort forinden.
- **Case 41:** Overførselssignaturen kan ikke verificeres.
- **Case 42:** Emailsignaturen kan ikke verificeres.
- Møntens integritet er brudt
  - **Case 43:** Møntens CES-signatur kan ikke verificeres.
  - Transaktionslistens integritet er brudt fordi:
    - **Case 44:** Et type 1 element ikke kan verificeres.
    - **Case 45:** Et type 2 element ikke kan verificeres.
- **Case 46:** Mønten har været ejer af mønten *n* gange før, og brugeren optræder *færre end n* gange i transaktionslisten.

- Send mail.
  - **Case 47:** Brugeren sender en email vha. sin emailklient. Her kontrolleres det, at mønten bliver vedhæftet korrekt og at overførselssignatur dannes korrekt.

Test af proxyfunktionalitet

Følgende cases tester proxyfunktionaliteten. Her er store emails (1-2 MB) blevet sendt og modtaget, når en timeout i klienten skulle fremprovokeres. Det er i alle nedenstående tilfælde kontrolleret, at proxyprogrammet viser et statusvindue, når der hentes og sendes email.

- Modtag email via POP3. Brugeren henter email med sin emailklient, der er sat op til at benytte POP3.
  - Der forekommer ingen timeout i emailklienten og
    - **Case 48:** Nul emails modtages og der opstår hverken fejl i mailklient eller proxyprogram.
    - **Case 49:** Én email modtages og der opstår hverken fejl i mailklient eller proxyprogram.
    - flere emails modtages og:
      - **Case 50:** Ingen af disse skal videregives til emailklienten og der opstår hverken fejl i mailklient eller proxyprogram.
      - **Case 51:** Én eller flere mails skal videregives til emailklienten og der opstår hverken fejl i mailklient eller proxyprogram.
  - **Case 52:** Der forekommer timeout i emailklienten, før alle emails er hentet af proxyen. Her skal det kontrolleres, at ingen emails går tabt og at klienten modtager emails, næste gang denne forsøger at hente disse.

- Send email via SMTP. Brugeren sender email med sin emailklient.
    - Én email findes i udbakken. Denne sendes og:
        - **Case 53:** Der forekommer ingen timeout i emailklienten
        - **Case 54:** Der forekommer timeout i emailklienten før proxyen har fået afsendt emailen (pga. dennes størrelse). Det kontrolleres, at proxyen sørger for, at emailen når frem alligevel.
    - Flere emails findes i udbakken og sendes og:
        - **Case 55:** Der forekommer ingen timeout i emailklienten
        - **Case 56:** Der forekommer timeout i emailklienten pga. størrelsen af emailsne, før proxyen har fået afsendt disse. Det kontrolleres, at proxyen sørger for at alle emails når frem.

- Modtag email via IMAP.
    - Klienten tilgår en ny folder vha. emailklienten og:
        - Der forekommer ikke timeout i emailklienten og:
            - **Case 57:** Ingen emails skal vises til klienten. Det kontrolleres, at emailklienten ikke får de emails at se som filtreres fra.
            - **Case 58:** Én eller flere emails kan verificeres. Det kontrolleres, at emailklienten kun får emails at se som *ikke* filtreres fra.
        - **Case 59:** Der forekommer timeout i emailklienten, før alle headere er hentet af proxyen og emailen er verificeret. Klienten skal her kunne tilgå folderen efter afsluttet filtrering på trods af fejlbeskeden.
    - Serveren opdager ny email og meddeler proxyen dette. Her vil ikke forekomme timeout i klienten, da denne ikke indblandes før *efter* filtrering:
        - **Case 60:** Ingen emails går igennem filtreringen. Her skal emailklienten ikke kontaktes.
        - **Case 61:** Én eller flere emails går igennem filtreringen. Klienten ser kun de emails, der ikke filtreres fra.

- Benytte mere end én forbindelse i IMAP.
    - **Case 62:** En af de situationer, der kræver mere end én forbindelse gennem proxyen afprøves. F.eks. kan administration af foldere påbegyndes i emailklienten. Dette kræver nemlig en ekstra forbindelse. Her skal ingen fejl opstå i emailklienten og email skal stadig kunne modtages korrekt efterfølgende.

Ved udførsel af ovenstående testcases 1 - 62 opførte klientprogrammet sig efter hensigten. Derudover blev det kontrolleret, at klientens protokolhåndtering foregik i overensstemmelse med beskrivelserne i designafsnit 6.9.1 ved at inspicere outputtet fra programmet. Uddybende tests af POP3-mailserverfunktionaliteten er ikke foretaget, men ingen fejl er dog blevet konstateret under ovenstående tests, hvorfor det er sandsynligt, at denne virker korrekt. Udover ovenstående testcases er en stresstest blevet udført vha. emailklienten. Dette indebærer, at en bruger i højt tempo trykker gentagende gange på knapperne i sin emailklient. Dette medførte, at der i IMAP-tilfældet oprettedes op til 6 forbindelser samtidig og at der i POP3 blev hentet email før forrige afhentning var

færdig. Derudover blev der sendt og modtaget samtidig, med SMTP i forbindelse med IMAP, og med SMTP i forbindelse med POP3. Alt dette blev håndteret af proxyprogrammet uden problemer.

Alle ovenstående tests blev således gennemført, uden at der blev konstateret fejlagtig opførsel. Dermed kan det forventes, at SpamCash-klienten generelt fungerer efter hensigten. I det følgende vil serverapplikationerne blive testet for at sikre, at systemet som helhed fungerer korrekt.

8.1.2.b Servere

I de følgende afsnit vil hver enkelt server blive testet. Her tages udgangspunkt i de netværkspakker, hver enkelt server kan modtage. For hver case er det beskrevet, hvordan serveren skal reagere på indholdet i modtagne netværkspakker. Dette indebærer både hvilke variable, der skal opdateres og hvilke pakker, der evt. skal sendes retur.

CES – Pengeserver

Pengeserveren kan modtage følgende pakker, der hver giver anledning til én eller flere testcases:

CES modtager en:

- LogonPacket(username, password) og
    - **Case 63:** Username er forkert. Her skal der sendes en StatusPacket(false) tilbage.
    - **Case 64:** Username er korrekt og password er forkert. Her skal der sendes en *StatusPacket(false)* tilbage.
    - **Case 65:** Username og password er korrekt. Her skal der sendes en *StatusPacket(true)* tilbage og brugernavnet skal være gemt i den pågældende kommunikationstråd.

- DepositRequestPacket(Coin[]) og
    - Logon er foregået succesfuldt og
        - Ingen af e-mønterne har tidligere været indløst og
            - **Case 66:** Alle e-mønter kan verificeres af VS. Her skal indløser krediteres for alle indløste e-mønter.
            - **Case 67:** En del af e-mønterne kan ikke verificeres af VS. Her skal indløser kun krediteres for de e-mønter, denne er retmæssig ejer af (dem der kunne verificeres).
        - En del af e-mønterne har tidligere være indløst (altså kopier) og
            - **Case 68:** Alle mønter kan verificeres af VS. Her skal indløser kun krediteres for de ikke tidligere indløste e-mønter og kopierede mønter skal sendes til BLS.
            - **Case 69:** En del af e-mønterne kan ikke verificeres af VS. Her skal indløser kun krediteres for e-mønter, som både kan verificeres af VS *og* som ikke tidligere har været indløst (altså kopier). Kopierede mønter skal sendes til BLS.
    - **Case 70:** Logon er ikke foregået succesfuldt. Her skal forbindelsen afbrydes til klienten.

- CurrencyExchangeRequestPacket(BlindedCoin[][]) og
    - Logon er foregået succesfuldt og
        - **Case 71:** Der er lige mange elementer i hvert sæt af blindede mønter. Her udvælges tilfældige indeks for de e-mønter, der ikke ønskes blindingfaktorer for. Disse skal være gemt i vektoren *CoinSets* sammen med brugernavnet, så det senere kan kontrolleres, at klienten returnerer de korrekte mønter.
        - **Case 72:** Der er ikke lige mange elementer i hvert sæt af blindede mønter. Her skal forbindelsen afbrydes til klienten
    - **Case 73:** Logon er ikke foregået succesfuldt. Her skal forbindelsen afbrydes til klienten.

- CurrencyBlindingFactorPacket(BlindingFactors[][], Coin[][], Certificate) og
    - Logon er foregået succesfuldt og
        - Certifikatet er gyldigt og
            - De korrekte mønter er fremsendt. De udvalgte e-mønter unblindes og verificeres og
                - **Case 74:** Verifikationen var succesfuld. Det kontrolleres, at den ene e-mønt i hvert sæt, som stadig er blinded, signeres korrekt og at de signerede mønters samlede værdi er debiteret på klientens konto.
                - **Case 75:** Verifikationen var ikke succesfuld. Her skal forbindelsen til klienten afbrydes.
            - **Case 76:** Forkerte mønter er fremsendt. Her skal forbindelsen til klienten afbrydes.
        - **Case 77:** Certifikatet er ikke gyldigt. Her skal forbindelsen til klienten afbrydes.
    - **Case 78:** Logon er ikke foregået succesfuldt. Her skal forbindelsen til klienten afbrydes.

- ActivationRequestPacket(activationNumber) og
    - Logon er foregået succesfuldt og
        - Det medsendte aktiveringsnummer har endnu ikke været brugt (betalt for) og
            - **Case 79:** Klientens certifikat kan ikke hentes fra RS. Her skal sendes en *StatusPacket(false)* til klienten og dennes konto skal *ikke* debiteres for oprettelsesbeløbet.
            - **Case 80:** Klientens certifikat kan hentes succesfuldt fra RS. Her skal klientens konto debiteres for oprettelsesbeløbet og der skal sendes en *StatusPacket(true)* til klienten.
        - **Case 81:** Det medsendte aktiveringsnummer har allerede været brugt (og betalt). Her skal blot returneres en *StatusPacket(true)* til klienten, der indikerer, at RS har fået besked på at åbne kontoen.
    - **Case 82:** Logon er ikke foregået succesfuldt. Her skal forbindelsen til klienten afbrydes.

BLS – Blacklistserver

Blacklistserveren kan modtage følgende pakker, der hver giver anledning til én eller flere testcases. Det skal her bemærkes at case 86-94 udgør en strukturel test af metoden *reportCoins* i blacklistserveren.

BLS modtager en:

- GetBlacklistPacket(*i*) og
    - **Case 83:** *i* er større end eller lig 0 og mindre end længden på blacklisten. Det kontrolleres, at blacklisten fra og med indeks *i* sendes tilbage i en BlacklistPacket(Blacklist). Hvis *i* er 0, skal hele blacklisten sendes.
    - **Case 84:** *i* er mindre end 0. Det kontrolleres, at intet sendes tilbage, og at den pågældende kommunikationstråd lukker forbindelsen til klienten.
    - **Case 85:** *i* er større end eller lig længden på blacklisten. Det kontrolleres, at intet sendes tilbage, og at den pågældende kommunikationstråd lukker forbindelsen til klienten.

- ReportCoinsPacket(Coin a, Coin b) og
    - **Case 86:** E-mønterne *a* og *b* har ikke samme serienummer. Ingen skal blacklistes.
    - **Case 87:** E-mønterne er identiske. Ingen skal blacklistes.
    - **Case 88:** Transaktionslisten i mønt *a* kan ikke dekrypteres med blacklistserverens private nøgle. Ingen skal blacklistes.
    - **Case 89:** Transaktionslisten i mønt *b* kan ikke dekrypteres med blacklistserverens private nøgle. Ingen skal blacklistes.
    - **Case 90:** Integriteten af transaktionslisten i mønt *a* kan ikke verificeres. Ingen skal blacklistes.
    - **Case 91:** Integriteten af transaktionslisten i mønt *b* kan ikke verificeres. Ingen skal blacklistes.
    - **Case 92 & 93:** Transaktionslisten i mønt *a* er *en del af* transaktionslisten i mønt *b*. Her skal det kontrolleres, at certifikatet i sidste element i transaktionslisten i mønt *a* tilføjes blacklisten.
    - **Case 94:** Transaktionslisterne har forskellige elementer ved indeks *i*. Her skal certifikatet i indeks *i*-1 tilføjes blacklisten.

I ovenstående er case 92 og 93 slået sammen i ét punkt, men udgør to cases. Dette skyldes, at det skal testes at blacklistserveren blacklister den korrekte i ovenstående tilfælde, men også hvis e-mønterne *a* og *b* ombyttes.

RS – Registreringsserver

Registreringsserveren kan modtage følgende pakker, der hver giver anledning til én eller flere testcases:

RS modtager en:

- RegistrationRequestPacket(certificateSigningRequest) og det kontrolleres at:
    - **Case 95:** Der oprettes en *pendingAccount* med det korrekte indhold og serveren svarer med en *RegistrationInfoPacket(activationNumber + 1)*.

- ActivationPacket(Sign(activationNumber)) og:
    - **Case 96:** Det kan ikke verificeres, at signaturen er foretaget af CES. Her skal RS ikke foretage sig mere og forbindelsen skal afsluttes.
    - **Case 97:** Det kan verificeres, at signaturen er foretaget af CES. Her skal det kontrolleres, at et signeret certifikat dannes ud fra den *pendingAccount*, der indeholder det pågældende activationNumber.

- RegistrationInfoPacket(activationNumber) og:
    - **Case 98:** Certifikatet findes ikke på disken. Her skal det kontrolleres, at serveren svarer med en *StatusPacket(false)*.
    - **Case 99:** Certifikatet findes på disken. Her skal det kontrolleres, at serveren returnerer certifikatet i en *ActivationCompletedPacket(Certificate)*.

VS – Verifikationsserver

En mønt og et certifikat sendes til serveren, og denne giver positivt eller negativt svar afhængigt af, om det kan verificeres, at mønten tilhører det medsendte certifikat eller ej. Serveren skal give negativt svar i alle cases herunder:

- **Case 100:** Transaktionslisten kan ikke dekrypteres
- **Case 101:** Det maksimale antal tilladte ejerskift er overskredet
- Transaktionslistens integritet er brudt idet:
    - **Case 102:** Et type 1 element ikke kan verificeres
    - **Case 103:** Et type 2 element ikke kan verificeres
- **Case 104:** Det medsendte certifikat stemmer ikke overens med det, som befinder sig sidst i transaktionslisten.

Serveren skal give et positivt svar hvis, og kun hvis:

- **Case 105:** Alle verifikationer er succesfulde og det medsendte certifikat stemmer overens med det, som befinder sig sidst i transaktionslisten.

Alle ovenstående testcases for serverne blev gennemført succesfuldt. Dermed er både klient og servere blevet testet med positivt resultat og det kan derfor forventes, at hele systemet virker efter hensigten.

8.1.3 Test af sikkerhed

Selv om en funktionel test af applikationerne er blevet gennemført, kan der stadig findes sikkerhedshuller i systemet. Dette kunne være forårsaget af problemer i selve designet af protokollerne eller datastrukturen, som benyttes i mønten. Om dette er tilfældet kan undersøges ved systematisk at angribe protokollerne i systemet med replay-, injektions- og man-in-the-middle angreb. Tilsvarende kan datastrukturerne testes ved systematisk at forsøge at bryde disse. Da tiden i dette projekt ikke tillader en gennemgribende praktisk test, er dette ikke blevet udført. Til gengæld er protokolbeskrivelserne i designafsnittet blevet gennemgået systematisk for at sikre, at ingen angreb er mulige. Det kan derfor sandsynliggøres, at der ikke findes sådanne fejl.

Af andre potentielle sikkerhedshuller, kan nævnes dem, som skyldes implementeringsfejl. Når et design skal implementeres, er det ikke altid muligt, at følge designet til punkt og prikke. Så snart det, af implementeringstekniske årsager, er nødvendigt at afvige, er det vigtigt at være særligt opmærksom. Et konkret eksempel fra SpamCash-systemet er f.eks. ved udstedelse af e-penge. Her er det ikke muligt at blinde en hel mønt, som det beskrives under design (afsnit 6.8.4.b). I stedet blindes en hashværdi af denne som beskrevet under implementering (afnit 7.3.9). Netop dette tilfælde har ikke betydning for systemets sikkerhed, men det er alligevel vigtigt at være opmærksom. Ved inspektion af kildekoden er det blevet kontrolleret, at implementationen nøje følger designet de steder, som har betydning for systemets sikkerhed. Vi er på dette tidspunkt *ikke* bekendt med, at der skulle eksistere sikkerhedshuller af nogen art i systemet.

8.2 Justering af systemet

Nu er et anti-spamsystem blevet designet, implementeret og testet, og mange systemparametre er blevet indført. Der er blevet argumenteret, for at systemet er sikret mod misbrug. Detaljer i designet er blevet diskuteret og fastlagt. Formålet med dette relativt komplekse system er dog ganske simpelt: At forhindre spammail. Derfor vil systemets parametre nu blive diskuteret og fastlagt, sådan at systemet fungerer bedst muligt. Dette både med henblik på at sikre betaling for spam, men også for at systemet er brugervenligt og brugbart i praksis.

8.2.1 Serverparametre

Det skal fastlægges, hvor ofte klienterne skal hente en opdateret blackliste fra blacklistserveren. Dette skal gøres hyppigt nok til at kunne afvise emails, men ikke så ofte at blacklistserveren bliver for presset. Derfor henter klienterne blacklisten én gang når der hentes email, *med mindre* dette blev gjort for mindre end 20 minutter siden. Dette begrænser trafikken fra blacklistserver til klienter, idet denne i reglen vil blive kontaktet langt sjældnere end ved hver enkelt modtagen email i systemet. Dette er for øvrigt forudsætningen for, at blacklistserveren kræver mindre trafik end en traditionel certifikatautoritet, som det blev beskrevet i afsnit 6.2.1.b. Antaget brugere modtager f.eks. tre emails ad gangen i gennemsnit vil trafikken være reduceret med en faktor 3. Derudover vil tidsbegrænsningen reducere trafikken yderligere og sikre, at klienter der trykker ”send/modtag” gentagende gange, ikke belaster blacklistserveren unødigt.

I pengeserveren skal det fastsættes, hvor mange mønter der skal være i et 'sæt' ved udstedelse af e-penge (se afsnit 6.8.4). Dette sættes til 2 og dermed er sandsynligheden $1:2^n$ for at snyde pengeserveren for *n* mønter. I princippet kan klienten vælge et hvilket som helst beløb i emønten i forbindelse med svindel. Pengeserveren kan dog få indbygget et globalt loft for værdien af mønter, der accepteres ved modtagelse (f.eks. 5 kr). Hermed vil en klient, der hæver penge, maksimalt kunne snyde pengeserveren for et beløb inden for dette loft multipliceret med *n*. Da chancen for at dette sker er $1:2^n$ og da straffen for svindel kan sættes passende hård, antages det at ingen vil forsøge at snyde her.

8.2.2 E-penge parametre

Der er en lang række parametre vedrørende de elektroniske penge, der skal fastsættes. Parametrene er i høj grad afhængige, og det kan derfor være svært at fastsætte dem hensigtsmæssigt. Derfor tages udgangspunkt i møntens værdi, der må siges at være den mest centrale parameter. I dag koster det omkring 0,0006 kr. at sende en email i strøm og internetabonnement [36]. At øge dette beløb til blot 0,01 kr. vil formentlig fjerne incitamentet til at sende spam med henblik på at tjene penge, altså kommercielle spammails [36]. Da det er muligt at sende en begrænset mængde spammails i systemet ved pengekopiering, skal møntværdien sammen med oprettelsesgebyret sikre, at det ikke kan betale sig at sende spam på denne måde. Sendes spammail med kopierede mønter, vil modtager ikke få betaling, men antallet af emails vil være begrænset og afsender vil *altid* betale for de afsendte emails (i tilfælde af kopiering bliver dette i form af oprettelsesgebyret). Derfor startes parametervalget med at sætte møntværdien til 0.05 kr. Denne er af begrænset størrelse og vores eksperimenter viser, at denne passer godt ind i systemet med de resterende parametre. Derudover er værdien tilstrækkelig lille til, at brugerne kan finjustere prisen for adgang til deres indbakke senere.

Mønters gyldighedsperiode

Mønterne skal have en gyldighedsperiode, der skal fastsættes ud fra pengeserverens offentlige nøgle, der benyttes til signatur af mønter. Mønter skal kunne udskiftes hurtigere end nøglen kræves udskiftet. I afsnit 6.7.3 blev nøglestørrelsen for serverne sat til 2048 bit. Da denne antages at være sikker de næste 10 år, behøver mønterne i princippet ikke udskiftes, før denne tid er overskredet. For at give mulighed for implementeringsopdateringer og lignende, vælges dog en gyldighedsperiode på 1 år.

Mønters udløbstid: Opkrævningsperiode

Hvor længe det skal være muligt, at opkræve betaling efter modtagelse af en email, skal også fastsættes. Dette bør svare nogenlunde til den tid email normalt ligger i indbakken og sættes derfor til 10 dage.

Mønters udløbstid: Overlapperiode

'Overlapperioden' er det tidsrum, der går fra en B-mønt hos en modtager ikke kan opkræves længere og til at den korresponderende C-mønt omdannes til en A-mønt hos afsender. Når der opkræves betaling for en email, vil en notifikationsmail sendes til afsender. Denne kan i princippet sendes sekundet før opkrævningsperiodens udløb. Derfor haves altså en overlapperiode, der sikrer at afsender ikke benytter mønten igen, efter at modtager har opkrævet mønten. Periodens længde bør derfor være større end den

tid, det tager for notifikationsmailen at nå frem. Idet det antages, at brugere almindeligvis vil checke email mindst hver tredje dag, sættes overlapperioden til 3 dage.

Antal tilladte ejerskift

Hvor mange gange en mønt kan opkræves af en modtager, før denne skal indløses, er afgørende for størrelsen på mønterne. Transaktionslisten er den del af mønten, der har størst betydning for møntens størrelse. Vi har valgt, at sætte det tilladte antal ejerskift (TE) til 10. Dermed begrænses belastningen på systemets servere og det sikres, at der ikke går alt for længe, før mønter indløses.

I implementationen benyttes X.509 certifikater. Klientcertifikaterne fylder med 1024 bits nøgler og 2048 bits nøglestørrelse i rodcertifikatet 647 bytes. Dette er undersøgt under implementationen. Hermed fylder en mønt ved udstedelse og efter 10 ejerskift som angivet i tabellen herunder (dette er kontrolleret i praksis):

Størrelse af mønters indhold	Ved udstedelse	Efter 10 ejerskift
Konstanter (værdi, udløbstid, …)	40 bytes	40 bytes
CES-Signatur	256 bytes	256 bytes
Transaktionsliste	Element 1: 256 bytes Element 2: 903 bytes Total: 1159 bytes	10*1159 = 11590 bytes
Total	1455 bytes	11886 bytes ≈ 12 KB

Tabel 4 – Størrelsen af en mønts indhold.

Med TE sat til 10 vil møntens størrelse altså maksimalt blive 12 KB før indløsning. Dette er en væsentlig båndbreddetyv, da langt de fleste emails fylder væsentlig mindre. At hver anden email i dag er spam betyder, at spam koster enormt meget båndbredde og der spildes dermed store summer på at sende disse rundt i systemet. Indføres SpamCash-systemet ser det dog ikke ud til, at dette båndbreddeforbrug reduceres væsentligt. Størrelsen af mønterne kan dog reduceres på flere måder. For det første kan klienternes nøglestørrelse reduceres. Sikkerheden vil dermed forringes, men da det er små beløb, der skal beskyttes, kan dette forsvares. Reduceres nøglestørrelsen f.eks. til 512 bit vil certifikatstørrelsen blive 516 bytes og mønten efter 10 anvendelser fylde 10,5 KB. Det vil altså være med begrænset gevinst at gå på kompromis med sikkerhedsniveauet på denne måde. Man kunne også forsøge at anvende kompression på mønterne, inden disse sendes. Herved skal klienterne dog bruge ekstra processorkraft til gengæld for besparelsen i båndbreddeforbruget. Da klientmaskiner i dag har rigelig processorkraft, virker dette umiddelbart som en oplagt optimeringsmulighed. Møntstørrelsen vil dog kun kunne reduceres med cirka 10-15 % (da kompression af kryptografiske nøgler ikke er effektiv). Møntstørrelsen er altså under alle omstændigheder et problem, der ikke er til at overse, idet der i gennemsnit stadig vil skulle sendes 4-5 KB ekstra i alle emails. Det må siges at være et af de største problemer ved systemet og en løsning ligger ikke lige for. At systemet skal være skalerbart krævede, at vi indførte den alternative certifikatmodel beskrevet i afsnit 6.2.1.b. Prisen blev altså en stor mønt, da alle certifikater skal medsendes. Det kan overvejes, om lavt båndbreddeforbrug er at foretrække frem for skalerbarhed.

Oprettelsesbeløb

Oprettelse i systemet skal koste penge for at sikre betaling for email. Når en bruger opretter sig i systemet, får denne et antal e-mønter til brug i systemet. En spammer, der betaler for at være med i systemet, og herefter kopierer mønter og misbruger disse, skal straffes økonomisk. I forbindelse med oprettelsen får brugeren et vist antal elektroniske mønter. Brugeren får *ikke* elektroniske mønter for hele oprettelsesbeløbet, da dette ville kunne misbruges. En spammer kunne nemlig derved kopiere sine e-mønter, indløse originalerne og på den måde få sit udlæg refunderet. Herefter kunne denne spamme løs med kopierne uden beregning. Spammeren ville naturligvis blive blacklistet før eller siden, men ville på dette tidspunkt have opnået sit mål: At sende spam med forsvindende økonomiske konsekvenser. Derfor opkræves et gebyr ved oprettelse, som ikke kan refunderes:

oprettelsesbeløb = mønters værdi + oprettelsesgebyr

Oprettelsesgebyret fastsættes i næste afsnit ud fra økonomiske betragtninger. Værdien af de mønter, der gives til brugerne under oprettelse, kan dog godt vælges her. Den enkelte mønts værdi, udløbs- og overlapsperiode bestemmer sammen med de udstedte mønters værdi, 'afsendelseshastigheden'. Altså hvor mange emails (der ikke er spam), der kan sendes pr. dag uden at løbe tør for mønter. Antages det, at de fleste ikke har brug for at sende mere end 10 emails om dagen, kan mønternes værdi sættes til:

$$10 \cdot (udløbsperiode + overlapsperiode) \cdot møntpris = 10\frac{email}{dag} \cdot (10dage + 3dage) \cdot 0.05\frac{kr.}{email} = 6{,}5kr.$$

Heraf kan f.eks. beregnes at en storbruger (måske et mellemstort firma), der afsender 1000 emails dagligt, ikke behøver at have store beløb bundet, men blot 650 kr.

8.2.2.a Oprettelsesgebyr og stikprøvekontrol

For at kunne fastlægge oprettelsesgebyret og lave fornuftig stikprøvekontrol, er det nødvendigt at estimere, hvor meget en spammer kan tjene på spammails. For at systemet skal fungere efter hensigten, skal følgende to kriterier være opfyldt:

- Det skal give betydeligt underskud at sende spam med henblik på fortjeneste.
- Det skal være dyrere at sende med kopierede mønter end på 'lovlig' vis.

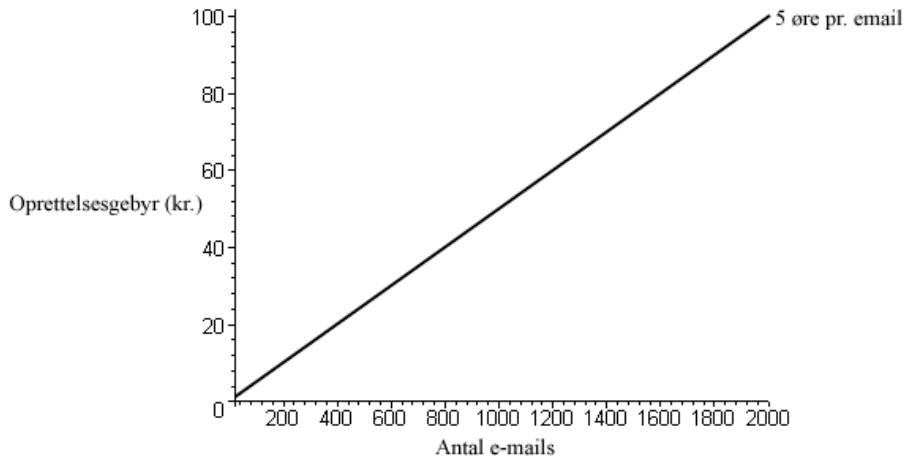
Ifølge *National Technology Readiness Survey* i USA udgivet februar 2005, hvor tilfældigt udvalgte blev udspurgt om deres emailbrugsmønster, modtog gennemsnitspersonen 18,5 spammails pr. dag i 2004 [49]. Den samme undersøgelse viser, at 4 % af personerne i undersøgelsen har købt mindst én vare, som de har set annonceret i disse spammails i samme periode. Antages det, at personerne i gennemsnit har købt hele 3 varer fra kommercielle spammails, kan profitten beregnes, idet profitten for gennemsnitsvaren sættes (højt) til 1000 kr. Fortjenesten pr. udsendt spammail bliver herved:

$$\frac{Antalkøb \cdot\ fortjeneste}{Antalmails} = \frac{Antalpersoner \cdot 0.04 \cdot 3 \cdot 1000}{Antalpersoner \cdot 18.5 \cdot 365} = \frac{0.04 \cdot 3 \cdot 1000}{18.5 \cdot 365} = 0.00096 \frac{kr}{email}$$

Dette tal virker fornuftigt, da omkostningerne ved at sende email ligger lavere, nemlig på 0,0006 kr. (se begyndelsen af afsnit 8.2.2). Det var altså profitabelt at sende kommerciel spammail i 2004. Det skal her bemærkes, at de spammails, der indgik i beregningerne, blot er dem, der er nået frem til brugeren. Udgiften vil muligvis være større for spammeren, dog stadig (åbenbart) ikke stor nok til at afskrække denne.

I SpamCash-systemet findes midlerne til at afskrække enhver kommerciel spammer. Da det koster 0,05 kr. at sende en spammeddelelse i systemet, er det ikke svært at se, at det meget hurtigt vil blive en underskudsforretning at sende kommerciel spam. Vælger spammeren at kopiere mønter og sende med kopier i stedet, skal dette heller ikke kunne betale sig. Sættes oprettelsesgebyret til blot 20 kr. og sikres det, at der ikke kan sendes mere end 2000 emails i gennemsnit før blacklisting finder sted, vil prisen pr. email være cirka 10 gange større end indtægten (nemlig 0,01 kr). Hermed vil det første kriterium være opfyldt. At sikre, at der ikke kan sendes mere end 2000 emails, gøres ved at justere klienternes *stikprøvefrekvens*, altså hvor stor en brøkdel af mønterne, der sendes til blacklistserveren til stikprøvekontrol. Jo højere *stikprøvefrekvens* jo hurtigere kan kopister blacklistes, dog på bekostning af en øget belastning i blacklistserveren.

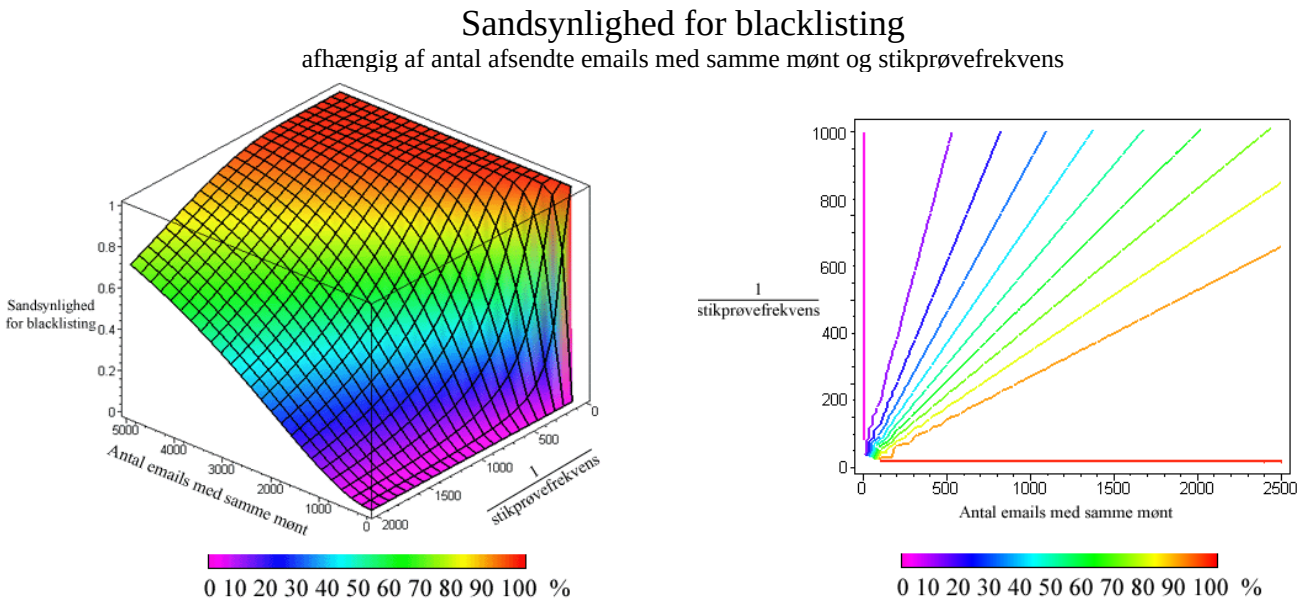
Det andet kriterium: at det skal være dyrere at sende med kopierede mønter end med legitime - er sværere at opfylde. I ovennævnte eksempel blev prisen pr. email (med kopieret mønt) 0,01 kr. Prisen skal dog være minimum 0,05 kr. For at undersøge hvilken kombination af oprettelsesgebyr og stikprøvefrekvens, der så bør vælges, kan nedenstående figur betragtes:



Figur 38 – Oprettelsesgebyret som funktion af antal emails med møntværdien sat til 5 øre.

På figuren ses oprettelsesgebyret som funktion af antallet af emails, der kan sendes med kopierede mønter før blacklisting (idet en møntværdi på 0.05 kr opretholdes). Ud fra denne graf skal altså vælges et antal emails, der er så stort som muligt, for at begrænse blacklistserverens arbejde. Samtidig må oprettelsesgebyret dog ikke være så stort, at

ingen vil benytte systemet. At skulle betale penge for at blive fri for spam kan folk acceptere, men hvor meget vil de betale? Det kan være svært at afgøre. Vi mener at de fleste vil gå med til at betale 50 kr. Derfor tages udgangspunkt i dette oprettelsesgebyr. Hermed angiver Figur 38, at en møntkopist skal være blacklistet, inden denne har sendt 1000 emails. Dette kan, da der er tale om en stikprøvekontrol, ikke garanteres, men sandsynliggøres ved at vælge den rette stikprøvefrekvens. På Figur 39 ses sandsynligheden for at blive blacklistet som funktion af stikprøvefrekvensen og antallet af afsendte emails med samme kopierede mønt. Figuren er konstrueret i Maple, og koden til dette kan ses i Appendiks F.



Figur 39 – Sandsynligheden for at stikprøvekontrol medfører blacklisting af en møntkopist. Dette beregnet ud fra hvor ofte mønter indsendes til blacklistserveren og hvor mange emails kopisten har sendt med samme mønt.

Til venstre kan ses fladen i 3D, hvor farverne angiver sandsynligheden for blacklisting. Til højre ses todimensionale kurver, der viser det samme. Her er medtaget en kurve (et snit i fladen) for 0, 10, 20, … 100 % sandsynlighederne. Indlægges nu et vandret snit i figuren til højre ved en frekvens på 1/400, kan det aflæses, at når en spammer har sendt 500 emails med samme mønt, er der 40 % chance for at blacklistserveren har opdaget dette via stikprøvekontrollen. Ved 1600 emails er sandsynligheden over 90 %.

Nu skal stikprøvefrekvensen vælges, så en spammer blacklistes efter 1000 emails. Hvis det blot sikres, at en spammer i *gennemsnit* kan sende 1000 emails før denne blacklistes, vil det koste det samme, at sende med kopierede mønter, som på legitim vis. Da spammeren også kan blacklistes ved at pengeserveren opdager kopiering, hvilket vil ske hvis blot to af modtagerne indløser mønten, vil spammeren blacklistes endnu hurtigere. Hermed vil det være dyrere, måske væsentligt dyrere at sende med kopier. Vi kan nu aflæse på grafen til højre, at en stikprøvefrekvens på 1/600 vil give en sandsynlighed på

50 % for at en kopist er blacklistet, når denne har sendt 1000 emails. Dermed skal klienterne altså indsende hver 600'ne mønt til blacklistserveren som stikprøvekontrol.

Det er muligt at en kopist standses længe før denne har afsendt 1000 emails. Omvendt vil en heldig spammer i princippet kunne sende mange tusinde emails, før denne opdages. Det vigtige er dog, at det *gennemsnitligt* ikke kan betale sig at benytte kopierede mønter. At enkelte spammere kan være heldige vil ikke kunne vælte systemet.
At tjene penge på at sende spam i systemet vil således svare til at være heldig i Lotto eller Tips.

8.3 Ydeevne

I afsnit 8.1 blev det implementerede system afprøvet og det blev vist, *at* programmet virker. Herefter blev parametrene i systemet justeret. I dette afsnit vil det blive undersøgt, *hvor godt* systemet virker med de valgte parametre. Systemets ydeevne skal altså vurderes. Der vil i dette afsnit blive opstillet et estimat for, hvor mange brugere systemet kan håndtere. Derudover vil det blive vurderet, hvordan en bruger vil opleve anvendelsen af systemet.

Serverne i systemet sætter grænsen for, hvor mange brugere systemet kan håndtere. Penge- og verifikationsservere kan indgå i relativt stort antal i systemet. Derfor er deres ydeevne ikke kritisk for systemets skalerbarhed, idet der blot kan indføres flere af disse, når dette bliver nødvendigt. Registreringsserverens ydeevne sætter udelukkende en grænse for hastigheden, hvormed der kan optages nye brugere. Registreringsserverens ydeevne vil således ikke have stor betydning for systemets skalerbarhed, så længe denne kan følge med registreringsforespørgslerne. Der kan ikke eksistere mere end én blacklistserver i systemet (en eventuelt distribueret udgave vil dog naturligvis udgøres af flere maskiner med en fælles blacklist). Derfor har det stor betydning for systemet, hvor mange brugere blacklistserveren kan håndtere. Dette også fordi systemets sikkerhed afhænger af den service, som blacklistserveren yder.

I det følgende vil systemets ydeevne blive vurderet med udgangspunkt i klienten. Her vil servernes ydeevne også blive behandlet. Herefter vil blacklistserverens ydeevne blive vurderet i et efterfølgende afsnit.

8.3.1 Klient

Brugerens oplevelse ved brug af systemet afhænger i høj grad af klientens opførsel i forskellige situationer. Derfor vil de følgende afsnit tage udgangspunkt i klientens funktioner. Først vil afsendelse og modtagelse af email blive afprøvet. Derefter vil oprettelse i systemet og håndtering af mønter blive vurderet. De funktioner, der ikke involverer nogen server (konfiguration af server-adresser og opdatering af whitelist), afprøves ikke, idet de kan udføres på meget kort tid og dermed ikke vil opleves som generende for brugeren.

8.3.1.a Emailhåndtering

SpamCash-systemet bygger på emailsystemet. Derfor er det vigtigt, at brugen af SpamCash ikke indfører begrænsninger eller væsentlige forsinkelser i brugen af emailsystemet. Derfor er der blevet udført tests af afsendelse og modtagelse af email gennem SpamCash-klienten. Den samme test blev udført ved at sende og modtage direkte til/fra mailserveren for at kunne opdage en evt. forsinkelse. I testen benyttes udelukkende beskeder med gyldig betaling (til sidst beskrives hvorfor). Testen blev udført med en mailserver i Ishøj med en internetforbindelse på 1Mbit / 512Kbit. Klientmaskinen befandt sig på Vesterbro med en internetforbindelse på 4Mbit / 512Kbit. Situationen afspejler altså en typisk situation med nogen afstand mellem server og klient. I testen benyttes to sæt beskeder. Det ene sæt indeholder 20 små beskeder (1 KB pr. styk). Med denne størrelse beskeder vil båndbredde ikke have så stor betydning, mens behandlingstid har større betydning. Det andet sæt beskeder indeholder to store beskeder (1000 KB pr. styk). Her vil den tilgængelige båndbredde have betydning. I skemaet nedenfor ses testresultaterne.

Sekunder pr. Besked	Afsendelse (SMTP)		Modtagelse (POP3)		Modtagelse (IMAP)	
	20 emails	2 emails	20 emails	2 emails	20 emails	2 emails
Direkte	0,34	32,02	0,25	32,6	0,2	0,6
Gennem Proxy	0,82	33,40	0,79	38,8	1,69	1,9

Tabel 5 – Tidsmålinger for afsendelse og modtagelse af email.

Ved afsendelse forårsager proxyen ikke store forsinkelser. Det tager under ét sekund at sende en besked. Det må antages, at brugeren ikke vil opleve dette som generende. Ved afsendelse af store beskeder er det primært båndbredden, der sætter grænsen for afsendelseshastigheden, hvorfor der stort set ikke er nogen forskel her.

Ved modtagelse vha. POP3 ligger tiderne meget tæt op ad SMTP-målingerne. Dette skyldes sandsynligvis, at protokollerne begge er meget simple. Igen er det behandlingstiden i proxyen, der giver en smule forsinkelse, hvilket især kan ses på målingerne med 20 beskeder. Her er der igen ikke tale om forsinkelser, der vil opleves som generende. Hentes dog over 50 beskeder ad gangen, vil de fleste mailklienter give timeout. For de fleste brugere vil dette dog forekomme meget sjældent og derfor vil de færreste opleve forstyrrelser.

Ved modtagelse vha. IMAP hentes ikke hele beskeder men kun headere. Dette afspejles i testresultaterne, idet tiderne er i samme størrelsesorden for både store og små beskeder. I POP3 var behandlingstiden i proxyen ca. 0,5 sek. Derfor virker det umiddelbart underligt, at behandlingstiden er omkring 1,4 sek. i IMAP. Dette kan dog forklares med anvendelsen af en kø i POP3. I POP3 kan én besked behandles, mens den næste hentes. I IMAP benyttes ikke en kø i implementationen. Dette betyder, at når en besked eller header er hentet, så skal denne behandles. Først derefter kan den næste hentes og behandles. En tid der nærmer sig 2 sekunder pr. besked, vil de fleste brugere nok ikke finde acceptabel. Derfor bør der alligevel indføres en kø i IMAP proxyen, hvilket kan gøres umiddelbart i implementationen. Gøres dette, kan det forventes, at IMAP-tiderne vil ligge som i søjlen med 20 beskeder under POP3.

Ud fra ovenstående kan det konkluderes, at proxyen ikke introducerer væsentlige forsinkelser og at systemet vil kunne bruges til afsendelse og modtagelse uden at brugeren vil opleve det som generende. Til sidst skal det bemærkes, at beskeder, der ikke indeholder gyldig betaling, vil blive behandlet og afvist væsentlig hurtigere end beskederne i det ovenstående. Var sådanne beskeder blevet benyttet i testen, ville proxyens ydeevne altså blot fremstå endnu bedre.

8.3.1.b Oprettelse

Oprettelsesproceduren involverer både certifikatsignering (RS) og udstedelse af e-penge (CES). Derfor afprøves i følgende test også udstedelse af e-penge. For at kunne teste dette kræves det, at alle servere i systemet kører. Alle applikationerne kan i princippet køres på samme maskine. For at afspejle forholdene i praksis bliver serverne dog så vidt muligt placeret på forskellige maskiner. Følgende maskiner benyttes under testen:

Maskinnr.	Processor	RAM	Applikation
1	1,6 GHz	512 MB	Klient
2	900 MHz	256 MB	RS og VS
3	2 GHz	640 MB	CES

Tabel 6 – Maskiner anvendt til test af systemets ydeevne.

Her skal det bemærkes, at RS og VS under de følgende tests aldrig belastes samtidig, hvorfor disse kan køre på samme maskine. Oprettelsesbeløbet blev i afsnit 8.2 fastsat til 56,5 kr, hvoraf 6,50 kr. bliver vekslet til e-mønter under oprettelsen. Dette betyder, at der skal udstedes 130 mønter á 0,05 kr. Under oprettelsen er der primært to ting, der er tidskrævende, nemlig certifikatdannelse i RS og udstedelse af penge i CES (og klient). Derfor ses der bort fra den tid netværkstrafikken kræver. Den totale tid for en oprettelse blev målt til 8:13,9. En tid på over 8 minutter må siges at være meget høj. I skemaet herunder ses tidsforbruget ved oprettelse fordelt på applikationer. Det ses hurtigt, at anvendelsen af blinde signaturer ved udstedelse af mønter, er meget tidskrævende (som forventet).

Tidsforbrug	Reel tid	Korrigeret tid
RS	12,6	5,6
CES	4:12,6	4:12,6
Klient	3:48,7	2:59,0

Tabel 7 – Tidsmålinger fordelt på applikationer ved oprettelse i systemet.

I søjlen 'korrigeret tid' er tiderne for RS og klient korrigeret, så tiderne svarer til samme processorhastighed i alle maskinerne. Derved kan tiderne bedre sammenlignes og forholdet mellem de forskellige tider kan vurderes. Undværes blinde signaturer under udstedelse, vil en oprettelse sandsynligvis kunne foretages på under 20 sekunder. Prisen for anonymitet er altså høj. En oprettelse skal dog kun foretages én gang. Vises et statusvindue for brugeren, vil denne muligvis kunne acceptere at skulle vente 8 min. Der kan også gives mulighed for at vælge at få mindre end 130 mønter udstedt. 130 mønter giver mulighed for afsendelse af 10 emails om dagen (som beskrevet i afsnit 8.2). Har

klienten brug for mindre, kunne denne blive oprettet hurtigere ved at hæve færre mønter. Generering af certifikat i RS foregår nemlig, som det ses, meget hurtigt.

8.3.1.c Opkrævning af e-mønter

Når en bruger ønsker at opkræve betaling for en modtaget email, kræver dette afsendelse af en notifikationsmail, samt at mønten flyttes på harddisken. En test, hvor der blev opkrævet betaling for 20 emails *samtidig,* gav en tid pr. email på 1,1 sekunder. Tiden vil ikke være ubetydelig for brugeren. Den kan dog ikke reduceres meget, idet den primært afhænger af tiden det tager at sende en email. Klientprogrammet kunne dog afsende notifikationsmails, uden at brugeren skal vente på dette. Hermed vil brugeren reelt kunne opkræve mønter uden ventetid.

8.3.1.d Udstedelse af e-mønter

Tiden det tager at udstede mønter, blev fastlagt i afsnittet omkring oprettelse, da der her også udstedes mønter. Derfor vil det her blot blive beregnet, hvor mange mønter den pågældende pengeserver i gennemsnit kan udstede. Da serveren i testen brugte godt 4 minutter på at udstede 130 mønter svarer dette til to mønter pr. sekund eller cirka 16 mill. mønter om året.

8.3.1.e Indløsning af e-mønter

Når mønter skal indløses, afhænger hastigheden dels af pengeserveren og dels af verifikationsserveren. I en test, hvor der blev indløst 20 mønter, blev tiden pr. mønt målt til 0,29 sekunder. Heraf gik de 0,15 sekunder til verifikation i verifikationsserveren. Omkring halvdelen af arbejdet udføres altså af verifikationsserveren.

Ved indløsning af mønter verificeres *alle* mønter ved at kontakte en verifikationsserver i implementationen. Dette er som omtalt i afsnit 6.8.4.c ikke nødvendigt. Belastningen på verifikationsserverne kan i praksis lempes, ved kun at verificere et udvalg af mønterne.

8.3.2 Blacklistserver

Når skalerbarheden skal vurderes er blacklistserveren uden tvivl flaskehalsen i SpamCash-systemet. Denne skal nemlig servicere *samtlige* brugere af systemet. Derfor er der gjort meget ud af, at dennes arbejde er så simpelt som muligt og at den tid, den enkelte klient er i kontakt med denne, er så kort som muligt.

I det følgende testes blacklistserverens ydeevne ved at lade en hær af klienter kontakte denne med forespørgsler. Der findes på nuværende tidspunkt ingen begrænsning på hvor mange kommunikationstråde, der kan oprettes i nogen af serverne. Dette vil i praksis være nødvendigt (specielt i BLS) for at forhindre DOS angreb. Målet med testen er, at give et estimat for, hvor mange brugere blacklistserveren kan servicere, og dermed give et estimat for, hvor mange brugere SpamCash-systemet kan håndtere.

Testen gennemføres ved at lade to maskiner danne 100 tråde hver og lade hver tråd kontakte blacklistserveren med de mulige forespørgsler:

- Hent hele transaktionslisten
- Hent en del af transaktionslisten
- Indrapporter mønter

Hver tråd vil i testen udføre en forespørgsel, vente på svar og dernæst starte forfra og udføre en ny forespørgsel. Dette gentages indtil testen ophører, dvs. indtil tilstrækkelig præcis testdata er indsamlet. Da der i praksis vil være stor forskel på frekvensen af de forskellige forespørgsler, har vi benyttet en sandsynlighedsfordeling, for at afgøre hvilken forespørgsel den enkelte klienttråd skal vælge. Sandsynlighedsfordelingen beskrives i det følgende.

Når systemet har nået en vis størrelse, vil andelen af nye brugere i forhold til antallet af eksisterende brugere ikke være ret stor. Vi antager, at denne andel er cirka 1% og at klienttrådene derfor kun forespørger på hele transaktionslisten i 1% af tilfældene. Det vil i reglen nemlig kun være nye brugere, som har behov for at hente hele transaktionslisten. I 98,5% af tilfældene hentes kun den nødvendige del af listen. Dette vil være langt den mest forekommende forespørgsel i systemet. I den sidste halve procent af tilfældene indsendes to (kopierede) mønter til blacklistserveren, som medfører en blacklistning og dermed en forøgelse af blacklistens størrelse. Forudsat at systemet fungerer korrekt (dvs. at der ikke er incitament for at misbruge systemet) synes 0.5% rimeligt.

Blacklistserveren blev startet på en maskine med en 2.0 Ghz Pentium-4 processor og 640 MB ram. Under testen observeredes det, at hukommelsesforbruget ikke var nævneværdigt højt og at dette holdt sig stabilt under hele testen. Blacklistserverens arbejde er således processorafhængigt frem for hukommelsesafhængigt. Dertil kommer, at der kræves en del båndbredde i forbindelse med kommunikationen med klienterne. Derfor fokuseres i det følgende på processor- og båndbreddeforbrug i serveren. Med cirka 55% processorforbrug formåede serveren i gennemsnit at afsende 301,5 blacklister pr. sekund under testen. I gennemsnit blev der fra serveren sendt med cirka 300 KByte/s og modtaget med cirka 230 Kbyte/s. Under testen observerede vi, at antallet af leverede blacklister steg nogenlunde lineært med processorforbruget. Vi antager derfor i det følgende, at denne lineære sammenhæng er gældende. Proxyprogrammet sørger for kun at kontakte blacklistserveren, hvis der er gået mere end 20 minutter siden dette skete sidst. Køres blacklistserveren på en 3.4 Ghz maskine forbundet til internettet med en 10 mbit / 10 mbit forbindelse, har vi følgende estimat for en øvre grænse på antallet af brugere i SpamCash-systemet.

I vores test-setup leverede blacklistserveren 301,5 blacklister pr. sekund. På en 3.4 Ghz processor under fuld belastning, ville dette, under antagelsen om linearitet give cirka en tredobling af dette antal. Dette vil igen medføre en tredobling af trafikken til og fra serveren, hvilket lige netop ligger inden for rammerne af hvad en 10 Mbit / 10 Mbit forbindelse kan klare. Dermed vil blacklistserveren kunne levere cirka 900 blacklister pr. sekund, hvilket giver 1.080.000 blacklists hvert tyvende minut. Såfremt samtlige brugere

af systemet checker mail døgnet rundt (og således henter en opdateret blacklist hvert tyvende minut), vil den øvre grænse for antallet af brugere være godt en million. Dette er dog en relativt pessimistisk sat øvre grænse, da langt de fleste brugere ikke vil checke mail så ofte. Et mere realistisk estimat for en øvre grænse vil derfor være i nærheden af det måske 4- eller 5-dobbelte antal brugere (altså cirka 5 millioner).

Det skal bemærkes, at blacklisten i vores test ikke når at blive ret stor. I det endelige system vil denne vokse med tiden og dette vil stille stadig større krav til internetforbindelsen til serveren. Langt den største del af brugerne vil kun hente opdateringer til blacklisten. En stor blacklist vil i disse tilfælde ikke være noget problem. Nyoprettede brugere vil dog udgøre en stadig større belastning for blacklistserverens internetforbindelse efterhånden som blacklisten vokser, da disse altid skal hente *hele* listen. Dermed kan det forventes, at flaskehalsen i blacklistserveren med tiden i højere grad vil være båndbredden end processorkraften.

Overordnet må vi udfra testen konkludere, at det er lykkes i vid udstrækning at gøre blacklistserveren effektiv. Skal systemet benyttes i global skala synes det dog nødvendigt at distribuere blacklistserverens arbejdsbyrde. Dette kommer vi nærmere ind på i følgende afsnit, hvor skalerbarheden af systemet diskuteres yderligere.

8.3.3 Skalerbarhed

I det følgende ser vi på om SpamCash-systemet egner sig til at blive benyttet i global skala. Dette gør vi ved at se nærmere på serverne i systemet, da det er disse, der vil blive udsat for pres fra brugerne i systemet. Hvor der er potentielle problemer med skalerbarheden gives der forslag til, hvordan disse kan løses. Det antages, at antallet af emailbrugere på verdensplan og dermed antallet af potentielle brugere af systemet, er 500 millioner, dvs. af størrelsesordenen 10^8 brugere.

Det fremgik af slutningen af afsnit 8.3.2, at flaskehalsen i SpamCash-systemet er blacklistserveren og at denne begrænser antallet af brugere til et tal af størrelsesordenen 10^6. Skal systemet bruges globalt, er det nødvendigt at distribuere blacklistserverens arbejdsbyrde til flere servere. Dette kunne gøres ved at udvide blacklistserveren til en decideret serverpark med mange servere, som alle kan nås fra internettet. Disse servere skulle opretholde én fælles blacklist og hver især kunne levere denne til klienter som ønsker dette. Ved opdatering af blacklisten på én server skulle en replikeringsmekanisme sørge for, at resten af serverparken blev informeret om denne opdatering. Blev disse servere endvidere optimeret yderligere med henblik på ydeevne, synes det ikke umuligt at systemet ville kunne anvendes i global skala.

Man kan dog ikke løbe fra, at båndbredde-forbruget, på især blacklistserveren, vil blive temmelig højt. Dette er naturligvis ikke gratis og vil skulle finansieres på en eller anden måde. En løsning kunne være at kræve et mindre månedligt gebyr hos brugerne til dækning af disse udgifter. En mere attraktiv løsning ville være at investere oprettelsesgebyret, som brugerne betaler under oprettelsen. Det årlige afkast fra denne investering kunne så benyttes til at finansiere udgifterne i forbindelse med driften af serverparken.

Vi ser nu på resten af serverne i systemet, for at få en ide om hvor mange af disse der vil være nødvendige i et endeligt globalt system og om dette antal synes acceptabelt.
I afsnit 8.3.1.d så vi, at en CES kunne udstede cirka to e-mønter i sekundet på en 2 Ghz maskine. Indløsning af en e-mønt tager i snit 0,29 sekund (afsnit 8.3.1.e), dvs. den samlede tid en CES skal bruge på at behandle en mønt i dennes levetid, er cirka 2,3 sekunder. Såfremt systemet virker efter hensigten, vil langt de fleste e-mønter under afsendelse af emails *ikke* blive indløst af modtager. Dette forlænger levetiden på e-mønterne, da der i disse tilfælde ikke tilføjes elementer til transaktionslisten på e-mønten (da der ikke har været tale om ejerskift). De 130 e-mønter er gyldige i 1 år, som beskrevet i afsnit 8.2.2. Herefter skal disse udskiftes med et nyt sæt e-mønter. Gennemsnitligt vil en CES således skulle bruge $130 \cdot 2{,}3$ sekunder eller cirka 5 minutter årligt pr. bruger. Dette svarer rundt regnet til, at hver CES vil kunne servicere 105.000 kunder med udstedelse og indløsning af penge. Dette tal kan øges ved at indsætte en kraftigere maskine, da operationerne i forbindelse med udstedelse og indløsning af e-mønter i høj grad er processorafhængige. Et realistisk tal vil altså være af størrelsesordenen 10^5 brugere pr. CES. Dette vil så kræve, at der i systemet fandtes CES'er i et antal af størrelsesordenen 10^3. Dette tal virker ikke skræmmende, da man sagtens kan forestille sig, at der findes mindst én server i hver bank.

VS'er er i snit 0,15 sekund om at verificere en mønt fra CES (afsnit 8.3.1.e). En CES er som nævnt ovenfor i alt 2,3 sekunder om at behandle en mønt i dennes levetid. For at VS'erne skal kunne servicere det omtalte antal CES'er, skal der derfor findes én VS for hver 15'ne CES. Dette gælder hvis samtlige indløste mønter verificeres. Udføres kun stikprøvekontrol på f.eks. hver tiende e-mønt, kræves kun én VS for hver 150 CES'er. Dermed kan antallet af VS'er holdes på et antal i størrelsesordenen 10^1. Det er ønskeligt, at dette antal er lavt, fordi hver VS skal have kendskab til blacklistserverens private nøgle og brugernes anonymitet afhænger af hemmeligholdelsen af denne.

RS er under 6 sekunder om at behandle en oprettelse. Dette giver mulighed for registrering af cirka $5 \cdot 10^6$ brugere årligt, hvilket umiddelbart virker fornuftigt. Set i sammenhæng med det estimerede antal af emailbrugere på verdensplan, ville det dog betyde at der bogstaveligt talt ville gå 100 år inden samtlige emailbrugere ville kunne være registreret. Dette er umiddelbart et problem, der dog vil kunne løses ved at opstille flere registreringsservere (med samme kopi af rodcertifikatet) og lade klienter vælge tilfældigt hvilken af disse, der skal kontaktes under oprettelsesproceduren. Ét problem i forbindelse med denne løsning er dog, at registreringsserverne så ikke længere ville kunne garantere, at samme emailadresse ikke findes i to forskellige certifikater, hvilket er en nødvendig sikkerhedsegenskab i SpamCash-systemet. For at denne løsning skal være brugbar, skal der altså findes en løsning på dette problem.

Samlet set kan det konkluderes, at det ikke virker urealistisk at systemet kan benyttes i global skala. Dette vil dog kræve, at systemet udvides og optimeres på en række centrale punkter, hvor blacklistserveren er det vigtigste af disse.

9 Konklusion

Dette afsluttende kapitel er delt i tre dele. Først vil vi kort opridse problemet, som er forsøgt løst i dette projekt. Herunder vil vi kort præsentere ideerne bag den udviklede løsning. Herefter beskrives de hovedresultater, der er opnået i forbindelse med projektet. Her identificeres også de problemer vi stødte på undervejs, og der gives et bud på, i hvilken sammenhæng den udviklede løsning kan indgå i en endelig spamløsning. Til sidst vil vi komme med en række forslag til forbedringer af systemet, når dette på et senere tidspunkt eventuelt skal videreudvikles.

9.1 SpamCash

At afsende kommercielle spammails i store mængder er i dag forbundet med en meget lille udgift for spammerne. Dertil kommer, at det er uhyre svært at opspore og identificere spammere, som ofte gemmer sig bag falske afsenderadresser. Den mest udbredte løsning på problemet i dag er, at benytte indholdsbaserede filtre for at komme størstedelen af spammails til livs. Dette introducerer dog risici for at legitime emails fejlagtigt filtreres fra. I takt med at mængden af spammails vokser, vil dette i stigende grad påvirke pålideligheden af emailsystemet i negativ retning.

Vores forslag er, at gribe fat i problemets kerne, i stedet for at forsøge at bekæmpe symptomerne. Vores løsning sigter på at fjerne incitamentet for at afsende spammails i første omgang. Dette gøres ved at lade en betaling blive påhæftet hver eneste email. Betalingens størrelse justeres således, at det økonomiske incitament for at spamme forsvinder. Ideen er nemlig, at opfatter en modtager en email som spam, kan betalingen beholdes, og ellers tilbageføres denne til afsender. Fungerer systemet som tilsigtet, vil spammere derfor blive straffet økonomisk, mens de, som ellers benytter emailsystemet, ikke vil blive påvirket økonomisk.

9.2 Resultater

Det er lykkedes at designe og implementere et system, der opfylder alle krav i kravspecifikationen. Det udviklede system er både skalerbart, offline, opretholder brugernes anonymitet og kan integreres i det eksisterende emailsystem. Systemets implementation antyder derudover, at systemet også fungerer i praksis. Der var dog flere steder, hvor der opstod problemer i forbindelse med designet.

For at systemet skal kunne skaleres, anvendes en alternativ certifikatmodel. Dette kræver en relativt stor mønt ved emailoverførsel, hvilket medfører et ikke ubetydeligt båndbreddeforbrug. Forsvinder al spammail, vil det totale båndbreddeforbrug på internettet falde betydeligt, men introducerer antispam-systemet et ligeså stort forbrug, er intet opnået på den front. Et andet problem i SpamCash-systemet er den centrale blacklistserver. Denne viste sig nødvendig, da systemet skal være offline og derudover understøtte transferability. Blacklistserveren gør det muligt at forhindre møntkopiering og misbrug, mens disse positive egenskaber bibeholdes. Til gengæld indføres en central kritisk enhed, hvilket kan blive et problem, hvis systemet skal anvendes i stor skala. Et

tredje problem er, at brugerne kan forhindre afsendelse af notifikationsmail. Dermed kan det komme til at se ud, som om uskyldige brugere har foretaget møntkopiering. Denne situation har vi ikke kunnet undgå, når det er et krav, at modtagere *altid* skal kunne indløse modtagne e-mønter.

Brugerne i systemet er anonyme, dog skal emailadresser registreres. Prisen for denne anonymitet er en tidskrævende registreringsproces, grundet betydeligt processorforbrug i pengeservere og klienter. Dette problem har begrænset betydning for, om brugere vil acceptere systemet og kan reduceres ved en optimeret implementation. Et andet forhold, der har stor betydning for brugernes accept af systemet, er oprettelsesbeløbet og spørgmålet er her, om brugere er villige til at betale 50kr., for at slippe for spam. I dag abonnerer mange på forskellige spamløsninger. Der findes altså brugere, der er villige til at betale for at løse problemet, og firmaer bruger allerede nu enorme summer på spambekæmpelse. Tages dette i betragtning, virker et engangsbeløb på 50 kr. som en lille forhindring.

SpamCash-systemet understøtter de mest almindelige emailprotokoller, men det er også nødvendigt at systemet fungerer fornuftigt i en overgangsfase, hvor ikke alle brugere er med i systemet. SpamCash-systemet giver brugere mulighed for at benytte en whitelist, men denne mekanisme kommer til kort mange steder (f.eks. i større virksomheder). Derfor vil en løsning hvor SpamCash udelukkende benyttes til at undgå false positives, måske være mere attraktiv. I SpamCash-systemet afvises al email, der ikke indeholder betaling. Det kunne være en mulighed, at lade SpamCash fungere med de eksisterende filtre på en sådan måde, at emails med SpamCash-betaling *altid* når frem, mens andre emails filtreres på normal vis. På denne måde kan risikoen for false positives elimineres, og samtidig kan en aggressiv filtrering sikre mod false negatives blandt de emails som ikke har vedhæftet betaling.

Løses ovennævnte problemer på en fornuftig måde, mener vi, at SpamCash-systemet kan være et seriøst bud på en effektiv spamløsning. Systemets enkle idegrundlag gør det intuitivt at bruge, og samtidig bliver emailsystemets velkendte positive egenskaber opretholdt.

9.3 Fremtidigt arbejde

Som systemet er implementeret nu, er det ikke muligt at justere på møntens værdi og dermed det beløb, der kræves for at få adgang til indbakken. Dermed skal der altid vedhæftes én emønt med værdien 5 øre, når der sendes en email. Dette er ikke umiddelbart hensigtsmæssigt, da det i mange tilfælde vil være forskelligt, hvor stort et beløb en bruger ønsker at kræve for adgang til dennes indbakke. I SpamCash kan det let lade sig gøre at vedhæfte mere end én mønt på hver email. Dermed vil justering af adgangsbeløb være muligt. Da en enkelt mønt i gennemsnit fylder 4-5KB, vil størrelsen af emails dog vokse betydeligt i denne situation. For at begrænse båndbreddeforbruget og for derigennem at kunne tillade justering af adgangsbeløb, er det nødvendigt at finde en måde at reducere møntens størrelse yderligere. I fremtidigt arbejde med systemet bør der fokuseres på dette, hvis systemet skal være mere praktisk anvendeligt.

Som systemet er nu, udføres automatisk signatur af afsendt email, sådan at modtager kan kontrollere, at emailen ikke har været modificeret undervejs fra afsender til modtager. Dette medfører dog, at afsender ikke kan løbe fra at have sendt emailen (*non-repudiation*). Der kan være situationer, hvor dette ikke er ønskeligt for afsender. I en fremtidig version af programmet kunne det derfor gøres valgfrit for afsender, om denne ønsker at signere indholdet af udgående emails. Bemærk at dette på ingen måde går ud over sikkerheden i betalingssystemet, da overførselssignatur og integritetssignatur naturligvis stadig skal foretages. Dog vil det give en angriber mulighed for, at ombytte indholdet af emailen med en spam-meddelelse (hvis denne har mulighed for at foretage et man-in-the-middle-angreb). Dermed sendes en spammail på afsenders regning. Hvis afsender kan acceptere denne risiko, er der intet i vejen for, at denne sender uden emailsignatur.

Der er flere ting i implementationen, der kan arbejdes med i fremtiden. For det første bør en kø-mekanisme indføres i IMAP-proxyen, for at opnå en hurtigere emailbehandling. Den del af implementationen, der omhandler udstedelse af penge, kan også optimeres for at opnå en kortere registreringsprocedure. Derudover kan der gives mulighed for, at registrere flere emailadresser og at en egentlig ejerkontrol-email afsendes ved oprettelse, for at kontrollere ejerskab af emailadresserne. Til sidst vil det gavne brugervenligheden væsentligt, hvis der blev givet flere muligheder for at se hvilke modtagere, der har opkrævet betaling. Skal systemet begrænse spam og straffe de rigtige økonomisk, er det utrolig vigtigt, at brugerne let kan se, hvem der opkræver betaling for hvilke emails. Derfor vil en brugergrænseflade, der tilbyder mere avanceret visning af pengestatus, være en ønskelig forbedring af systemet.

Appendiks A Installationsvejledning

A.1 Program-CD 152
A.2 Installation og start af klientprogram 153
A.3 Installation af CES (CurrencyExchangeServer) 153
A.4 Installation af BLS (BlackListServer) 153
A.5 Installation af VS (VerificationServer) 153
A.6 Installation af RS (RegistrationServer) 153
A.6.1 OpenSSL – Installation 154
A.6.2 OpenSSL – Oprettelse af certifikatautoritet 155

A.1 Program-CD

På den medfølgende program-CD findes følgende mapper:

- Client
- CES
- BLS
- RS
- VS
- Hjælpeprogrammer
    - OpenSSL
    - JAVA

Der findes altså en mappe for hver applikation samt en mappe med hjælpeprogrammer (jar-filer). For alle applikationer skal følgende installeres. Der skal installeres et JRE (Java Runtime Environment), samt de kryptografi- og mailbiblioteker, der skal bruges af programmerne. Nedenstående angiver punkt for punkt, hvordan dette konkret gøres:

1. Et JRE skal være installeret, version 1.4.2 eller nyere. Dette kan hentes direkte fra Sun's hjemmeside (http://java.sun.com), eller installeres (i MS Windows) fra den vedlagte program-cd.

2. De fire jar-filer i biblioteket programCD/hjælpeprogrammer kopieres til biblioteket $*JAVAHOME$/jre/lib/ext* (hvor *$JAVAHOME$* angiver placeringen af JRE). Herefter vil disse være tilgængelige for applikationerne. De fire jar-filer er som følger:
    - *bcprov-jdk14-120.jar*: Kryptografipakken *BouncyCastle* (www.Bouncycastle.org)
    - *logi.crypto1.1.2.jar*: Kryptografipakken *logi.crypto* (http://www.logi.org/logi.crypto/)
    - *mail.jar*: Sun's *javamail* (http://java.sun.com/products/javamail/)
    - *activation.jar*: Sun bibliotek, der kræves af javamail

3. I filen $*JAVAHOME$/jre/lib/security/java.security* tilføjes følgende linie (efter linien der begynder med *security.provider.5*):

    ```
    security.provider.6=org.bouncycastle.jce.provider.BouncyCastleProvider
    ```

    Dette er for at Bouncycastle kan fungere korrekt.

Herefter er det blot nødvendigt at installere den enkelte applikation, som beskrevet herunder. Her skal det bemærkes, at stierne til programmernes placering ikke må indeholde mellemrum:

A.2 Installation og start af klientprogram

Mappen *Client* fra program-cd'en kopieres til et valgfrit sted på harddisken (stien til programmet må dog ikke indeholde mellemrum). Derefter startes programmet vha. kommandoen:

```
.../Client/java –jar client.jar
```

Ved første opstart vil programmet guide brugeren igennem registreringsprocessen. Detaljerne omkring dette er beskrevet i brugsvejledningen i Appendiks B.

A.3 Installation af CES (CurrencyExchangeServer)

Mappen *CES* fra program-cd'en kopieres til et valgfrit sted på harddisken. Derefter startes programmet vha. kommandoen:

```
.../CES/java –jar currencyexchangeserver.jar
```

A.4 Installation af BLS (BlackListServer)

Mappen *BLS* fra program-cd'en kopieres til et valgfrit sted på harddisken. Derefter startes programmet vha. kommandoen:

```
.../BLS/java –jar blacklistserver.jar
```

A.5 Installation af VS (VerificationServer)

Mappen *VS* fra program-cd'en kopieres til et valgfrit sted på harddisken. Derefter startes programmet vha. kommandoen:

```
.../VS/java –jar verificationserver.jar
```

A.6 Installation af RS (RegistrationServer)

På maskinen hvor registreringsserveren skal installeres, skal OpenSSL også installeres (se næste afsnit). Først installeres OpenSSL som beskrevet i næste afsnit. Derefter kopieres mappen *RS* fra program-cd'en til en mappe i **roden af samme drev**, som OpenSSL er installeret. Herefter startes programmet vha. kommandoen:

```
.../RS/java –jar registrationserver.jar
```

A.6.1 OpenSSL – Installation

I SpamCash-systemet genereres certifikater vha. OpenSSL (www.OpenSSL.org). Registreringsserveren i systemet står for udstedelsen af certifikater, og fungerer således som certifikatautoritet. I det følgende beskrives, hvordan OpenSSL installeres og konfigureres til at køre sammen med SpamCash.

Installationsprogrammet køres fra biblioteket 'OpenSSL' på den medfølgende program-cd. Efter endt installation skal stien `openssl/bin/` tilføjes til systemets *path*. Herefter åbnes i biblioteket `openssl/bin/` den skjulte fil `openssl.cnf`. Her ændres den relative sti angivet i linien:

dir = ./demoCA

til den absolutte sti (uden '.')

dir = /RS/CA

Derudover ændres linien:

policy = policy_match

til

policy = policy_anything

I SpamCash systemet er der ikke behov for, at de udstedte certifikater skal indeholde information om brugerne (som f.eks. navn, lokalitet, køn, cpr-nummer etc.). Derfor denne sidste ændring, der netop gør, at alle felter ikke er nødvendige at udfylde i de udstedte certifikater.

Nu er OpenSSL konfigureret og installeret korrekt. Skulle der opstå problemer, kan følgende vejledning i hvordan certifikatautoriteten i systemet initialiseres benyttes.

A.6.2 OpenSSL – Oprettelse af certifikatautoritet

For at oprette en certifikatautoritet i OpenSSL, som er kompatibel med SpamCash-systemet, skal mappen `/RS/CA` kopieres fra program-cd'en til roden af det drev, hvor OpenSSL er installeret. Mappen er oprindeligt dannet ved at følge nedenstående vejledning.

I roden af det drev hvor OpenSSL er installeret, dannes først en tom mappe ved navn *RS*, og i denne dannes en tom mappe kaldet *CA* (Certificate Authority). I CA-mappen dannes nu to mapper, som navngives hhv. *private* og *newcerts*. Derudover dannes i mappen *CA* en tom fil ved navn *index.txt* samt en fil ved navn *serial* indeholdende udelukkende strengen '01'. Når dette er gjort skulle følgende mapper og filer findes på drevet:

`/RS/CA/private`	[bibliotek]
`/RS/CA/newcerts`	[bibliotek]
`/RS/CA/index.txt`	[fil]
`/RS/CA/serial`	[fil]

Med det på plads er næste skridt, at certifikatautoriteten danner sit selvsignerede rodcertifikat. Dette gøres vha. følgende kommando:

```
openssl req -x509 -newkey rsa:2048 -keyout /RS/CA/private/cakey.pem -
out /RS/CA/cacert.pem
```

Der vil i forbindelse med udførslen af ovenstående kommando, blive spurgt efter et password samt de oplysinger, der skal stå i certifikatet (land, lokalitet, navn etc.). Passwordet vil blive brugt, hver gang certifikatautoriteten skal udstede et nyt certifikat. Når ovenstående oplysninger er indtastet, vil rodcertifikatet blive dannet og gemt i filen `cacert.pem`. Den private nøgle bliver gemt i filen `cakey.pem`.
Certifikatautoriteten er nu klar til at udstede certifikater, og vil blive benyttet til at udstede samtlige certifikater, som vil blive benyttet i SpamCash systemet.

Appendiks B Brugsvejledning

B.1 SpamCash systemet 158
B.2 Registrering i systemet 158
B.3 Mail-proxy 159
B.4 Adgang til krypterede filer 161
B.5 Status-vinduet og Timeout-fejlbeskeder 163
B.6 Hovedvinduet 164
B.6.1 Pengestatus 164
B.6.2 Hæv e-mønter 165
B.6.3 Indsæt e-mønter 166
B.6.4 Indkasser ”SpamCash” 167
B.6.5 Opdater whitelist 169

B.1 SpamCash systemet

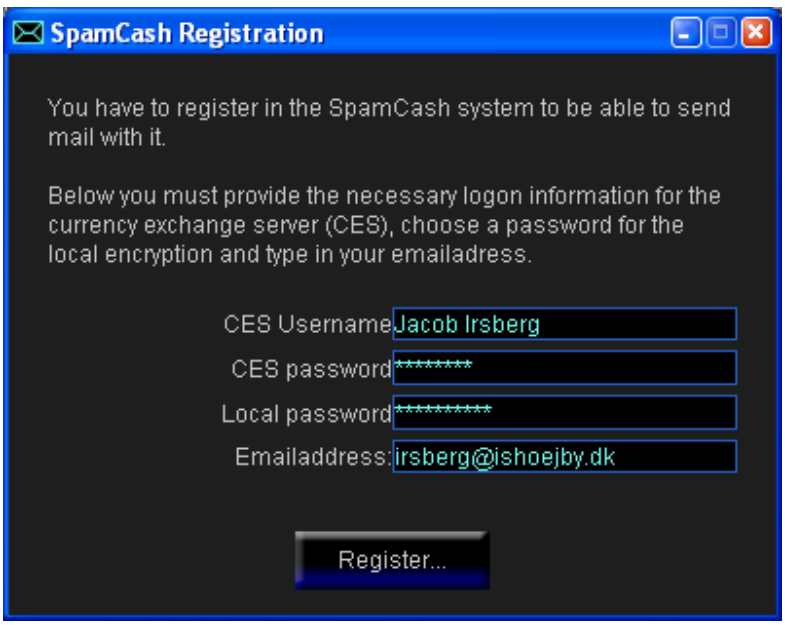
Når der sendes email med SpamCash-systemet, vil disse altid være påhæftet en elektronisk mønt. Når en sådan email modtages, afgør modtageren alene, om denne kan betragtes som spam. Hvis emailen ikke betragtes som spam, behøver modtageren intet at foretage sig og afsenderen vil efter en begrænset periode automatisk kunne sende med mønten igen. Er mailen derimod at betragte som spam, kan modtageren beholde mønten og afsender vil modtage besked om dette.

Udelukkende emails vedhæftet en gyldig mønt godtages, dvs. al anden mail vil blive afvist af systemet og aldrig nå modtagers indbakke, med mindre brugeren har tilføjet afsenderadressen til en whitelist over godkendte afsendere. Dermed forhindres spam i at nå en SpamCash-brugers indbakke, med mindre afsender betaler modtager for dette (eller er på whitelisten).

Når man oprettes som bruger i SpamCash-systemet, veksles et fast beløb til elektroniske mønter. Disse kan herefter benyttes til at sende email. Når man er oprettet, er det muligt at indløse og købe ekstra e-mønter i en af systemets bankservere.

B.2 Registrering i systemet

For at benytte systemet skal man registreres. Registreringen er anonym. Det eneste der skal angives, er den eller de emailadresser, der ønskes benyttet i systemet. Her kan 'anonyme' emailadresser, såsom Hotmail-adresser, også benyttes. Formålet med oprettelsen er at få tildelt et SpamCash-certifikat. Dette er nødvendigt for at kunne sende emails i systemet. Det første vindue der vises, når programmet startes første gang ses på Figur 40.



Figur 40 – Registrerings-vinduet. Det første der møder brugeren ved programmets første opstart.

For at starte registreringsproceduren skal felterne i vinduet udfyldes. Informationen i felterne ”CES Username” og ”CES password” er logon information til bankserveren (CES). At dette skal udfyldes, skyldes at programmet skal veksle penge til brug i systemet. Derudover skal vælges et password til lokale filer. De elektroniske penge, der hentes fra CES, gemmes i en fil på harddisken. Denne fil er altså penge værd, og skal derfor beskyttes, sådan at andre ikke kan stjæle og bruge pengene i filen. Den private nøgle hørende til SpamCash-certifikatet skal også beskyttes. Dette gøres på samme måde som hos de mest almindelige netbanker. Filerne krypteres 2 gange, nemlig med hhv.:

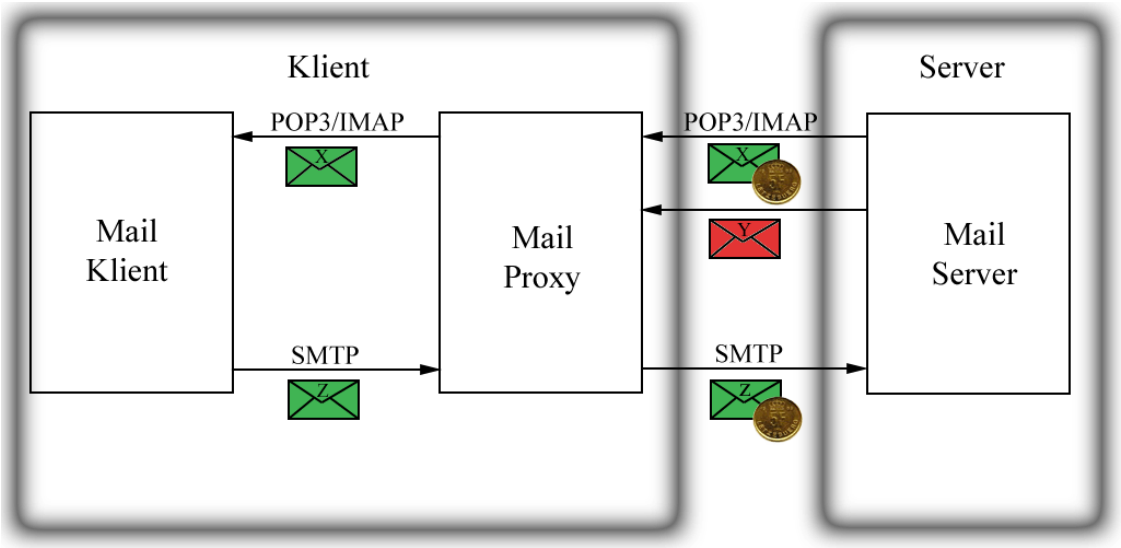
- Password og
- Nøglefil

Under registreringen vil programmet generere en nøglefil i ’data’-mappen i programbiblioteket med filnavnet ’Safe-key.key’. Det anbefales at denne fil efter afsluttet registrering flyttes til f.eks. en diskette. Hermed vil en hacker ikke kunne få adgang til pengene, selv hvis denne tiltvinger sig adgang til computerens harddisk.

Når felterne er udfyldt korrekt, trykkes på knappen ”Register…”. Herefter vil certifikatet og e-penge blive modtaget og gemt på harddisken og programmets hovedvindue vises. Programmet er herefter klar til at blive konfigureret.

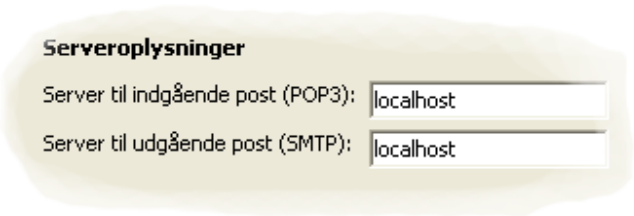
B.3 Mail-proxy

SpamCash-klienten fungerer blandt andet som en såkaldt mailproxy. Denne sørger for at al udgående email påhæftes elektroniske mønter inden afsendelse, samt for at opsnappe gyldige mønter på indkommende emails, inden disse leveres til indbakken. Derudover sørges der for, at emails uden gyldige påhæftede mønter slettes inden disse når indbakken. Mailproxyen er altså et slags mellemled mellem mailklient og mailserver. Email håndteres som vist på Figur 41.



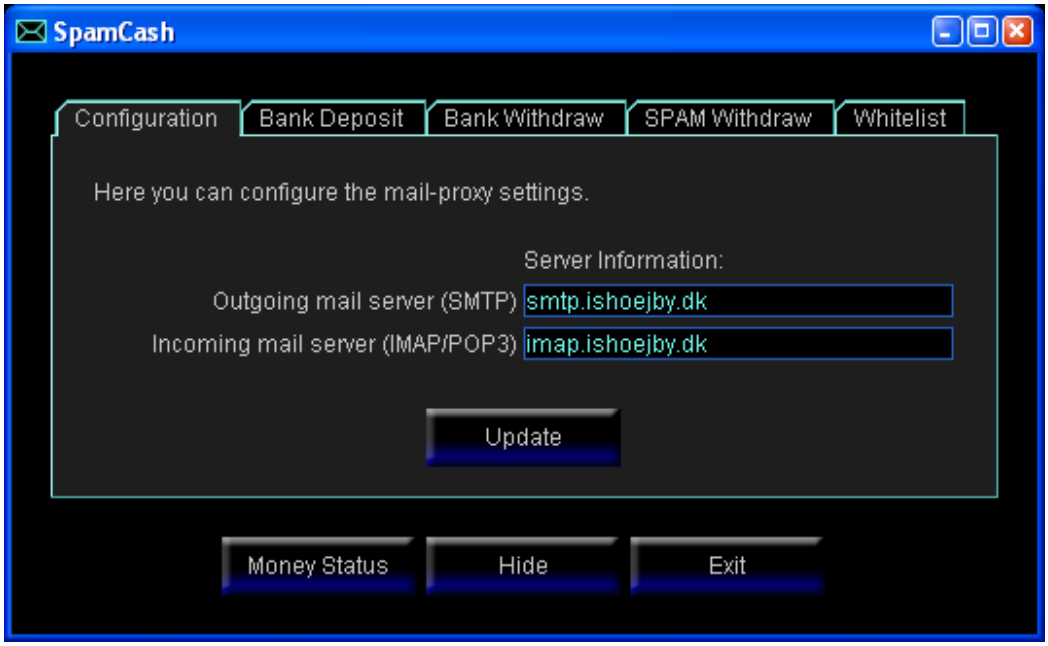
Figur 41 – Hvordan en mailproxy indgår i afsendelse og modtagelse af email.

Når man ønsker at sende og modtage mails med SpamCash-systemet, er fremgangsmåden nøjagtig den samme som i det gængse emailsystem. Din foretrukne emailklient kan altså benyttes som hidtil. Emailklienten og SpamCash-klienten skal dog konfigures til at fungere som illustreret på Figur 41. Dette gøres ved at sætte indgående og udgående mailserver til 'localhost' i emailklienten. I Microsoft Outlook ser dette ud som på figuren herunder



Figur 42 – Konfiguration af servere i emailklient

De servernavne, der normalt ville stå i disse felter (altså uden SpamCash) skal indtastes i SpamCash-klienten i stedet. På Figur 43 ses hovedvinduet i programmet, hvor konfigurations-fanebladet er valgt.

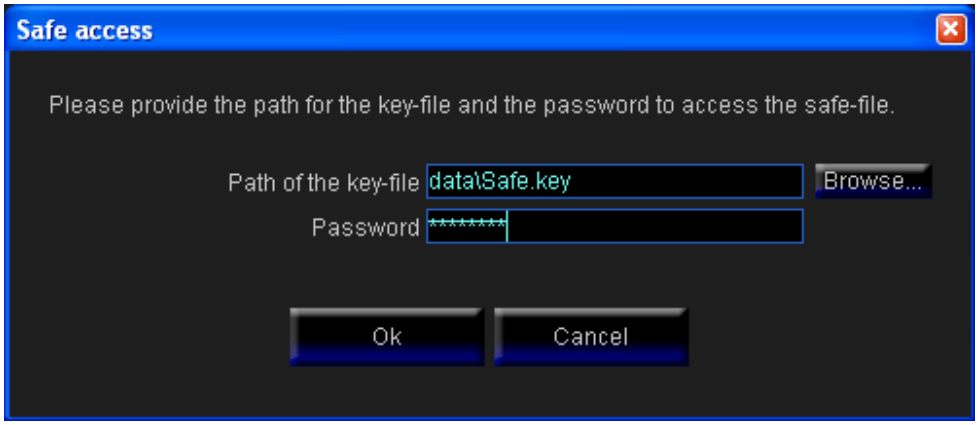


Figur 43 Hovedvinduet under fanebladet 'Configuration'.

Her indtastes servernavn eller IP-adresse for indgående og udgående emailservere. Ved tryk på "Update"-knappen, gemmes den indtastede konfiguration på harddisken. Programmet er nu klar til brug og du kan sende den første email. Det anbefales dog, at du læser afsnittet 'Adgang til krypterede filer', før du gør dette.

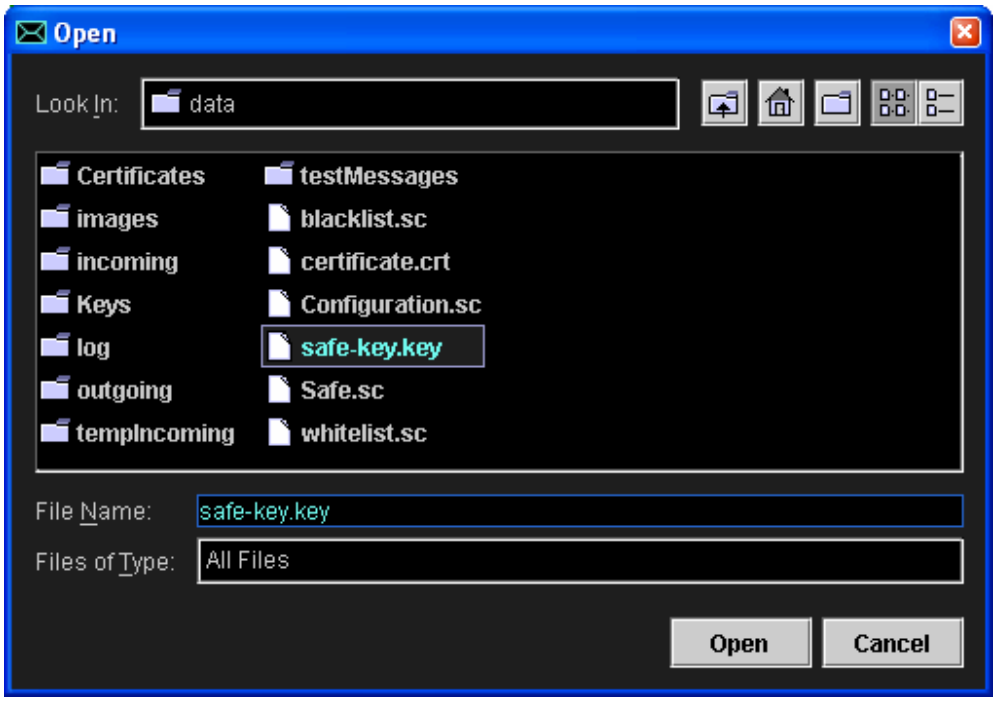
B.4 Adgang til krypterede filer

Elektroniske penge opbevares i filen *Safe.sc*. Filen er krypteret med nøglen i filen *Safe-key.key* og det lokale password, der vælges ved registrering. Programmet vil efter opstart, første gang det er nødvendigt, bede om stien til denne nøglefil og passwordet. Dette kan ske, når der skal hæves eller indsættes penge i banken eller når der skal sendes eller modtages email. Når dette sker, viser programmet en dialogboks som illustreret på Figur 44.



Figur 44 – Adgangsdialogboks der vises, når de krypterede elektroniske penge skal tilgås

Her indtastes stien til 'Safe-key.key'-filen og det lokale password. Det anbefales at filen anbringes på et flytbart drev, sådan at denne er bedre beskyttet. Programmet foreslår selv en sti til filen, men en valgfri sti kan indtastes eller vælges ved at trykke på "Browse". Trykkes "Browse" kan sti og filnavn angives vha. en dialogboks som vist på Figur 45.



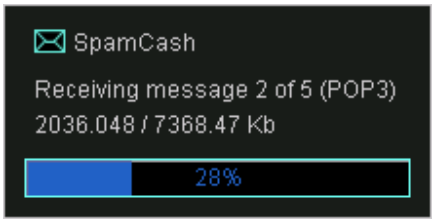
Figur 45 – Efter tryk på "Browse" fremkommer ovenstående dialogboks, hvor en fil kan vælges.

Når dette er gjort trykkes på "Open". Kan de elektroniske penge dekrypteres med den angivne fil og password, vil programmet udføre den handling, som gjorde at 'Safe access'-dialogboksen blev vist. Er passwordet forkert eller filens indhold ukorrekt, vil programmet give en fejlbesked, som den der er vist på Figur 46, hvor passwordet er forkert. Ved at trykke på "Cancel" kan operationen fortrydes.

Figur 46 – Fejlbesked ved åbning af pengeskabet.

B.5 Status-vinduet og Timeout-fejlbeskeder

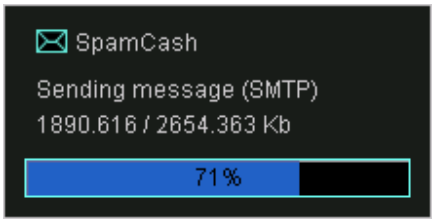
Ved afsendelse og modtagelse af email vises et statusvindue i nederste højre hjørne af skærmen. Dette er illustreret på Figur 47, hvor der hentes email vha. POP3.



Figur 47 – Status-vinduet ved modtagelse af email.

Hentes f.eks. emails med POP3 som på Figur 47, og det tager over 20-30 sekunder, kan der forekomme timeout i mailklienten. Dermed vil der sandsynligvis blive vist en fejlmeddelelse i denne. Hvis statusvinduet er til stede, når en sådan timeout indtræffer, vil det blot betyde følgende. SpamCash-klienten har ikke kunnet nå at hente emailen fra serveren og begynde at videregive denne til klienten i tide. **Ingen emails vil gå tabt** pga. dette. Alt der kræves, er blot endnu et tryk på 'send/modtag'-knappen i emailklienten, når statusvinduet er forsvundet, og emailen vil blive modtaget i klienten.

Ved afsendelse af email vil der ikke opstå timeout i de mest gængse emailklienter pga. SpamCash-klienten. Der vises dog stadig et statusvindue, så det er muligt at følge emailens vej på skærmen. Se Figur 48.



Figur 48 – Status-vinduet ved afsendelse af email.

B.6 Hovedvinduet

De øvrige knapper nederst i hovedvinduet har følgende funktioner:

- **Exit** Programmet lukkes ned
- **Hide** Vinduet minimeres
- **Money-status** viser status over de elektroniske penge. Et eksempel på dette kan ses i afsnittet ’Pengestatus’.

B.6.1 Pengestatus

Når der trykkes på knappen ’Money status’ i hovedvinduet, fremkommer en dialogboks som vist på Figur 49.

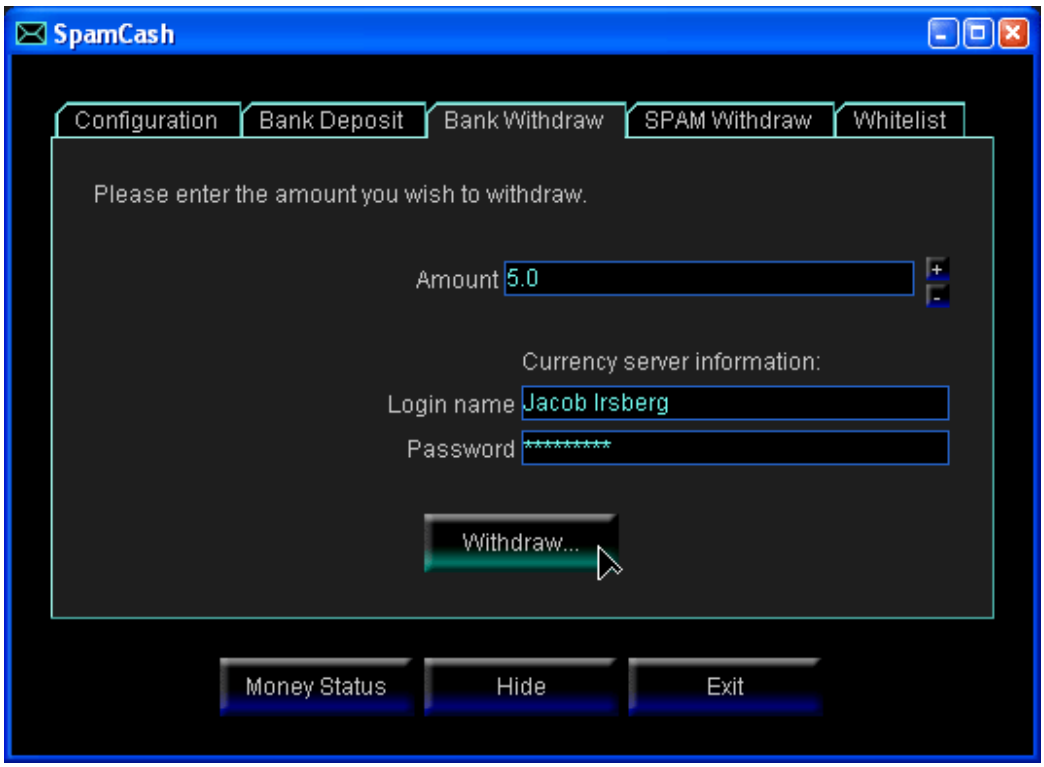
Figur 49 – Pengestatusvinduet der vises efter tryk på knappen ’Money-status’ i hovedvinduet.

Her kan ses status for de lokalt opbevarede e-penge. Det er vist, hvor mange mønter, der er klar til brug (*Usable coins*), altså hvor mange emails, der kan sendes her og nu. Derudover er vist, hvor mange mønter der er i ’Vente tilstand’ (*Waiting coins*). Disse mønter har tidligere været anvendt til at sende emails og bliver altså returneret efter en vis periode (såfremt modtagerne ikke indløser disse). Efter møntantallet er angivet værdien af mønterne.

Under møntantallene kan ses en liste med mailadresser. Denne liste viser hvor mange mønter, de forskellige modtagere har beholdt. I hver række er først angivet, hvor mange emails en given modtager har beholdt. Derefter hvor mange emails, der er blevet sendt til denne og til sidst emailadressen. Listen er sorteret efter, hvor mange emails modtagerne har beholdt. Den øverste på listen er altså den, der har beholdt flest mønter.

B.6.2 Hæv e-mønter

Vælges fanebladet 'Bank Withdraw' ser hovedvinduet ud som på Figur 50. Her gives mulighed for at hæve penge i banken.

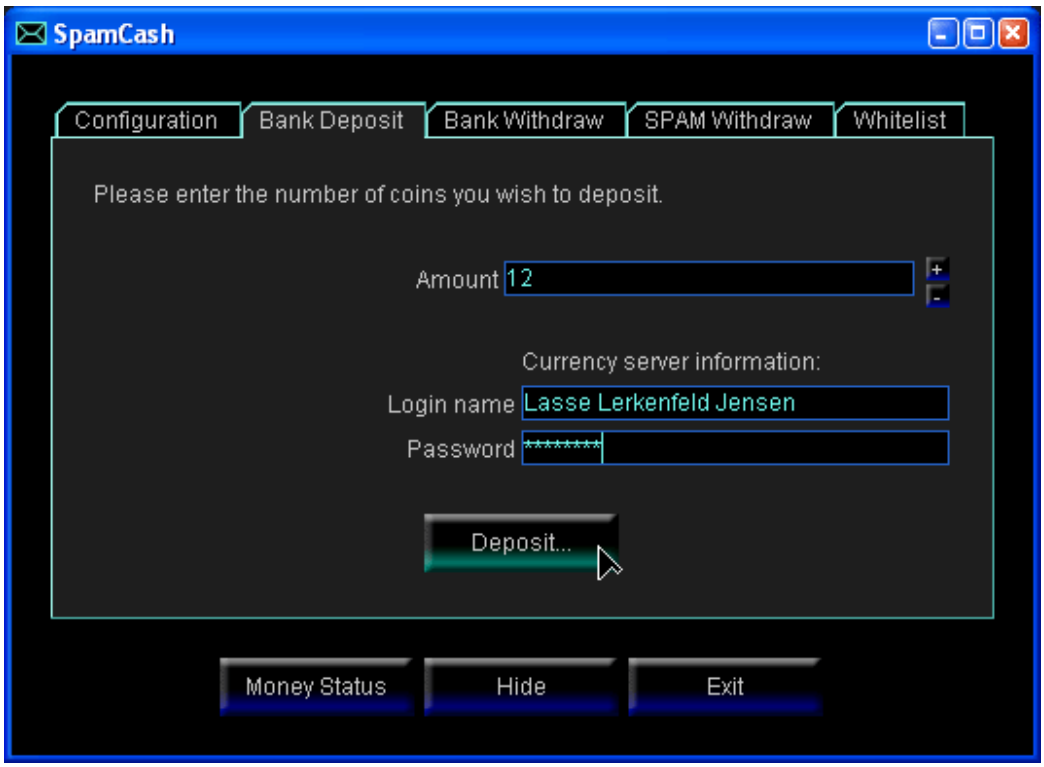


Figur 50 – 'Bank Withdraw'-fanebladet

Er der brug for flere e-mønter, kan hér hæves flere. I feltet 'Amount' indtastes det beløb, der ønskes hævet. Alternativt kan trykkes på '+' og '–' knapperne til højre. Når det ønskede beløb er valgt, udfyldes felterne 'Login name' og 'Password' med login-information til pengeserveren. Til sidst trykkes på knappen 'Withdraw', hvorefter programmet vil forsøge at hæve det ønskede beløb. Er dette successfuldt, vil 'Pengestatus'-vinduet vises. Her kan det ses, om det ønskede antal e-mønter er blevet hævet. Kan det ikke lade sig gøre at hæve det ønskede beløb, vises en passende fejlbesked. Ud fra fejlbeskeden kan de indtastede data rettes, og der kan forsøges igen. **Ingen penge vil blive hævet, hvis der opstår en fejl, uanset hvilken fejl der opstår.**

B.6.3 Indsæt e-mønter

Ønskes det at indsætte købte eller indkasserede e-mønter i banken, gøres dette ved først at vælge fanebladet 'Deposit'. Se figur Figur 51.



Figur 51 – Deposit fanebladet i hovedvinduet.

Her skal de samme felter udfyldes, som når der hæves penge. Beløbet skal dog være et heltal, da det angiver antallet af mønter, der ønskes indsat i banken. Når det ønskede antal mønter er valgt og login-informationen er udfyldt, trykkes på knappen 'Deposit'. Herefter vil programmet forsøge at indsætte det valgte antal mønter. Er dette successfuldt, vil det blive vist i en dialogboks, som den på Figur 52

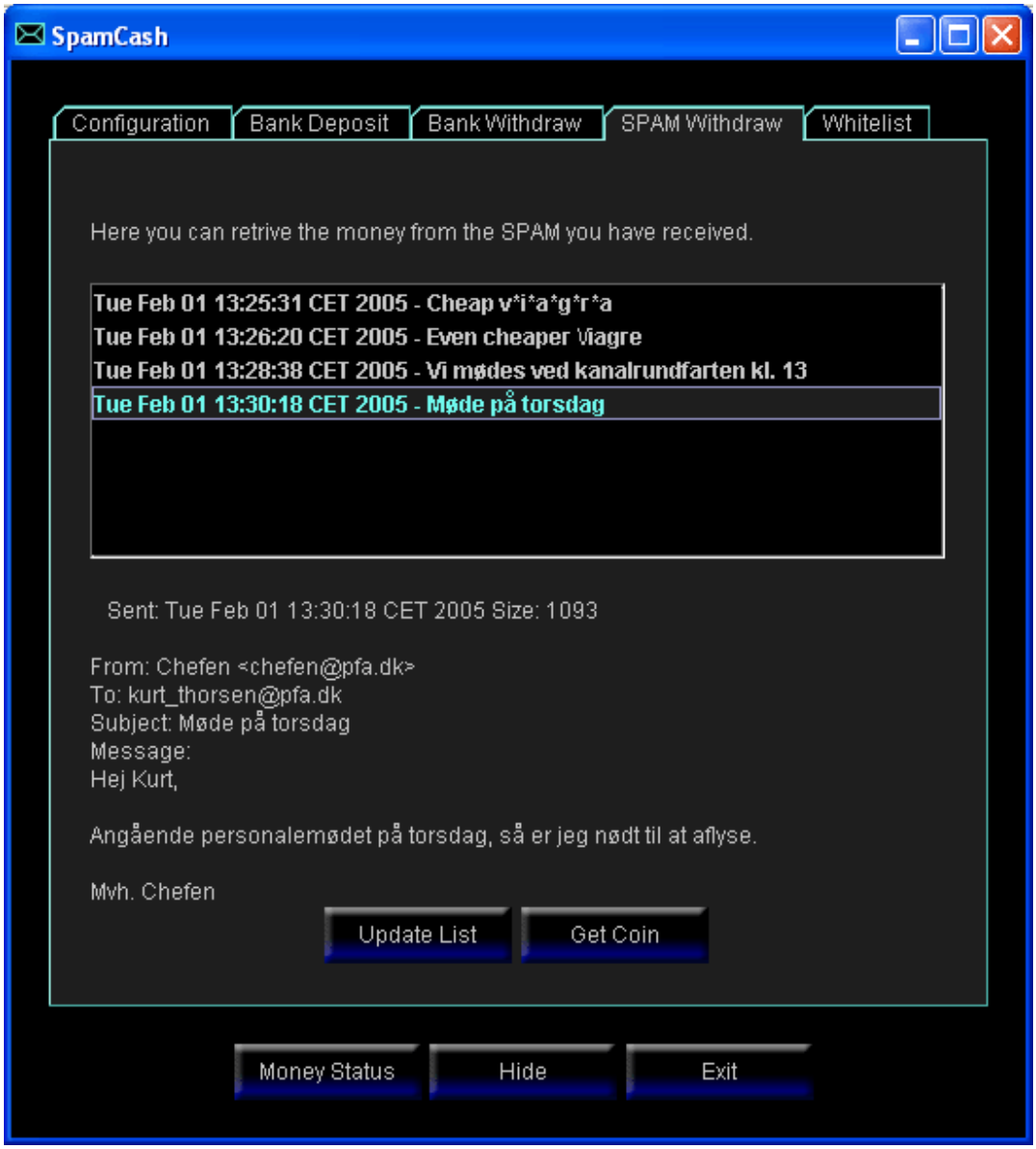
Figur 52 – Informationsbesked efter successfuld overførsel af e-mønter til banken.

Sker der en fejl eller er det ikke muligt at indsætte de ønskede antal mønter, vises en fejlbesked som f.eks. den på Figur 53, hvor det har været forsøgt at indsætte 100.000 mønter, mens der blot var 16 tilstede på harddisken. Ud fra fejlbeskeden kan de indtastede data rettes og der kan forsøges igen. **Ingen penge vil gå tabt** uanset hvilken fejl der opstår.

Figur 53 – Fejlbesked når brugeren forsøger at indsætte flere mønter end denne har.

B.6.4 Indkasser ”SpamCash”

Under fanebladet ’Spam Withdraw’ er det muligt at opkræve penge for modtagne emails. Alle modtagne emails, for hvilke betalingen ikke er udløbet, vises på listen under dette faneblad. På Figur 54 kan ses et eksempel på en sådan liste, hvor der både er modtaget spammails og regulære emails (naturligvis alle med betaling). I listen vises ét element pr. email. I hvert element i listen vises hvilket tidspunkt emailen er afsendt samt emnet i emailen. Tidspunktet kan give en indikation af, hvornår betalingen udløber og returneres til afsender. Ved at trykke på ’Update’-knappen opdateres listen og evt. udløbne mønter vil forsvinde. Klikkes på et element i listen fremkommer yderligere information om den pågældende email, som det kan ses på Figur 54, hvor bl.a. dato, størrelse, emne og indhold er vist.



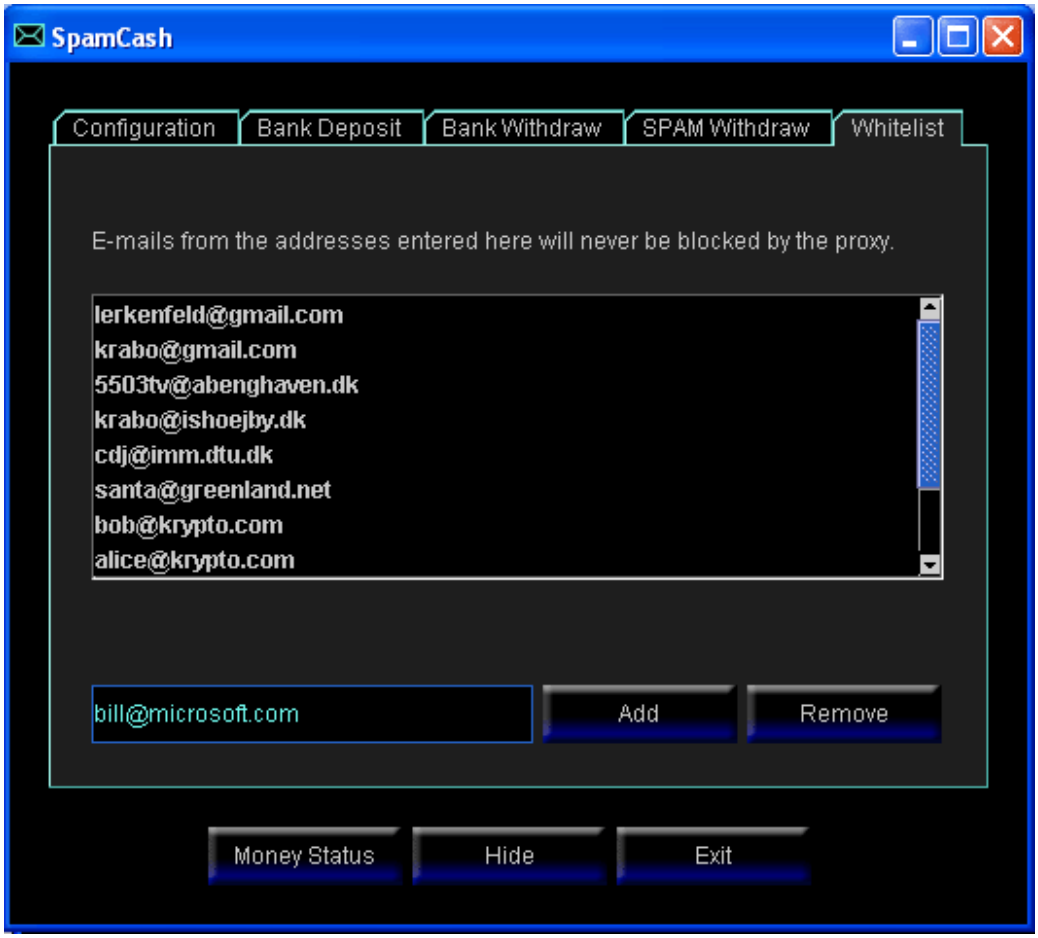
Figur 54 – 'Spam withdraw'-fanebladet hvor det er muligt at opkræve penge for modtagne emails.

For at opkræve betaling for en eller flere emails, markeres disse i listen ved at klikke på dem. Dette kan kombineres med SHIFT og CTRL tasterne, for at markere flere på en gang. Herefter trykkes på knappen 'Get coin' og den (eller de) pågældende mønt(er) vil blive gemt på harddisken og kan herefter bruges til at sende email. Når dette er succesfuldt overstået, vises beskeden på Figur 55. Her kan ses hvor mange mønter, der er blevet opkrævet og pengestatus (se evt. afsnittet 'Pengestatus').

Figur 55 – Dialogboks efter tryk på knappen 'Get coin'.

B.6.5 Opdater whitelist

Emails, der ikke er vedhæftet betaling afvises af programmet, med mindre afsenderadressen står på whitelisten. Vælges fanebladet 'Whitelist' i hovedvinduet kan denne liste ses. Se Figur 56. I SpamCash-systemet er det altså muligt at modtage emails fra personer, der ikke benytter systemet.



Figur 56 – Whitelistfanebladet i hovedvinduet.

For at tilføje en emailadresse til listen, indtastes emailadressen i tekstfeltet i bunden af fanebladet. Herefter trykkes på knappen 'Add'. Nu tilføjes emailadressen til whitelisten og email fra denne adresse vil herefter altid modtages. Ønskes det siden hen at fjerne en adresse fra listen, markeres denne ved at klikke i listen. Herefter trykkes på 'Remove', og adressen fjernes fra listen.

Appendiks C Litteratur

I følgende referencer optræder en række internetadresser, hvis indhold kan have ændret sig på nuværende tidspunkt. Derfor understreges det, at der er refereret til internetadressernes indhold i Januar og Februar 2005.

[1] *The Time Overhead of Email*, A Ferris Research White Paper, Dec 2003

[2] *The Cost of Spam*, A Ferris Research White Paper, March 2003

[3] DNS Blacklist – www.solarwinds.net

[4] Spam Cannibal – www.spamcannibal.org.

[5] MAPS (Mail Abuse Prevention System) – www.mail-abuse.org

[6] Postini Statistics – www.emcs.net/Protection/statistics.htm

[7] Sahami, M., Dumais, S., Heckerman, D., and Horvitz, *A Bayesian approach to filtering junk e-mail*, 1998.

[8] SpamFighter – www.spamfighter.com

[9] Razor – razor.sourceforge.net

[10] Distributed Checksum Clearinghouse – www.rhyolite.com/anti-spam/dcc

[11] Brightmail – www.brightmail.com, Symantec corporation, www.symantec.com

[12] *Filtering Technologies in Symantec Brightmail AntiSpam™ 6.0*, Symantec Corporation.

[13] SpamAssassin – www.spamassassin.apache.org,

[14] C. Dwork et al, *Pricing via Processing, Combatting Junk Mail*, 1992

[15] C. Dwork et al, *On Memory-Bound Functions for Fighting Spam*

[16] HashCash – www.hashcash.org

[17] M. Abadi et al, *Moderately Hard, Memory-bound Functions*

[18] *IBM Systems Journal Vol 41, No 4*, 2002

[19] M.Abadi et al, Certified *Email with a Light Online Trusted Third Party: Design and Implementation*, Maj 2002

[20] Bruce Schneier & James Riordan, *A Certified E-Mail Protocol*

[21] Cashette – www.cashette.com

[22] Lorrie Faith Cranor and Brian A. LaMacchia, *Spam!*, Communications of the *ACM*. Vol. 41, No. 8, Aug. 1998

[23] GnuPG – www.gnupg.org

[24] Sender ID – www.ietf.org/internet-drafts/draft-ietf-marid-core-03.txt

[25] Captcha project – www.captcha.net

[26] KnowSpam – www.knowspam.net

[27] SpamSlam – www.ilesa.com/software/spamslam-win.html

[28] Robert J. Hall, *An overview of the email-channels technology underlying ZoEmail*, April 2003

[29] Robert J. Hall, *Channels: Avoiding unwanted electronic mail*, 1997

[30] CANSPAM – http://informationweek.com/story/showArticle.jhtml?articleID=56900503

[31] CANSPAM – http://biz.yahoo.com/bw/050103/35140_1.html

[32] Martin Abadi et al., *Bankable Postage for Network Services*, 2003

[33] T. Okamoto & K. Otha, *Universal electronic cash*, 1992

[34] Ricarda Weber, *Chablis - Market Analysis of Digital Payment Systems*, August 1998

[35] David A. Turner and Ni Deng, *Payment-Based Email*, Dec. 2003

[36] David A. Turner & Daniel M. Harvey, Controlling Spam through Lightweight Currency, Nov. 2003

[37] A Ronald L. Rivest & Adi Schamir, *PayWord and Micromint: Two simple micropayment schemes*, Maj 1996

[38] B T. Eng & T. Okamoto, *Single term divisible electronic coins*, 1998

[39] C Wenbo Mao, *Lightweight Micro-cash for the Internet*, 1996

[40] Hitesh Tewari, Donal O'Mahony, Michael Peirce, *Reusable Offline Electronic Cash Using Secret Splitting*, December 1998.

[41] NIST. *Key Management Guideline - Workshop Document*. Draft, October 2001. http://csrc.nist.gov/encryption/kms/key-management-guideline-(workshop).pdf.

[42] Douglas R. Stinson, *Cryptography Theory and Practice Second Edition*, 2002 Chapman & Hall

[43] Bruce Schneier, *Applied Cryptography Second Edition*, 1997 WILEY

[44] J. M. Seigneur, C. D. Jensen, *Privacy Recovery with Disposable Email Addresses*, IEEE Security&Privacy, special issue on Understanding Privacy, November-December 2003

[45] Levente Buttyán and Jean-Pierre Hubaux, *Nuglets: a Virtual Currency to Stimulate Cooperation in Self-Organized Mobile Ad Hoc Networks*, Jan 2001

[46] R. Sharp & J. T. Kristensen, *Software Development Projects*, 1999

[47] Xiaoyun Wang et. Al, *MD4, MD5, HAVAL-128 og RIPEMD*, August 2004

[48] Xiaoyun Wang et. Al, *Collision search attacks on SHA-1*, Feb 2005

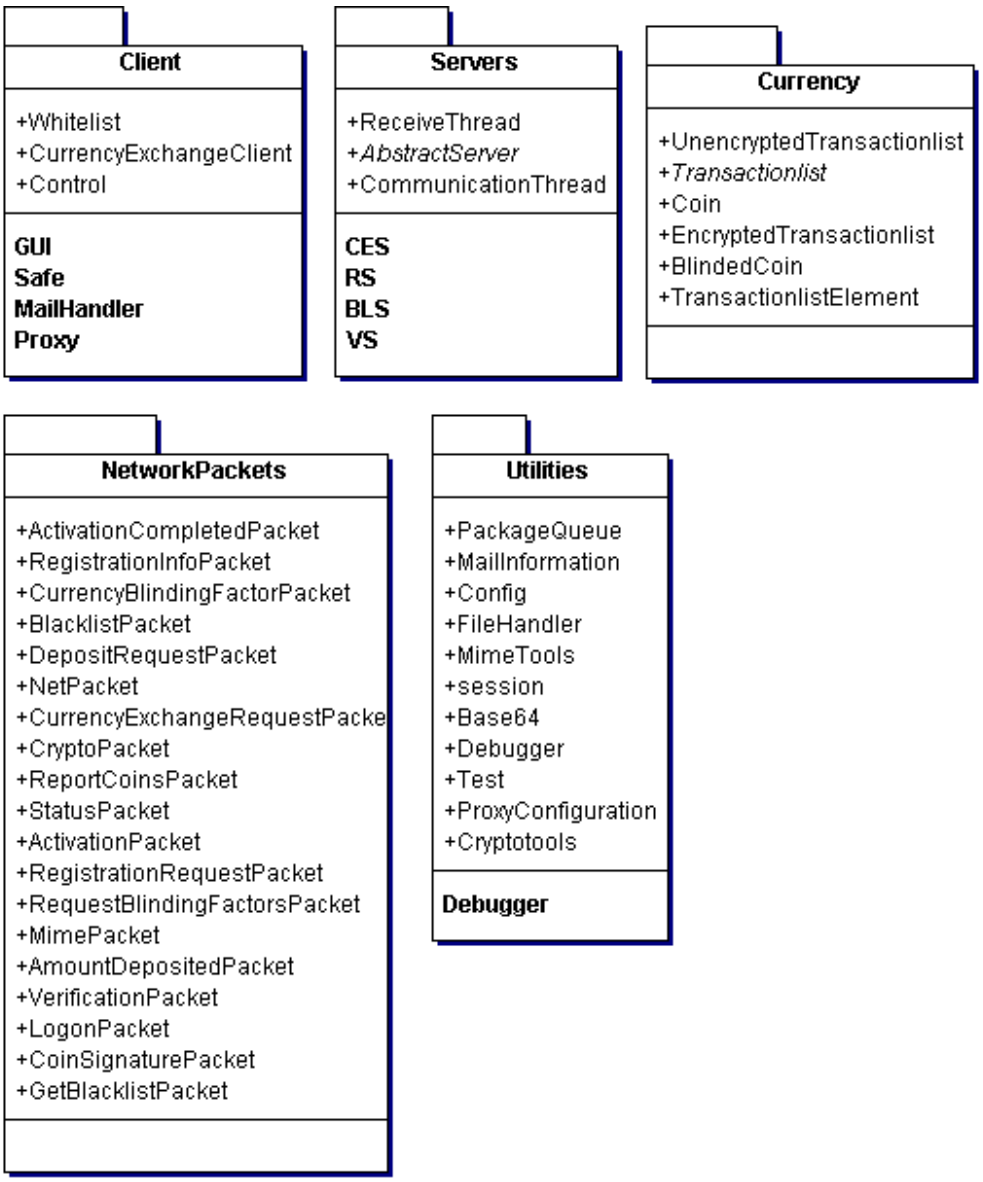
[49] Rockbridge Associates Inc., The 2004 National Technologi Readiness Survey, Feb 2005

Appendiks D Diagrammer

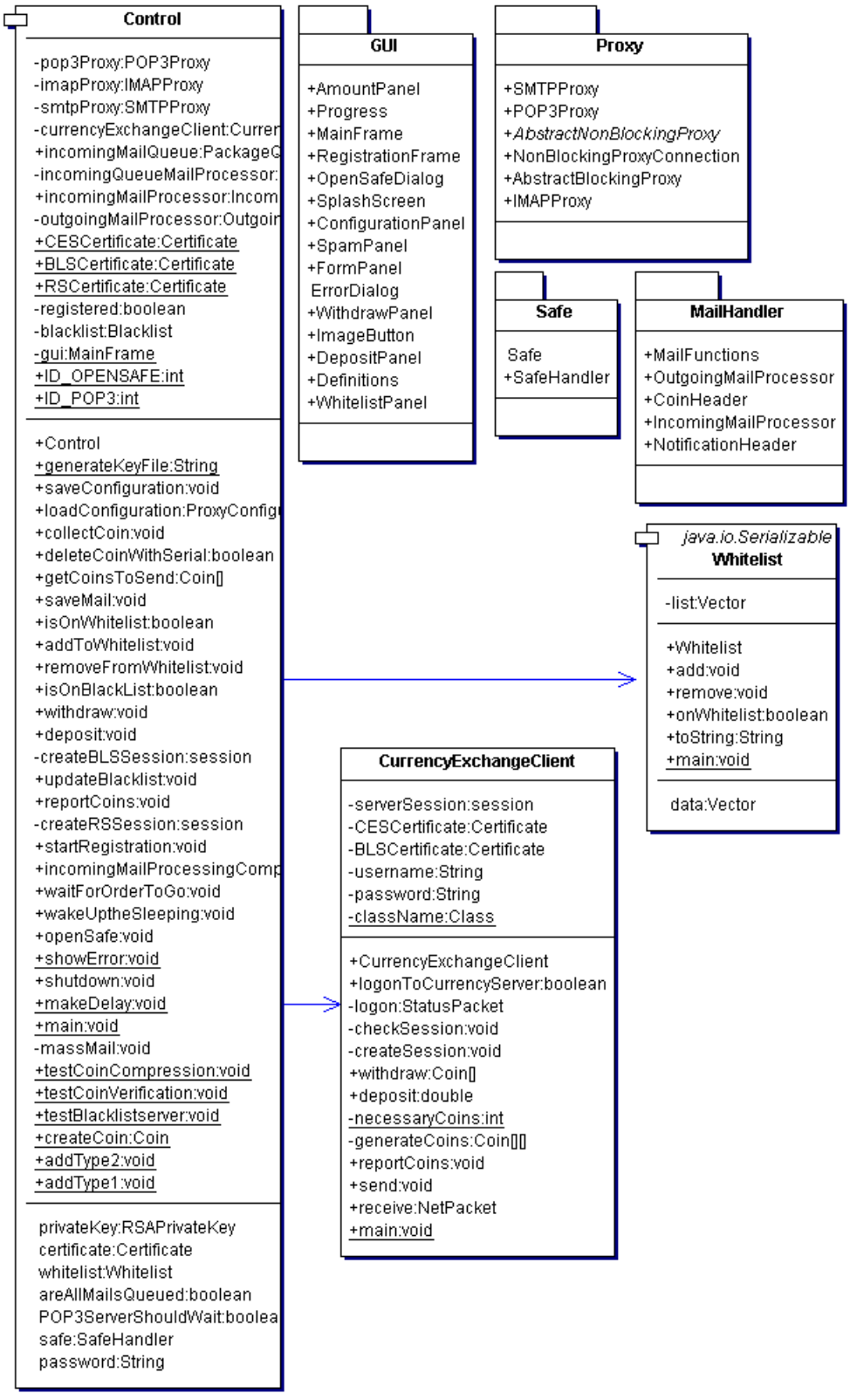
I dette appendiks forefindes UML-klassediagrammer for alle klasser i programmet. Der findes et diagram for hver pakke, og pakkerne kan ses herunder:

- SpamCash
  - Client
    - GUI
    - MailHandler
    - Proxy
    - Safe
  - Currency
  - NetworkPackets
  - Servers
    - BLS
    - CES
    - RS
    - VS
  - Utilities

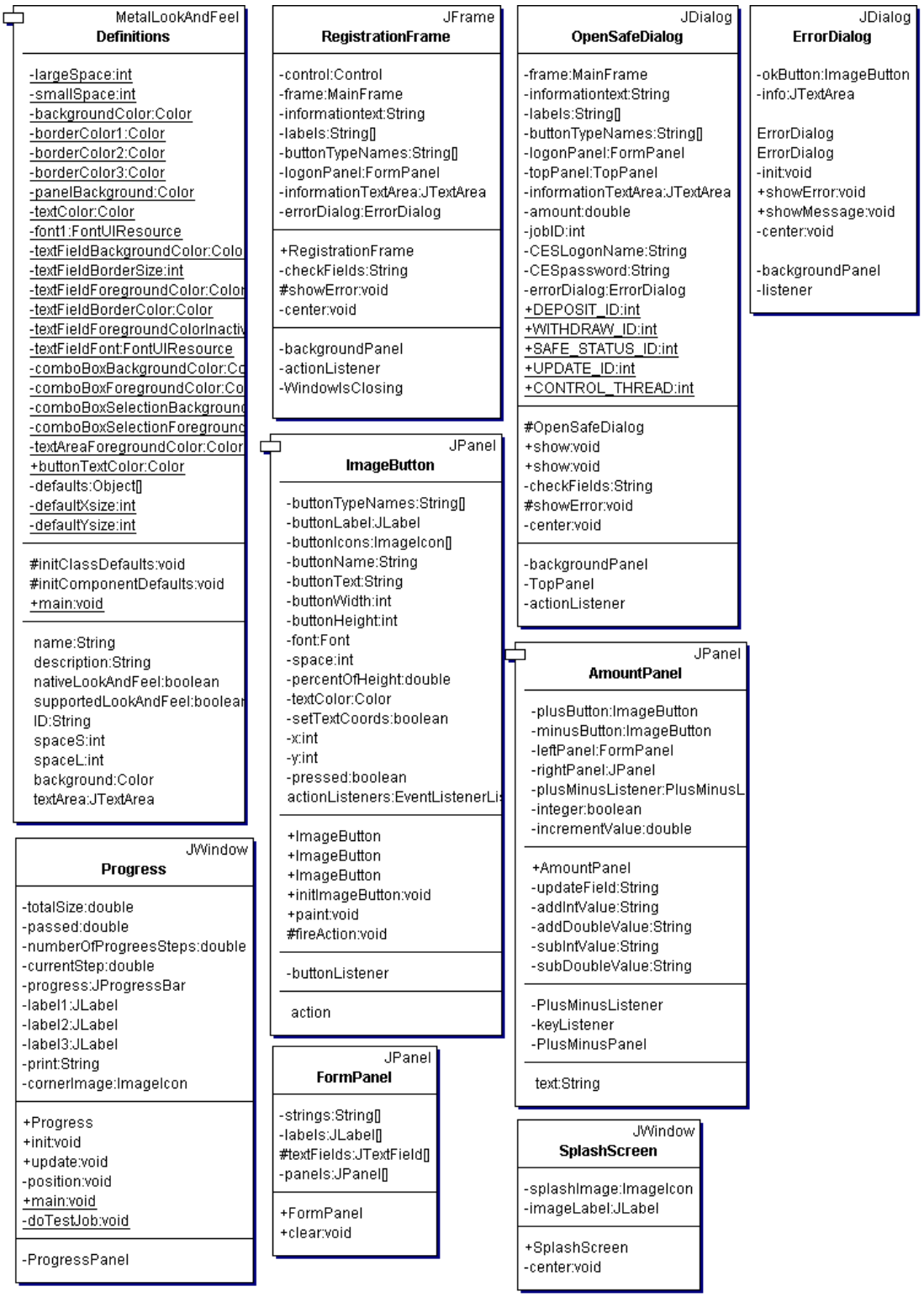
SpamCash


Client
+Whitelist
+CurrencyExchangeClient
+Control
GUI
Safe
MailHandler
Proxy
Servers
+ReceiveThread
+AbstractServer
+CommunicationThread
CES
RS
BLS
VS
Currency
+UnencryptedTransactionlist
+Transactionlist
+Coin
+EncryptedTransactionlist
+BlindedCoin
+TransactionlistElement
NetworkPackets
+ActivationCompletedPacket
+RegistrationInfoPacket
+CurrencyBlindingFactorPacket
+BlacklistPacket
+DepositRequestPacket
+NetPacket
+CurrencyExchangeRequestPacke
+CryptoPacket
+ReportCoinsPacket
+StatusPacket
+ActivationPacket
+RegistrationRequestPacket
+RequestBlindingFactorsPacket
+MimePacket
+AmountDepositedPacket
+VerificationPacket
+LogonPacket
+CoinSignaturePacket
+GetBlacklistPacket
Utilities
+PackageQueue
+MailInformation
+Config
+FileHandler
+MimeTools
+session
+Base64
+Debugger
+Test
+ProxyConfiguration
+Cryptotools
Debugger

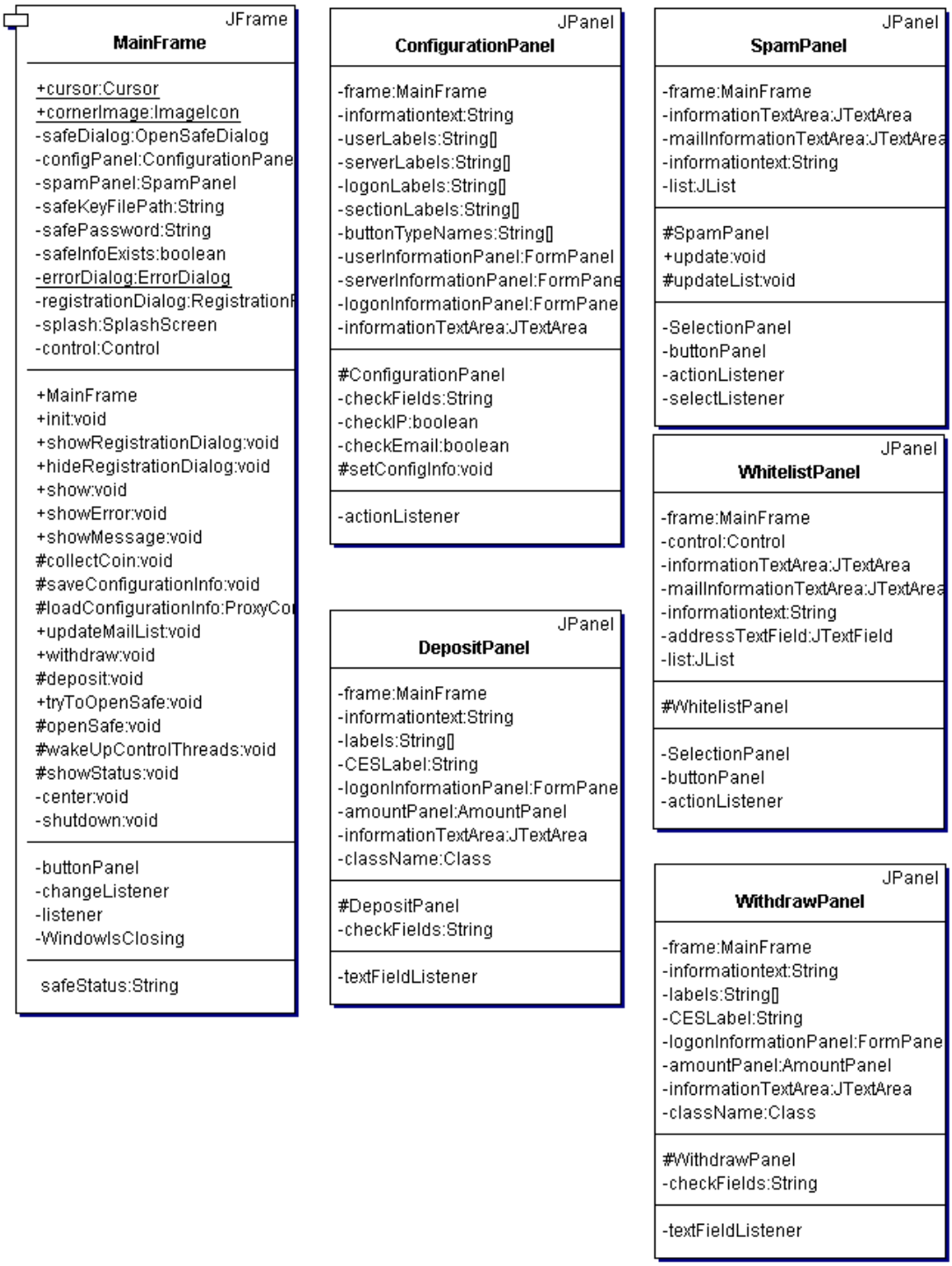
Client


Control
-pop3Proxy:POP3Proxy
-imapProxy:IMAPProxy
-smtpProxy:SMTPProxy
-currencyExchangeClient:Currer
+incomingMailQueue:PackageQ
-incomingQueueMailProcessor:
+incomingMailProcessor:Incom
-outgoingMailProcessor:Outgoir
+CESCertificate:Certificate
+BLSCertificate:Certificate
+RSCertificate:Certificate
-registered:boolean
-blacklist:Blacklist
-gui:MainFrame
+ID_OPENSAFE:int
+ID_POP3:int
+Control
+generateKeyFile:String
+saveConfiguration:void
+loadConfiguration:ProxyConfigu
+collectCoin:void
+deleteCoinWithSerial:boolean
+getCoinsToSend:Coin[]
+saveMail:void
+isOnWhitelist:boolean
+addToWhitelist:void
+removeFromWhitelist:void
+isOnBlackList:boolean
+withdraw:void
+deposit:void
-createBLSSession:session
+updateBlacklist:void
+reportCoins:void
-createRSSession:session
+startRegistration:void
+incomingMailProcessingComp
+waitForOrderToGo:void
+wakeUptheSleeping:void
+openSafe:void
+showError:void
+shutdown:void
+makeDelay:void
+main:void
-massMail:void
+testCoinCompression:void
+testCoinVerification:void
+testBlacklistserver:void
+createCoin:Coin
+addType2:void
+addType1:void
privateKey:RSAPrivateKey
certificate:Certificate
whitelist:Whitelist
areAllMailsQueued:boolean
POP3ServerShouldWait:boolea
safe:SafeHandler
password:String
GUI
+AmountPanel
+Progress
+MainFrame
+RegistrationFrame
+OpenSafeDialog
+SplashScreen
+ConfigurationPanel
+SpamPanel
+FormPanel
ErrorDialog
+WithdrawPanel
+ImageButton
+DepositPanel
+Definitions
+WhitelistPanel
Proxy
+SMTPProxy
+POP3Proxy
+AbstractNonBlockingProxy
+NonBlockingProxyConnection
+AbstractBlockingProxy
+IMAPProxy
Safe
Safe
+SafeHandler
MailHandler
+MailFunctions
+OutgoingMailProcessor
+CoinHeader
+IncomingMailProcessor
+NotificationHeader
java.io.Serializable
Whitelist
-list:Vector
+Whitelist
+add:void
+remove:void
+onWhitelist:boolean
+toString:String
+main:void
data:Vector
CurrencyExchangeClient
-serverSession:session
-CESCertificate:Certificate
-BLSCertificate:Certificate
-username:String
-password:String
-className:Class
+CurrencyExchangeClient
+logonToCurrencyServer:boolean
-logon:StatusPacket
-checkSession:void
-createSession:void
+withdraw:Coin[]
+deposit:double
-necessaryCoins:int
-generateCoins:Coin[][]
+reportCoins:void
+send:void
+receive:NetPacket
+main:void

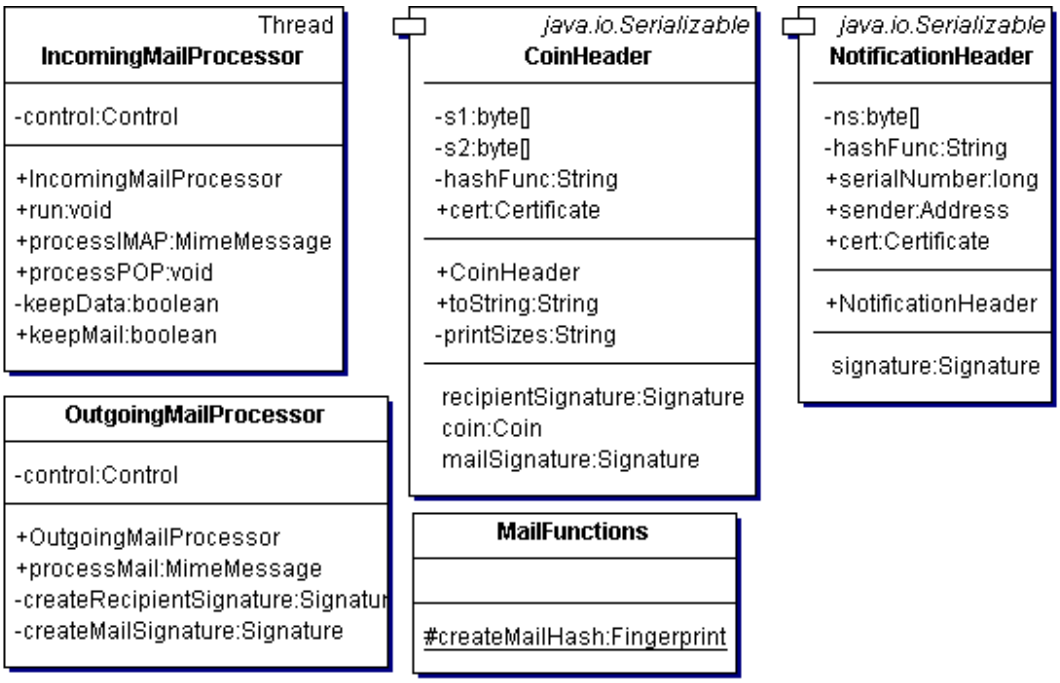
GUI


MetalLookAndFeel
Definitions
-largeSpace:int
-smallSpace:int
-backgroundColor:Color
-borderColor1:Color
-borderColor2:Color
-borderColor3:Color
-panelBackground:Color
-textColor:Color
-font1:FontUIResource
-textFieldBackgroundColor:Colo
-textFieldBorderSize:int
-textFieldForegroundColor:Color
-textFieldBorderColor:Color
-textFieldForegroundColorInactiv
-textFieldFont:FontUIResource
-comboBoxBackgroundColor:Co
-comboBoxForegroundColor:Co
-comboBoxSelectionBackgroun
-comboBoxSelectionForegroun
-textAreaForegroundColor:Color
+buttonTextColor:Color
-defaults:Object[]
-defaultXsize:int
-defaultYsize:int
#initClassDefaults:void
#initComponentDefaults:void
+main:void
name:String
description:String
nativeLookAndFeel:boolean
supportedLookAndFeel:boolean
ID:String
spaceS:int
spaceL:int
background:Color
textArea:JTextArea
JFrame
RegistrationFrame
-control:Control
-frame:MainFrame
-informationtext:String
-labels:String[]
-buttonTypeNames:String[]
-logonPanel:FormPanel
-informationTextArea:JTextArea
-errorDialog:ErrorDialog
+RegistrationFrame
-checkFields:String
#showError:void
-center:void
-backgroundPanel
-actionListener
-WindowIsClosing
JDialog
OpenSafeDialog
-frame:MainFrame
-informationtext:String
-labels:String[]
-buttonTypeNames:String[]
-logonPanel:FormPanel
-topPanel:TopPanel
-informationTextArea:JTextArea
-amount:double
-jobID:int
-CESLogonName:String
-CESpassword:String
-errorDialog:ErrorDialog
+DEPOSIT_ID:int
+WITHDRAW_ID:int
+SAFE_STATUS_ID:int
+UPDATE_ID:int
+CONTROL_THREAD:int
#OpenSafeDialog
+show:void
+show:void
-checkFields:String
#showError:void
-center:void
-backgroundPanel
-TopPanel
-actionListener
JDialog
ErrorDialog
-okButton:ImageButton
-info:JTextArea
ErrorDialog
ErrorDialog
-init:void
+showError:void
+showMessage:void
-center:void
-backgroundPanel
-listener
JPanel
ImageButton
-buttonTypeNames:String[]
-buttonLabel:JLabel
-buttonIcons:ImageIcon[]
-buttonName:String
-buttonText:String
-buttonWidth:int
-buttonHeight:int
-font:Font
-space:int
-percentOfHeight:double
-textColor:Color
-setTextCoords:boolean
-x:int
-y:int
-pressed:boolean
actionListeners:EventListenerLi
+ImageButton
+ImageButton
+ImageButton
+initImageButton:void
+paint:void
#fireAction:void
-buttonListener
action
JPanel
AmountPanel
-plusButton:ImageButton
-minusButton:ImageButton
-leftPanel:FormPanel
-rightPanel:JPanel
-plusMinusListener:PlusMinusL
-integer:boolean
-incrementValue:double
+AmountPanel
-updateField:String
-addIntValue:String
-addDoubleValue:String
-subIntValue:String
-subDoubleValue:String
-PlusMinusListener
-keyListener
-PlusMinusPanel
text:String
JWindow
Progress
-totalSize:double
-passed:double
-numberOfProgreesSteps:double
-currentStep:double
-progress:JProgressBar
-label1:JLabel
-label2:JLabel
-label3:JLabel
-print:String
-cornerImage:ImageIcon
+Progress
+init:void
+update:void
-position:void
+main:void
-doTestJob:void
-ProgressPanel
JPanel
FormPanel
-strings:String[]
-labels:JLabel[]
#textFields:JTextField[]
-panels:JPanel[]
+FormPanel
+clear:void
JWindow
SplashScreen
-splashImage:ImageIcon
-imageLabel:JLabel
+SplashScreen
-center:void

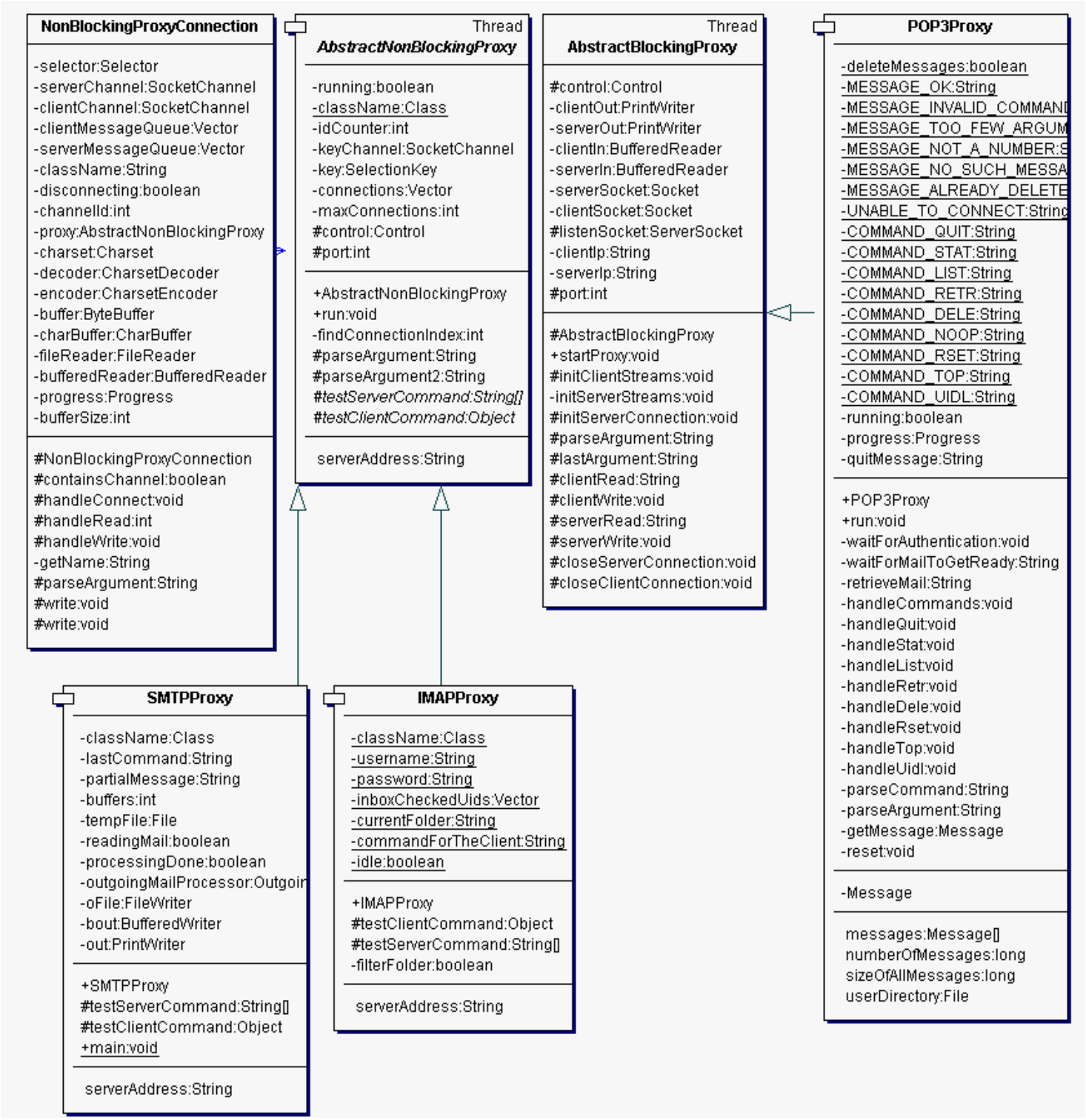
GUI (fortsat)


JFrame
MainFrame
+cursor:Cursor
+cornerImage:ImageIcon
-safeDialog:OpenSafeDialog
-configPanel:ConfigurationPane
-spamPanel:SpamPanel
-safeKeyFilePath:String
-safePassword:String
-safeInfoExists:boolean
-errorDialog:ErrorDialog
-registrationDialog:RegistrationF
-splash:SplashScreen
-control:Control
+MainFrame
+init:void
+showRegistrationDialog:void
+hideRegistrationDialog:void
+show:void
+showError:void
+showMessage:void
#collectCoin:void
#saveConfigurationInfo:void
#loadConfigurationInfo:ProxyCo
+updateMailList:void
+withdraw:void
#deposit:void
+tryToOpenSafe:void
#openSafe:void
#wakeUpControlThreads:void
#showStatus:void
-center:void
-shutdown:void
-buttonPanel
-changeListener
-listener
-WindowIsClosing
safeStatus:String
JPanel
ConfigurationPanel
-frame:MainFrame
-informationtext:String
-userLabels:String[]
-serverLabels:String[]
-logonLabels:String[]
-sectionLabels:String[]
-buttonTypeNames:String[]
-userInformationPanel:FormPanel
-serverInformationPanel:FormPane
-logonInformationPanel:FormPane
-informationTextArea:JTextArea
#ConfigurationPanel
-checkFields:String
-checkIP:boolean
-checkEmail:boolean
#setConfigInfo:void
-actionListener
JPanel
DepositPanel
-frame:MainFrame
-informationtext:String
-labels:String[]
-CESLabel:String
-logonInformationPanel:FormPane
-amountPanel:AmountPanel
-informationTextArea:JTextArea
-className:Class
#DepositPanel
-checkFields:String
-textFieldListener
JPanel
SpamPanel
-frame:MainFrame
-informationTextArea:JTextArea
-mailInformationTextArea:JTextArea
-informationtext:String
-list:JList
#SpamPanel
+update:void
#updateList:void
-SelectionPanel
-buttonPanel
-actionListener
-selectListener
JPanel
WhitelistPanel
-frame:MainFrame
-control:Control
-informationTextArea:JTextArea
-mailInformationTextArea:JTextArea
-informationtext:String
-addressTextField:JTextField
-list:JList
#WhitelistPanel
-SelectionPanel
-buttonPanel
-actionListener
JPanel
WithdrawPanel
-frame:MainFrame
-informationtext:String
-labels:String[]
-CESLabel:String
-logonInformationPanel:FormPane
-amountPanel:AmountPanel
-informationTextArea:JTextArea
-className:Class
#WithdrawPanel
-checkFields:String
-textFieldListener

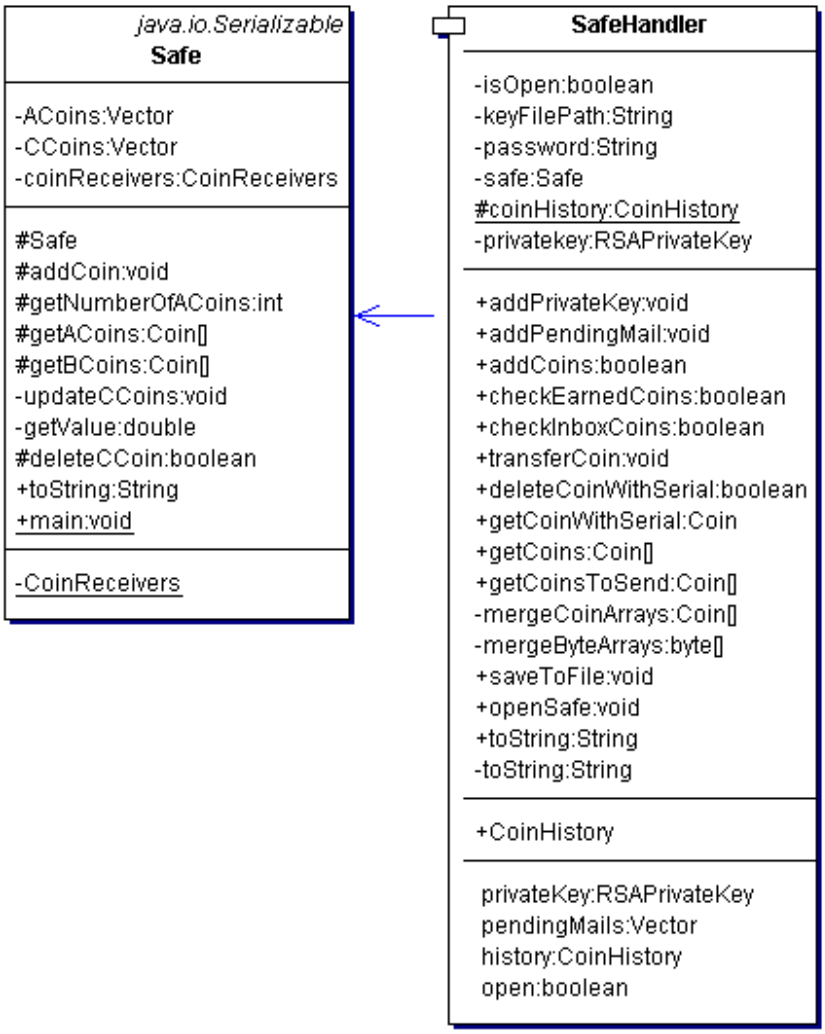
MailHandler



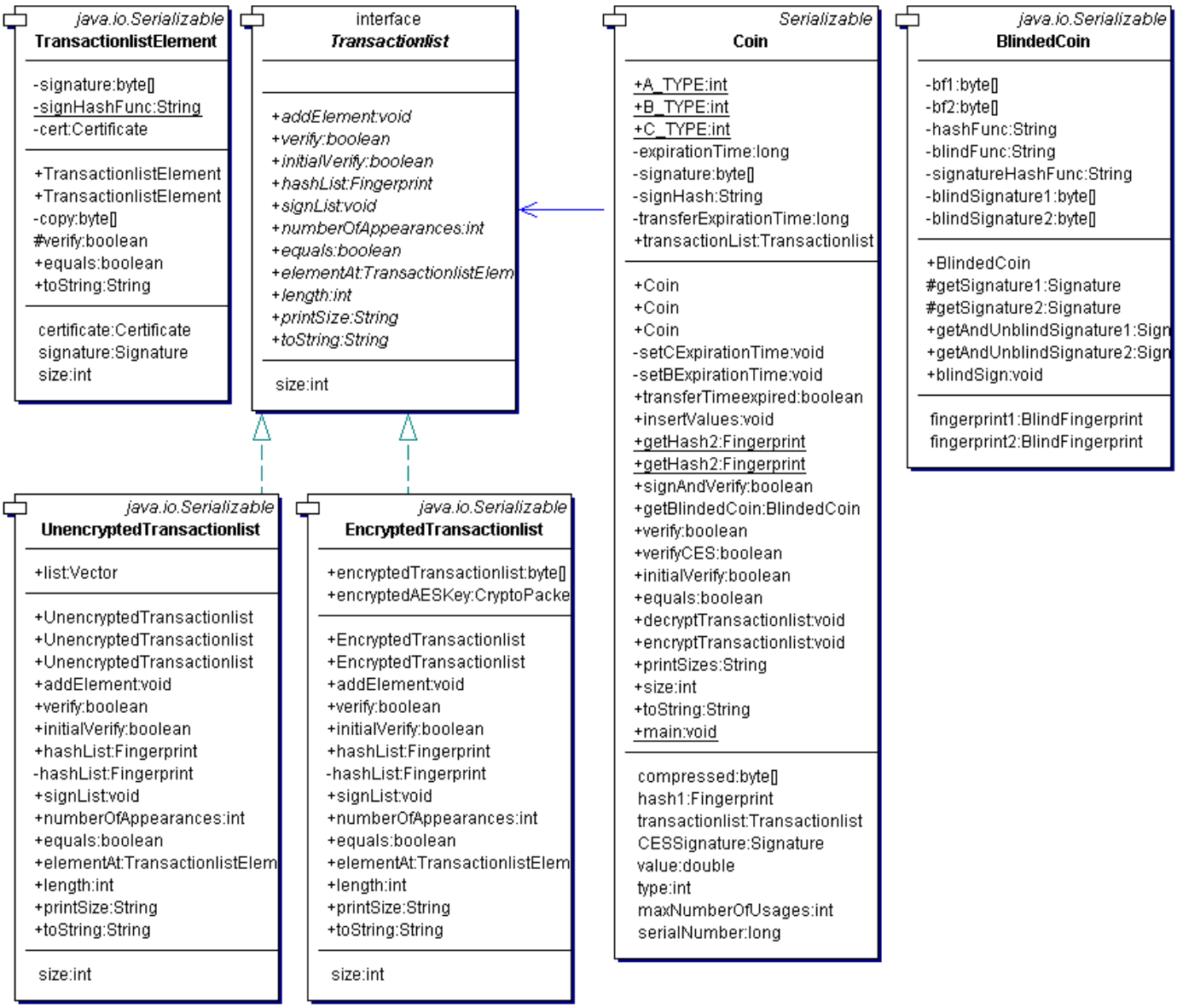
Proxy


NonBlockingProxyConnection
-selector:Selector
-serverChannel:SocketChannel
-clientChannel:SocketChannel
-clientMessageQueue:Vector
-serverMessageQueue:Vector
-className:String
-disconnecting:boolean
-channelId:int
-proxy:AbstractNonBlockingProxy
-charset:Charset
-decoder:CharsetDecoder
-encoder:CharsetEncoder
-buffer:ByteBuffer
-charBuffer:CharBuffer
-fileReader:FileReader
-bufferedReader:BufferedReader
-progress:Progress
-bufferSize:int
#NonBlockingProxyConnection
#containsChannel:boolean
#handleConnect:void
#handleRead:int
#handleWrite:void
-getName:String
#parseArgument:String
#write:void
#write:void
Thread
AbstractNonBlockingProxy
-running:boolean
-className:Class
-idCounter:int
-keyChannel:SocketChannel
-key:SelectionKey
-connections:Vector
-maxConnections:int
#control:Control
#port:int
+AbstractNonBlockingProxy
+run:void
-findConnectionIndex:int
#parseArgument:String
#parseArgument2:String
#testServerCommand:String[]
#testClientCommand:Object
serverAddress:String
Thread
AbstractBlockingProxy
#control:Control
-clientOut:PrintWriter
-serverOut:PrintWriter
-clientIn:BufferedReader
-serverIn:BufferedReader
-serverSocket:Socket
-clientSocket:Socket
#listenSocket:ServerSocket
-clientIp:String
-serverIp:String
#port:int
#AbstractBlockingProxy
+startProxy:void
#initClientStreams:void
-initServerStreams:void
#initServerConnection:void
#parseArgument:String
#lastArgument:String
#clientRead:String
#clientWrite:void
#serverRead:String
#serverWrite:void
#closeServerConnection:void
#closeClientConnection:void
POP3Proxy
-deleteMessages:boolean
-MESSAGE_OK:String
-MESSAGE_INVALID_COMMAN
-MESSAGE_TOO_FEW_ARGUM
-MESSAGE_NOT_A_NUMBER:S
-MESSAGE_NO_SUCH_MESSA
-MESSAGE_ALREADY_DELETE
-UNABLE_TO_CONNECT:String
-COMMAND_QUIT:String
-COMMAND_STAT:String
-COMMAND_LIST:String
-COMMAND_RETR:String
-COMMAND_DELE:String
-COMMAND_NOOP:String
-COMMAND_RSET:String
-COMMAND_TOP:String
-COMMAND_UIDL:String
-running:boolean
-progress:Progress
-quitMessage:String
+POP3Proxy
+run:void
-waitForAuthentication:void
-waitForMailToGetReady:String
-retrieveMail:String
-handleCommands:void
-handleQuit:void
-handleStat:void
-handleList:void
-handleRetr:void
-handleDele:void
-handleRset:void
-handleTop:void
-handleUidl:void
-parseCommand:String
-parseArgument:String
-getMessage:Message
-reset:void
-Message
messages:Message[]
numberOfMessages:long
sizeOfAllMessages:long
userDirectory:File
SMTPProxy
-className:Class
-lastCommand:String
-partialMessage:String
-buffers:int
-tempFile:File
-readingMail:boolean
-processingDone:boolean
-outgoingMailProcessor:Outgoi
-oFile:FileWriter
-bout:BufferedWriter
-out:PrintWriter
+SMTPProxy
#testServerCommand:String[]
#testClientCommand:Object
+main:void
serverAddress:String
IMAPProxy
-className:Class
-username:String
-password:String
-inboxCheckedUids:Vector
-currentFolder:String
-commandForTheClient:String
-idle:boolean
+IMAPProxy
#testClientCommand:Object
#testServerCommand:String[]
-filterFolder:boolean
serverAddress:String

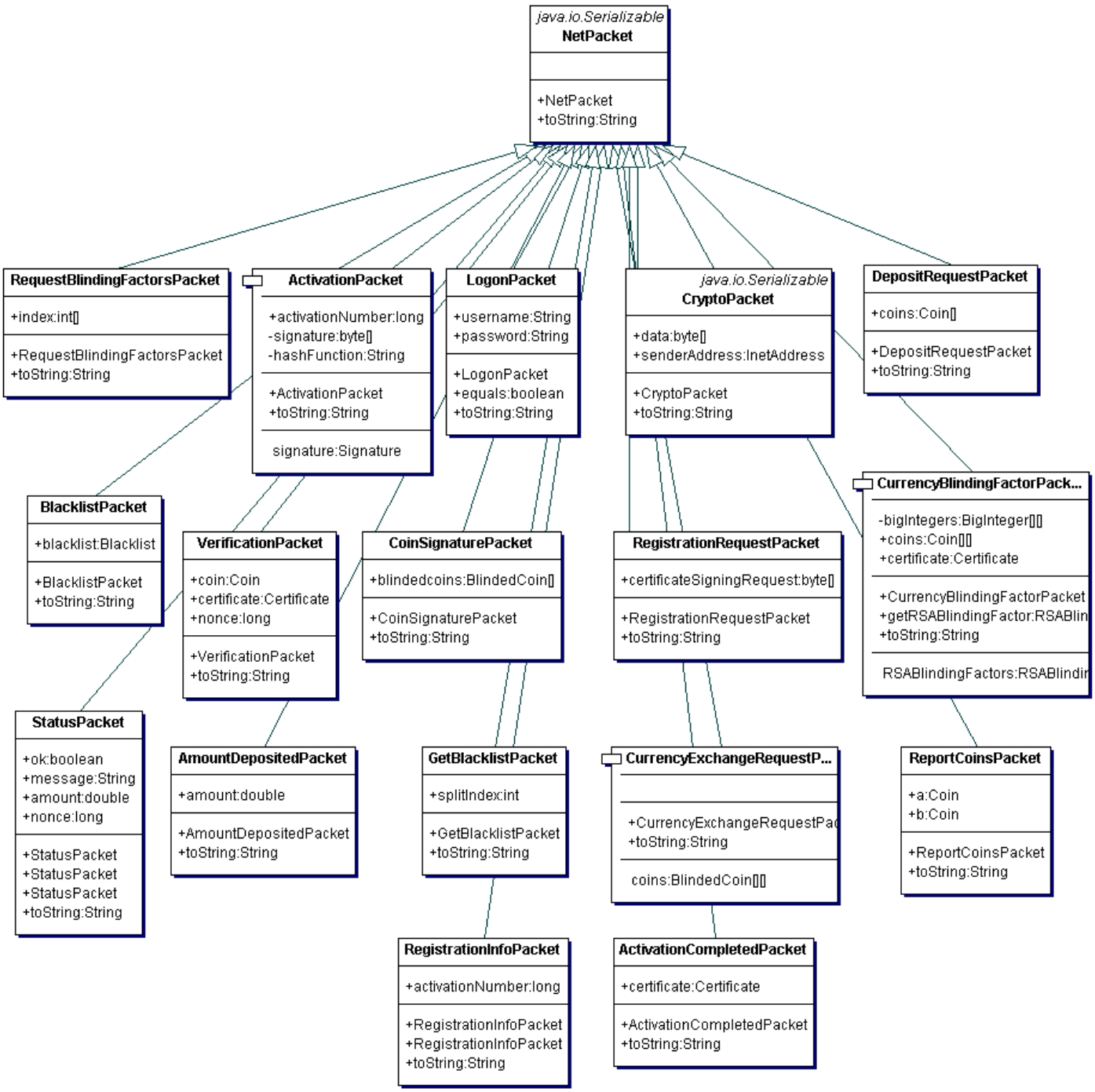
Safe


java.io.Serializable
Safe
-ACoins:Vector
-CCoins:Vector
-coinReceivers:CoinReceivers
#Safe
#addCoin:void
#getNumberOfACoins:int
#getACoins:Coin[]
#getBCoins:Coin[]
-updateCCoins:void
-getValue:double
#deleteCCoin:boolean
+toString:String
+main:void
-CoinReceivers
SafeHandler
-isOpen:boolean
-keyFilePath:String
-password:String
-safe:Safe
#coinHistory:CoinHistory
-privatekey:RSAPrivateKey
+addPrivateKey:void
+addPendingMail:void
+addCoins:boolean
+checkEarnedCoins:boolean
+checkInboxCoins:boolean
+transferCoin:void
+deleteCoinWithSerial:boolean
+getCoinWithSerial:Coin
+getCoins:Coin[]
+getCoinsToSend:Coin[]
-mergeCoinArrays:Coin[]
-mergeByteArrays:byte[]
+saveToFile:void
+openSafe:void
+toString:String
-toString:String
+CoinHistory
privateKey:RSAPrivateKey
pendingMails:Vector
history:CoinHistory
open:boolean

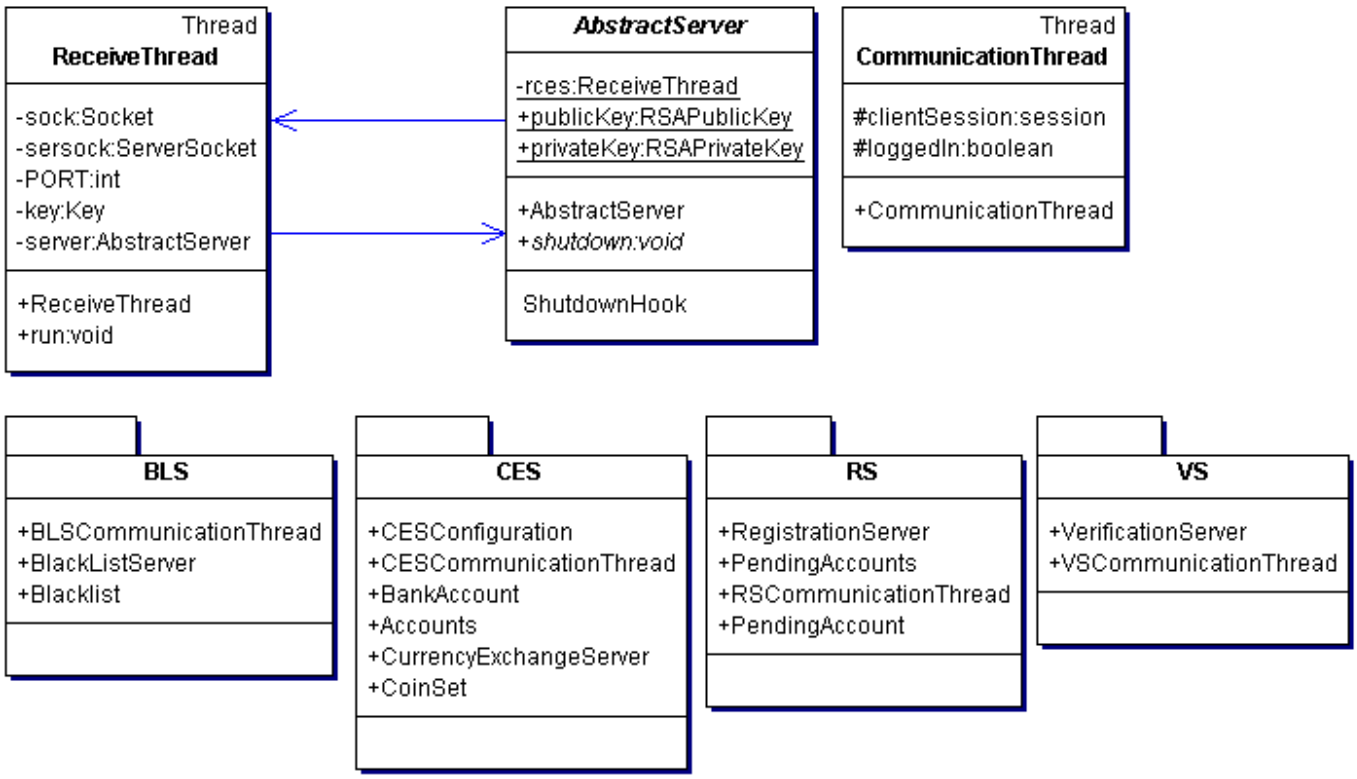
Currency


java.io.Serializable
TransactionlistElement
-signature:byte[]
-signHashFunc:String
-cert:Certificate
+TransactionlistElement
+TransactionlistElement
-copy:byte[]
#verify:boolean
+equals:boolean
+toString:String
certificate:Certificate
signature:Signature
size:int
interface
Transactionlist
+addElement:void
+verify:boolean
+initialVerify:boolean
+hashList:Fingerprint
+signList:void
+numberOfAppearances:int
+equals:boolean
+elementAt:TransactionlistElem
+length:int
+printSize:String
+toString:String
size:int
Serializable
Coin
+A_TYPE:int
+B_TYPE:int
+C_TYPE:int
-expirationTime:long
-signature:byte[]
-signHash:String
-transferExpirationTime:long
+transactionList:Transactionlist
+Coin
+Coin
+Coin
-setCExpirationTime:void
-setBExpirationTime:void
+transferTimeexpired:boolean
+insertValues:void
+getHash2:Fingerprint
+getHash2:Fingerprint
+signAndVerify:boolean
+getBlindedCoin:BlindedCoin
+verify:boolean
+verifyCES:boolean
+initialVerify:boolean
+equals:boolean
+decryptTransactionlist:void
+encryptTransactionlist:void
+printSizes:String
+size:int
+toString:String
+main:void
compressed:byte[]
hash1:Fingerprint
transactionlist:Transactionlist
CESSignature:Signature
value:double
type:int
maxNumberOfUsages:int
serialNumber:long
java.io.Serializable
BlindedCoin
-bf1:byte[]
-bf2:byte[]
-hashFunc:String
-blindFunc:String
-signatureHashFunc:String
-blindSignature1:byte[]
-blindSignature2:byte[]
+BlindedCoin
#getSignature1:Signature
#getSignature2:Signature
+getAndUnblindSignature1:Sign
+getAndUnblindSignature2:Sign
+blindSign:void
fingerprint1:BlindFingerprint
fingerprint2:BlindFingerprint
java.io.Serializable
UnencryptedTransactionlist
+list:Vector
+UnencryptedTransactionlist
+UnencryptedTransactionlist
+UnencryptedTransactionlist
+addElement:void
+verify:boolean
+initialVerify:boolean
+hashList:Fingerprint
-hashList:Fingerprint
+signList:void
+numberOfAppearances:int
+equals:boolean
+elementAt:TransactionlistElem
+length:int
+printSize:String
+toString:String
size:int
java.io.Serializable
EncryptedTransactionlist
+encryptedTransactionlist:byte[]
+encryptedAESKey:CryptoPacke
+EncryptedTransactionlist
+EncryptedTransactionlist
+addElement:void
+verify:boolean
+initialVerify:boolean
+hashList:Fingerprint
-hashList:Fingerprint
+signList:void
+numberOfAppearances:int
+equals:boolean
+elementAt:TransactionlistElem
+length:int
+printSize:String
+toString:String
size:int

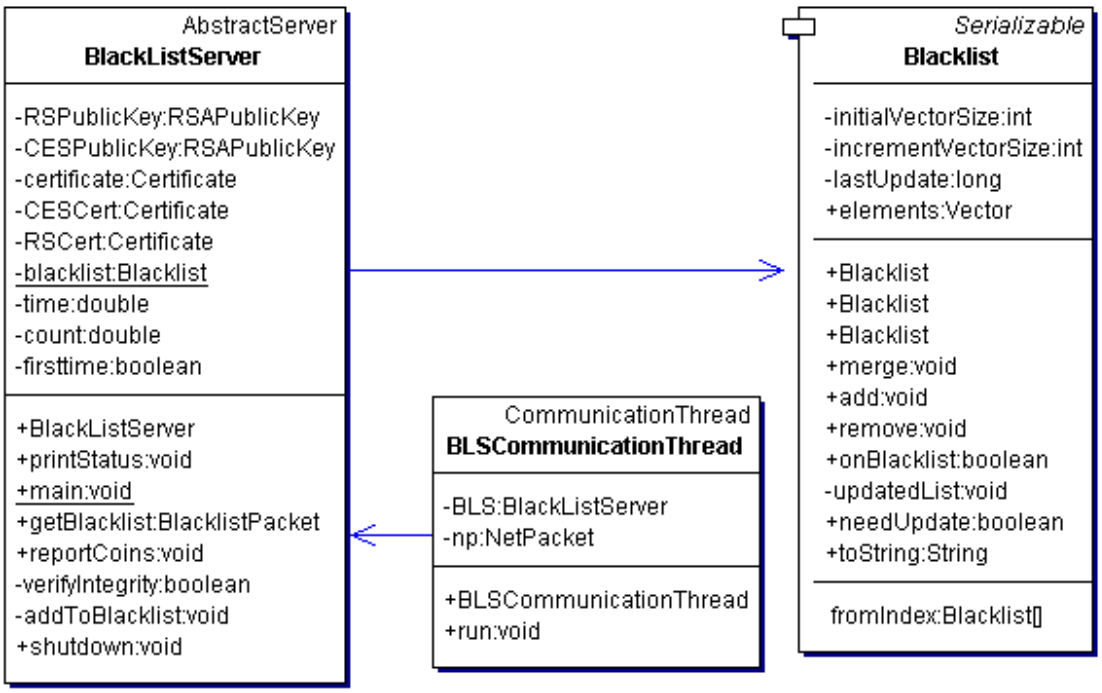
NetworkPackets



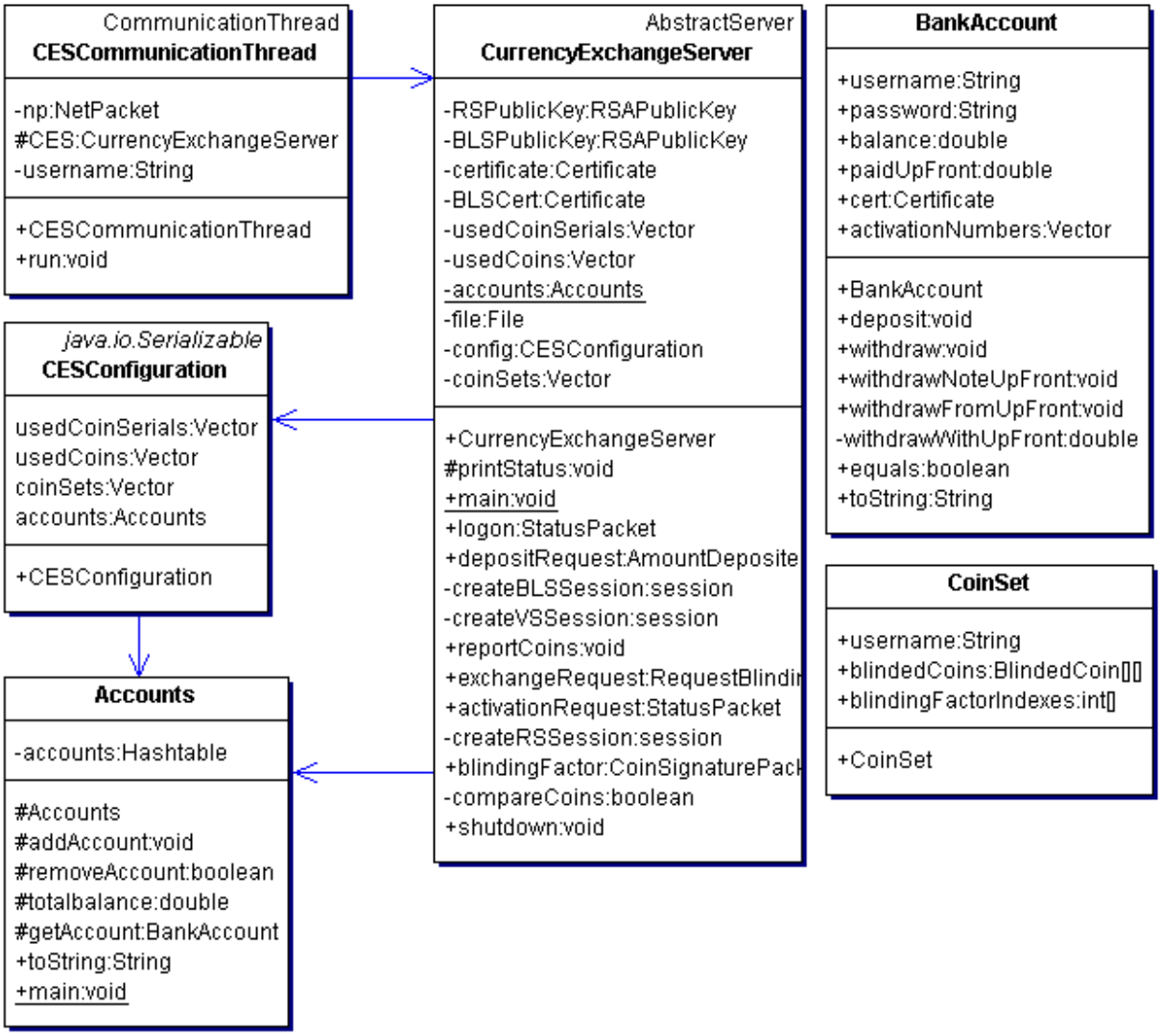
Servers


Thread
ReceiveThread
-sock:Socket
-sersock:ServerSocket
-PORT:int
-key:Key
-server:AbstractServer
+ReceiveThread
+run:void
AbstractServer
-rces:ReceiveThread
+publicKey:RSAPublicKey
+privateKey:RSAPrivateKey
+AbstractServer
+shutdown:void
ShutdownHook
Thread
CommunicationThread
#clientSession:session
#loggedIn:boolean
+CommunicationThread
BLS
+BLSCommunicationThread
+BlackListServer
+Blacklist
CES
+CESConfiguration
+CESCommunicationThread
+BankAccount
+Accounts
+CurrencyExchangeServer
+CoinSet
RS
+RegistrationServer
+PendingAccounts
+RSCommunicationThread
+PendingAccount
VS
+VerificationServer
+VSCommunicationThread

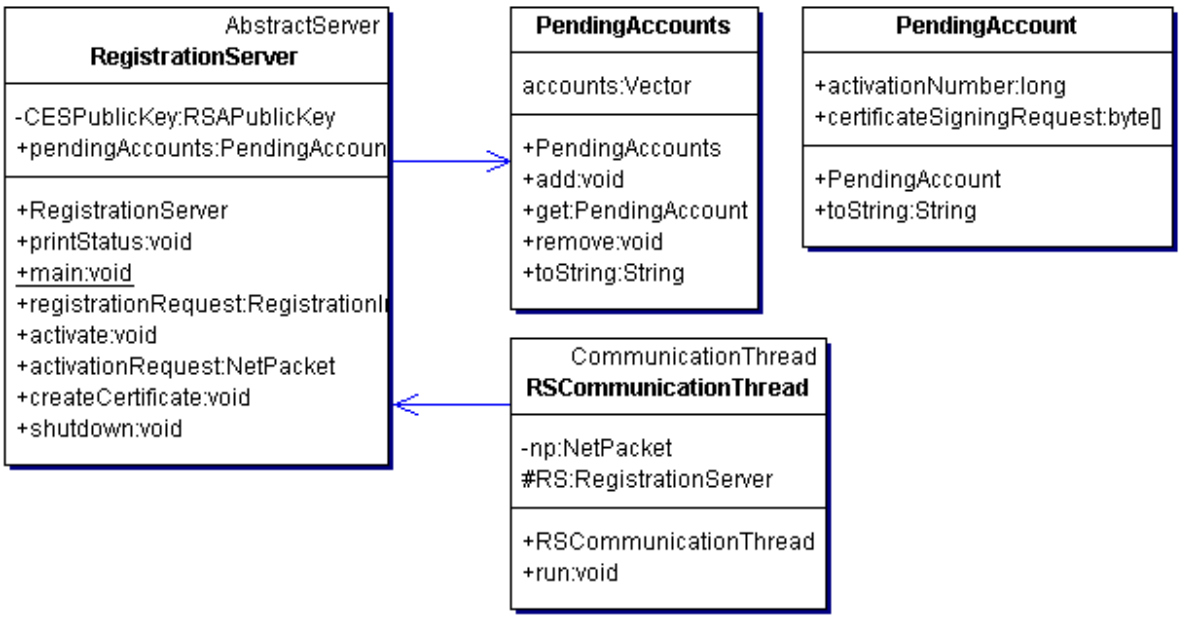
BLS


AbstractServer
BlackListServer
-RSPublicKey:RSAPublicKey
-CESPublicKey:RSAPublicKey
-certificate:Certificate
-CESCert:Certificate
-RSCert:Certificate
-blacklist:Blacklist
-time:double
-count:double
-firsttime:boolean
+BlackListServer
+printStatus:void
+main:void
+getBlacklist:BlacklistPacket
+reportCoins:void
-verifyIntegrity:boolean
-addToBlacklist:void
+shutdown:void
Serializable
Blacklist
-initialVectorSize:int
-incrementVectorSize:int
-lastUpdate:long
+elements:Vector
+Blacklist
+Blacklist
+Blacklist
+merge:void
+add:void
+remove:void
+onBlacklist:boolean
-updatedList:void
+needUpdate:boolean
+toString:String
fromIndex:Blacklist[]
CommunicationThread
BLSCommunicationThread
-BLS:BlackListServer
-np:NetPacket
+BLSCommunicationThread
+run:void

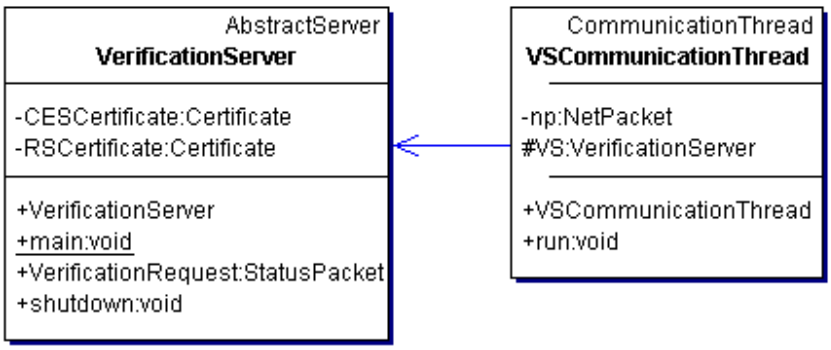
CES


CommunicationThread
CESCommunicationThread
-np:NetPacket
#CES:CurrencyExchangeServer
-username:String
+CESCommunicationThread
+run:void
AbstractServer
CurrencyExchangeServer
-RSPublicKey:RSAPublicKey
-BLSPublicKey:RSAPublicKey
-certificate:Certificate
-BLSCert:Certificate
-usedCoinSerials:Vector
-usedCoins:Vector
-accounts:Accounts
-file:File
-config:CESConfiguration
-coinSets:Vector
+CurrencyExchangeServer
#printStatus:void
+main:void
+logon:StatusPacket
+depositRequest:AmountDeposite
-createBLSSession:session
-createVSSession:session
+reportCoins:void
+exchangeRequest:RequestBlindi
+activationRequest:StatusPacket
-createRSSession:session
+blindingFactor:CoinSignaturePac
-compareCoins:boolean
+shutdown:void
BankAccount
+username:String
+password:String
+balance:double
+paidUpFront:double
+cert:Certificate
+activationNumbers:Vector
+BankAccount
+deposit:void
+withdraw:void
+withdrawNoteUpFront:void
+withdrawFromUpFront:void
-withdrawWithUpFront:double
+equals:boolean
+toString:String
java.io.Serializable
CESConfiguration
usedCoinSerials:Vector
usedCoins:Vector
coinSets:Vector
accounts:Accounts
+CESConfiguration
CoinSet
+username:String
+blindedCoins:BlindedCoin[][]
+blindingFactorIndexes:int[]
+CoinSet
Accounts
-accounts:Hashtable
#Accounts
#addAccount:void
#removeAccount:boolean
#totalbalance:double
#getAccount:BankAccount
+toString:String
+main:void

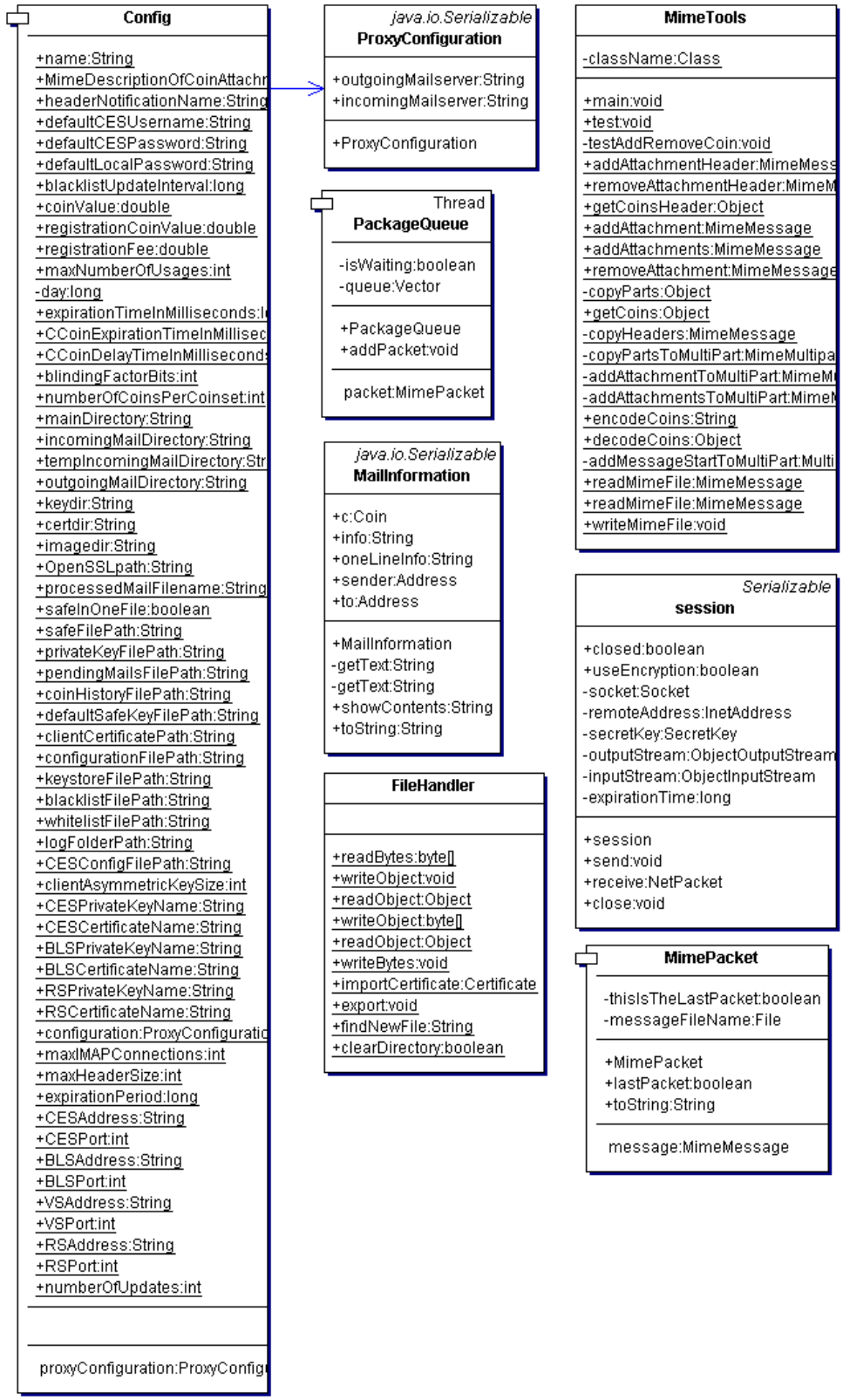
RS


AbstractServer
RegistrationServer
-CESPublicKey:RSAPublicKey
+pendingAccounts:PendingAccoun
+RegistrationServer
+printStatus:void
+main:void
+registrationRequest:RegistrationI
+activate:void
+activationRequest:NetPacket
+createCertificate:void
+shutdown:void
PendingAccounts
accounts:Vector
+PendingAccounts
+add:void
+get:PendingAccount
+remove:void
+toString:String
PendingAccount
+activationNumber:long
+certificateSigningRequest:byte[]
+PendingAccount
+toString:String
CommunicationThread
RSCommunicationThread
-np:NetPacket
#RS:RegistrationServer
+RSCommunicationThread
+run:void

VS


AbstractServer
VerificationServer
-CESCertificate:Certificate
-RSCertificate:Certificate
+VerificationServer
+main:void
+VerificationRequest:StatusPacket
+shutdown:void
CommunicationThread
VSCommunicationThread
-np:NetPacket
#VS:VerificationServer
+VSCommunicationThread
+run:void

Utilities


Config
+name:String
+MimeDescriptionOfCoinAttachr
+headerNotificationName:String
+defaultCESUsername:String
+defaultCESPassword:String
+defaultLocalPassword:String
+blacklistUpdateInterval:long
+coinValue:double
+registrationCoinValue:double
+registrationFee:double
+maxNumberOfUsages:int
-day:long
+expirationTimeInMilliseconds:l
+CCoinExpirationTimeInMillisec
+CCoinDelayTimeInMilliseconds
+blindingFactorBits:int
+numberOfCoinsPerCoinset:int
+mainDirectory:String
+incomingMailDirectory:String
+tempIncomingMailDirectory:Str
+outgoingMailDirectory:String
+keydir:String
+certdir:String
+imagedir:String
+OpenSSLpath:String
+processedMailFilename:String
+safeInOneFile:boolean
+safeFilePath:String
+privateKeyFilePath:String
+pendingMailsFilePath:String
+coinHistoryFilePath:String
+defaultSafeKeyFilePath:String
+clientCertificatePath:String
+configurationFilePath:String
+keystoreFilePath:String
+blacklistFilePath:String
+whitelistFilePath:String
+logFolderPath:String
+CESConfigFilePath:String
+clientAsymmetricKeySize:int
+CESPrivateKeyName:String
+CESCertificateName:String
+BLSPrivateKeyName:String
+BLSCertificateName:String
+RSPrivateKeyName:String
+RSCertificateName:String
+configuration:ProxyConfiguratio
+maxIMAPConnections:int
+maxHeaderSize:int
+expirationPeriod:long
+CESAddress:String
+CESPort:int
+BLSAddress:String
+BLSPort:int
+VSAddress:String
+VSPort:int
+RSAddress:String
+RSPort:int
+numberOfUpdates:int
proxyConfiguration:ProxyConfigu
java.io.Serializable
ProxyConfiguration
+outgoingMailserver:String
+incomingMailserver:String
+ProxyConfiguration
Thread
PackageQueue
-isWaiting:boolean
-queue:Vector
+PackageQueue
+addPacket:void
packet:MimePacket
java.io.Serializable
MailInformation
+c:Coin
+info:String
+oneLineInfo:String
+sender:Address
+to:Address
+MailInformation
-getText:String
-getText:String
+showContents:String
+toString:String
FileHandler
+readBytes:byte[]
+writeObject:void
+readObject:Object
+writeObject:byte[]
+readObject:Object
+writeBytes:void
+importCertificate:Certificate
+export:void
+findNewFile:String
+clearDirectory:boolean
MimeTools
-className:Class
+main:void
+test:void
-testAddRemoveCoin:void
+addAttachmentHeader:MimeMess
+removeAttachmentHeader:MimeM
+getCoinsHeader:Object
+addAttachment:MimeMessage
+addAttachments:MimeMessage
+removeAttachment:MimeMessage
-copyParts:Object
+getCoins:Object
-copyHeaders:MimeMessage
-copyPartsToMultiPart:MimeMultipa
-addAttachmentToMultiPart:MimeM
-addAttachmentsToMultiPart:MimeM
+encodeCoins:String
+decodeCoins:Object
-addMessageStartToMultiPart:Multi
+readMimeFile:MimeMessage
+readMimeFile:MimeMessage
+writeMimeFile:void
Serializable
session
+closed:boolean
+useEncryption:boolean
-socket:Socket
-remoteAddress:InetAddress
-secretKey:SecretKey
-outputStream:ObjectOutputStream
-inputStream:ObjectInputStream
-expirationTime:long
+session
+send:void
+receive:NetPacket
+close:void
MimePacket
-thisIsTheLastPacket:boolean
-messageFileName:File
+MimePacket
+lastPacket:boolean
+toString:String
message:MimeMessage

Utilities (fortsat)

Appendiks E Logfil

Herunder ses et eksempel på indholdet af logfilen `Client.log`, der dannes i klienten. Dette er medtaget for at demonstrere hvordan der kan dannes overblik over programmets opførsel og tilstand.

```
Control (1): Configuration file could not be loaded.
POP3Proxy (1): Proxy is initializing...
POP3Proxy (1): Proxy is ready on port 110
IMAPProxy (1): Proxy is ready on port 143
SMTPProxy (1): Proxy is ready on port 25
Control (1): Loaded server certificates succesfully.
Control (1): Certificate could not be loaded or safe does not exist. Probably
first program launch.
Control (1): Blacklist could not be loaded. Creating new.
Control (1): Whitelist could not be loaded. Creating new.
Control (1): Starting registration procedure...
SafeHandler (1): Loading data\Safe.sc using password: password and key-file-
path: data\Safe-key.key
SafeHandler (1): The safefiles could not be found. Creating new safe file.
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
Control (1): Transaction successful. Withdrawed 2 coins (worth 0.1) from your
account.
Control (1): Test of the received certificate with the privatkey...passed
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
Control (1): Transaction successful. Withdrawed 5 coins (worth 0.25) from your
account.
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
Control (1): Transaction successful. Deposited 2 coins (worth 0.1) to your
account.
SMTPProxy (1): Sending mail...
OutgoingMailProcessor (1): Processing mail to: 5503tv@abenghaven.dk
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
```

```
SafeHandler (1): Safe stored succesfully in data\Safe.sc
SMTPProxy (1): Mail Sent succesfully.
SMTPProxy (1): Sending mail...
OutgoingMailProcessor (1): Processing mail to: lerkenfeld@gmail.com
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
SMTPProxy (1): Mail Sent succesfully.
POP3Proxy (1): Connecting to server enghaven1:110
IncomingMailProcessor (1): Testing if it is a notificationmail....
IncomingMailProcessor (1): It is NOT a notificationmail.
IncomingMailProcessor (1): Testing message from: "Lasse Lerkenfeld Jensen"
<5503tv@abenghaven.dk>...
Control (1): Updating blacklist....
Control (1): Blacklist updated succesfully. New list:
Blacklist:

SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
IncomingMailProcessor (1): Message passed all verifications. Processing of
message comleted.
IncomingMailProcessor (1): Testing if it is a notificationmail....
IncomingMailProcessor (1): It is NOT a notificationmail.
IncomingMailProcessor (1): Testing message from: "Lasse Lerkenfeld Jensen"
<5503tv@abenghaven.dk>...
Control (1): Updating blacklist....
Blacklist (1): No need to update the blacklist until Wed Mar 09 15:43:59 CET
2005.
Control (1): No need to update the blacklist.
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
IncomingMailProcessor (1): Message passed all verifications. Processing of
message comleted.
POP3Proxy (1): Mail retrieved successfully.
Control (1): Shutting down...
Control (1): Writing safe to disk...
SafeHandler (1): Saving Safe to data\Safe.sc using password: password and key-
file-path: data\Safe-key.key
SafeHandler (1): Safe stored succesfully in data\Safe.sc
Control (1): Data was written successfully.
```

Appendiks F Maple kode

Herunder ses Maple-koden der benyttes til at vælge stikprøvefrekvensen i afsnit 8.2.2.a:

```
>
```

Herunder ses Maple-koden, der benyttes til at vælge stikprøvefrekvensen i afsnit 9.2.2.a:

```
> restart:
```

Vi vil først opstille sandsynligheden for, at en kopist blacklistes fordi denne har sendt med en kopieret mønt. Vi antager at kopister kun fældes, ved at klienter indrapporterer tilfældigt udvalgte modtagne mønter. I det følgende ses det altså bort fra, at kopister kan fældes når mønter forsøges indløst (vekslet til 'rigtige' penge).
Når (minimum) to kopier af samme mønt indrapporteres vil blacklisting finde sted. Målet i det følgende er således at bestemme sandsynligheden for at mindst to kopier indrapporteres. I det følgende angiver variablen **x** antallet af afsendte kopier med den pågæ}ldende mønt, mens **f** angiver den frekvens hvormed klienterne indrapporterer mønter. Hvis f.eks. f = 1/100 vil det altså betyde at klienter vil indrapportere en tilfældig modtaget mønt med sandsynligheden 1/100 = 0.01. Sandsynligheden for at en kopist kan afsende **x** kopierede mønter **uden** at indrapportering finder sted hos klienter er dermed givet ved:

```
> p1:=(1-f)^x;
```

$$p1 := (1 - f)^{x}$$

Sandsynligheden for at en kopist kan afsende **x** kopierede mønter og **netop** én indrapportering finder sted hos klienter er givet ved

```
> p2:=x*(1-f)^(x-1)*f;
```

$$p2 := x\,(1 - f)^{(x - 1)}\,f$$

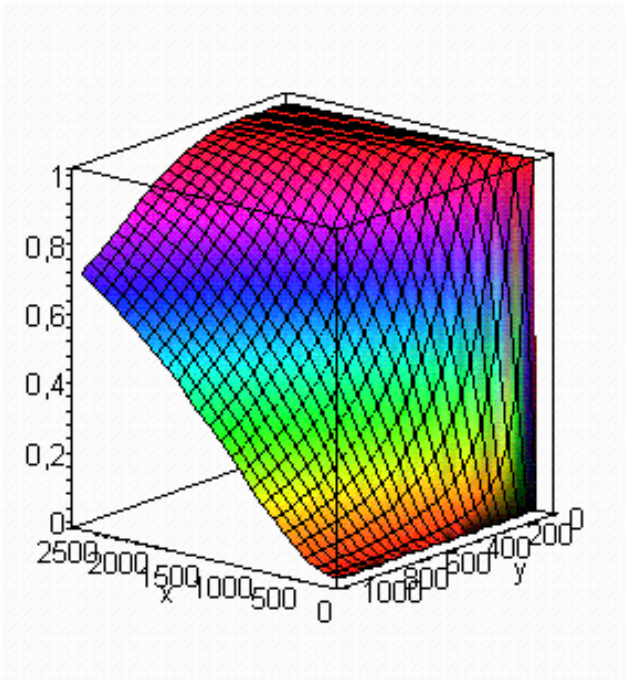
Sandsynligheden for at en kopist kan afsende **x** kopierede mønter og **mindst to** indrapporteringer finder sted hos klienter kan nu let beregnes:

```
> p3:=1-(p1+p2);
```

$$p3 := 1 - (1 - f)^{x} - x\,(1 - f)^{(x - 1)}\,f$$

p3 er således en funktion af to variable, nemlig antallet af afsendte kopier **x** samt stikprøvefrekvensen **f**, hvis output er sandsynligheden for at en kopist detekteres. Følgende 3D-plot viser funktionens output for værdier af x i intervallet [1;2500] og frekvenser [1/1000;1], hvilket kan benyttes til at bestemme en egnet stikprøvefrekvens for systemet:

```
> f:=1/y:
>
plot3d(p3,x=1..2500,y=1..1000,axes=boxed,style=patch,color=
p3);
```


1
0,8
0,6
0,4
0,2
0
2500
2000
1500
1000
500
0
x
1000
800
600
400
200
0
y

>

Appendiks G Matlab kode

I dette appendiks er findes MATLAB-koden der benyttes til kryptere/dekryptere billeder i afsnit 4.2.

```
close all;
clear all;

%data = 0:15
data = imread('børn.tif');
imshow(data);
sizes = size(data);
data = double(data(:)') + 1;

%imshow(uint8(reshape(data2 - 1,sizes(1),sizes(2))));

%the key
subst_key = randperm(256)
perm_key = randperm(58)

y_subst = encrypt_subst(data,subst_key);
data2 = decrypt_subst(y_subst,subst_key);
sum(data - data2)
figure;
title('Substitution')
imshow(uint8(reshape(y_subst - 1,sizes(1),sizes(2))));

y_perm = encrypt_perm(data,perm_key);
data2 = decrypt_perm(y_perm,perm_key);
min(data2)
sum(data - data2);
figure;
title('Permutation')
imshow(uint8(reshape(y_perm - 1,sizes(1),sizes(2))));

encrypted = encrypt_subst(mod(y_perm+1,256),subst_key);
figure;
title('Both')
imshow(uint8(reshape(encrypted - 1,sizes(1),sizes(2))));

end

function y = encrypt_subst(x,k)
    y = zeros(size(x,1),size(x,2));
    for i=1:size(x,2)
        y(i) = k(x(i));
    end
end

function y = decrypt_subst(x,k)
    y = zeros(size(x,1),size(x,2));
    for i=1:size(x,2)
        y(i) = find(x(i) == k);
    end
end

function y = encrypt_perm(x,k)
    y = zeros(size(x,1),size(x,2));
    prev = 1;
    for i=size(k,2):size(k,2):size(x,2)-1
        y(prev:i) = perm(x(prev:i),k);
```

```
        prev = i+1;
    end
end

function y = decrypt_perm(x,k)
    %calculate inverse permutation key
    k2 = [k;1:size(k,2)]';
    k2 = sortrows(k2);
    k = k2(:,2)'

    %permute
    y = zeros(size(x,1),size(x,2));
    prev = 1;
    for i=size(k,2):size(k,2):size(x,2)-1
        y(prev:i) = perm(x(prev:i),k);
        prev = i+1;
    end
end

function y = perm(x,k)
   y = zeros(size(x,1),size(x,2));
   for i=1:size(x,2)
      y(k(i)) = x(i);
    end
end
```

Appendiks H JAVA Kode

I dette bilag forefindes JAVA-kildekoden til programmet. Programmet består i alt af 14 pakker og 85 klasser, der er ordnet på følgende måde. Pakkerne i programmet ses i nedenstående punktopstilling, hvor indrykning angiver underpakker:

- SpamCash
  - Client
    - GUI
    - MailHandler
    - Proxy
    - Safe
  - Currency
  - NetworkPackets
  - Servers
    - BLS
    - CES
    - RS
    - VS
  - Utilities

Herunder ses en indholdsfortegnelse for dette appendiks. Hver pakke og klasse er tilknyttet en overskrift. Pakkerne er vist i alfabetisk rækkefølge. I hver pakke er først vist klasserne og dernæst indre pakker i alfabetisk rækkefølge.

10 Kildekode 199
10.1 Client 199
10.1.1 Control 199
10.1.2 CurrencyExchangeClient 214
10.1.3 Whitelist 217
10.1.4 GUI 218
10.1.4.a AmountPanel 218
10.1.4.b ConfigurationPanel 221
10.1.4.c Definitions 223
10.1.4.d DepositPanel 235
10.1.4.e ErrorDialog 237
10.1.4.f FormPanel 238
10.1.4.g ImageButton 239
10.1.4.h MainFrame 242
10.1.4.i OpenSafeDialog 247
10.1.4.j Progress 251
10.1.4.k RegistrationFrame 253
10.1.4.l SpamPanel 256
10.1.4.m SplashScreen 258
10.1.4.n WhitelistPanel 259
10.1.4.o WithdrawPanel 261

10.1.5 MailHandler 262
10.1.5.a CoinHeader 262
10.1.5.b IncomingMailProcessor 263
10.1.5.c MailFunctions 267
10.1.5.d NotificationHeader 268
10.1.5.e OutgoingMailProcessor 268
10.1.6 Proxy 270
10.1.6.a AbstractBlokingProxy 270
10.1.6.b AbstractNonBlokingProxy 273
10.1.6.c IMAPProxy 276
10.1.6.d NonBlokingProxyConnection 278
10.1.6.e POP3Proxy 283
10.1.6.f SMTPProxy 295
10.1.7 Safe 297
10.1.7.a Safe 297
10.1.7.b SafeHandler 302
10.2 Currency 309
10.2.1 BlindedCoin 309
10.2.2 Coin 311
10.2.3 EncryptedTransactionlist 316
10.2.4 Transactionlist 318
10.2.5 TransactionlistElement 318
10.2.6 UnencryptedTransactionlist 320
10.3 NetworkPackets 323
10.3.1 ActivationCompletedPacket 323
10.3.2 ActivationPacket 324
10.3.3 AmountDepositedPacket 324
10.3.4 BlacklistPacket 325
10.3.5 CoinSignaturePacket 325
10.3.6 CryptoPacket 325
10.3.7 CurrencyBlindingFactorPacket 326
10.3.8 CurrencyExchangeRequestPacket 327
10.3.9 DepositRequestPacket 328
10.3.10 GetBlacklistPacket 328
10.3.11 LogonPacket 329
10.3.12 NetPacket 329
10.3.13 RegistrationInfoPacket 329
10.3.14 RegistrationRequestPacket 330
10.3.15 ReportCoinsPacket 330
10.3.16 RequestBlindingFactorsPacket 330
10.3.17 StatusPacket 331
10.3.18 VerificationPacket 331
10.4 Servers 333
10.4.1 AbstractServer 333
10.4.2 CommunicationThread 334
10.4.3 ReceiveThread 334

10.4.4 BLS 335
10.4.4.a Blacklist 335
10.4.4.b BlacklistServer 337
10.4.4.c BLSCommunicationThread 340
10.4.5 CES 341
10.4.5.a Accounts 341
10.4.5.b BankAccount 343
10.4.5.c CESCommunicationThread 344
10.4.5.d CESConfiguration 346
10.4.5.e CoinSet 346
10.4.5.f CurrencyExchangeServer 346
10.4.6 RS 353
10.4.6.a PendingAccount 353
10.4.6.b PendingAccounts 353
10.4.6.c RegistrationServer 354
10.4.6.d RSCommunicationThread 357
10.4.7 VS 360
10.4.7.a VerificationServer 360
10.4.7.b VSCommunicationThread 362
10.5 Utilities 363
10.5.1 Base64 363
10.5.2 Config 384
10.5.3 Cryptotools 386
10.5.4 Debugger 400
10.5.5 FileHandler 403
10.5.6 MailInformation 406
10.5.7 MimeTools 407
10.5.8 PackageQueue 414
10.5.9 ProxyConfiguration 415
10.5.10 Session 415
10.5.11 Test 418

10 Kildekode

10.1 Client

10.1.1 Control

```
package SpamCash.Client;


import java.util.Vector;
import java.net.*;
import java.security.cert.Certificate;
import java.security.interfaces.RSAPublicKey;
import java.security.Security;
import java.io.File;
import javax.crypto.SecretKey;
import java.security.KeyStore;
import java.io.FileInputStream;
import java.security.PrivateKey;
import java.util.Random;

import SpamCash.Currency.*;
import SpamCash.Client.Safe.*;
import SpamCash.Utilities.*;
import SpamCash.Client.Proxy.*;
import SpamCash.Client.MailHandler.*;
import SpamCash.NetworkPackets.*;
import SpamCash.Servers.BLS.Blacklist;
import SpamCash.Utilities.session;
import SpamCash.Client.GUI.MainFrame;
import java.security.interfaces.RSAPrivateKey;
import javax.mail.Address;
import java.security.cert.X509Certificate;
import javax.mail.internet.MimeMessage;
import java.util.Properties;
import javax.mail.Session;
import javax.mail.Message;
import javax.mail.Transport;
import java.util.GregorianCalendar;
import org.logi.crypto.sign.Fingerprint;
import org.logi.crypto.sign.Signature;
import javax.mail.internet.InternetAddress;
import javax.mail.MessagingException;
import java.util.*;
import SpamCash.Client.GUI.OpenSafeDialog;
import javax.mail.internet.AddressException;

public class Control
{
    private POP3Proxy pop3Proxy;
    private IMAPProxy imapProxy;
    private SMTPProxy smtpProxy;

    private CurrencyExchangeClient currencyExchangeClient;

    public PackageQueue incomingMailQueue;
    private IncomingMailProcessor incomingQueueMailProcessor;
    public IncomingMailProcessor incomingMailProcessor;
    private OutgoingMailProcessor outgoingMailProcessor;

    private boolean POP3ServerShouldWait = false;
    private boolean areAllMailsQueued = false;

    public static Certificate CESCertificate, BLSCertificate, RSCertificate;
```

```
    private static Certificate certificate;
    private boolean registered;

    private Blacklist blacklist;
    private Whitelist whitelist;

    private SafeHandler safe;
    private String password;

    private static MainFrame gui;
    public final static int ID_OPENSAFE = 2, ID_POP3 = 1;

    public Control()
    {
        Debugger.setApplicationName("Client");
        try
        {
            gui = new MainFrame();

            //Clear temporary mail directorys
            FileHandler.clearDirectory(Config.tempIncomingMailDirectory);
            FileHandler.clearDirectory(Config.outgoingMailDirectory);

            //Load proxy configuration
            try
            {
                Config.setProxyConfiguration(this.loadConfiguration());
                Debugger.debug(getClass(), 1, "Configuration file loaded succesfully.");
            }
            catch(Exception e)
            {
                Debugger.debug(getClass(), 1, "Configuration file could not be loaded.");
                registered = false;
            }

            pop3Proxy = new POP3Proxy(this, 110);
            pop3Proxy.start();


            imapProxy = new IMAPProxy(this, 143);
            imapProxy.start();

            smtpProxy = new SMTPProxy(this, 25);
            smtpProxy.start();

            incomingMailQueue = new PackageQueue();
            incomingMailProcessor = new IncomingMailProcessor(this);
            incomingQueueMailProcessor = new IncomingMailProcessor(this);
            incomingQueueMailProcessor.start();

            outgoingMailProcessor = new OutgoingMailProcessor(this);

            //Load keys...
            BLSCertificate = FileHandler.importCertificate(new
File(Config.BLSCertificateName));
            RSCertificate = FileHandler.importCertificate(new
File(Config.RSCertificateName));
            CESCertificate = FileHandler.importCertificate(new
File(Config.CESCertificateName));

            Debugger.debug(getClass(), 1, "Loaded server certificates succesfully.");

            //Load certificate
            try
            {
                certificate = FileHandler.importCertificate(new
File(Config.clientCertificatePath));
                Debugger.debug(getClass(), 1, "Loaded client certificate succesfully:
"+((X509Certificate) certificate).getSubjectDN().getName());
                if( ! new File(Config.safeFilePath).exists())
```

```
                    throw new Exception("Error");
                registered = true;
            }
            catch(Exception e)
            {
                Debugger.debug(getClass(), 1, "Certificate could not be loaded or safe
does not exist. Probably first program launch.");
                registered = false;
            }


            //Load the blacklist
            try
            {
                blacklist = (Blacklist) FileHandler.readObject(new
File(Config.blacklistFilePath));
                Debugger.debug(getClass(), 1, "Loaded blacklist succesfully.");
            }
            catch(Exception e)
            {
                Debugger.debug(getClass(), 1, "Blacklist could not be loaded. Creating
new.");
                blacklist = new Blacklist();
            }

            //Load the whitelist
            try
            {
                whitelist = (Whitelist) FileHandler.readObject(new
File(Config.whitelistFilePath));
                Debugger.debug(getClass(), 1, "Loaded whitelist succesfully.");
            }
            catch(Exception e)
            {
                Debugger.debug(getClass(), 1, "Whitelist could not be loaded. Creating
new.");
                whitelist = new Whitelist();
            }

            safe = new SafeHandler();

            currencyExchangeClient = new CurrencyExchangeClient();

            gui.init(this);

            if(registered)
            {
                byte[] testbytes =
                    {2, 99, 7, 45, 56, 7, 7, 89, 98, 7, 78, 4, 12, 3, 4, 4, 23, 22, 3, 4,
45, 45, 32};
                boolean passed = (Cryptotools.verify((RSAPublicKey)
getCertificate().getPublicKey(),
                                                     Cryptotools.sign(getPrivateKey(),
testbytes), testbytes));
                Debugger.debug(getClass(), 1,
                               "Test of the received certificate with the privatkey..." +
(passed ? "passed" : "failed"));
                if( ! passed)
                    registered = false;
            }

            if(!registered)
            {
                Debugger.debug(getClass(), 1, "Starting registration procedure...");
                gui.showRegistrationDialog();
            }
            else
                gui.show();

        }
        catch(Throwable e)
```

```
        {
            Debugger.debug(getClass(), 1, " Could not initialize " + e);
        }
    }

    public static String generateKeyFile(String filePath) throws Exception
    {
        SecretKey key = Cryptotools.generateAESKey();
        FileHandler.writeObject(key, new File(filePath));
        return filePath;
    }

    public RSAPrivateKey getPrivateKey() throws Exception
    {
        String password = getPassword();
        return getSafe().getPrivateKey();
    }

    public Certificate getCertificate()
    {
        return certificate;
    }

    public void saveConfiguration(ProxyConfiguration config) throws Exception
    {
        Config.setProxyConfiguration(config);
        FileHandler.writeObject(config, new File(Config.configurationFilePath));
    }

    public ProxyConfiguration loadConfiguration() throws Exception
    {
        return(ProxyConfiguration) FileHandler.readObject(new
File(Config.configurationFilePath));
    }

    public void collectCoin(MailInformation m) throws Exception
    {
        getSafe().transferCoin(m,getCertificate(),getPrivateKey());

        //Make the notification locally right away, in case the recipient is this
client...
        deleteCoinWithSerial(m.c.getSerialNumber());

        //Send a notification mail to the sender
        try
        {
            Properties props = System.getProperties();

            // Attaching to default Session, or we could start a new one

            props.put("mail.smtp.host", Config.configuration.outgoingMailserver);
            Session session = Session.getDefaultInstance(props, null);

            // Create a new message
            MimeMessage msg = new MimeMessage(session);

            // Set the FROM and TO fields
            msg.setFrom(m.to);
            msg.setRecipients(Message.RecipientType.TO, new Address[]
                              {m.sender});
            msg.setSubject(Config.name + " notification mail");
            msg.setText("This is a special message used in the " + Config.name
                        +
                        " system. An error has occured since you are reading this. So
this email should just be ignored.");

            // Add the signature to the header...
            Fingerprint hash = Coin.getHash2(m.c.getSerialNumber(), m.sender.toString());
            Signature notificationSignature =
Cryptotools.sign(this.getPrivateKey(),hash);
```

```
            NotificationHeader header = new NotificationHeader(notificationSignature,
m.c.getSerialNumber(), m.sender, getCertificate());
            msg = (MimeMessage)MimeTools.addAttachmentHeader(msg, header);
            msg.saveChanges();

            msg.setHeader(Config.headerNotificationName," notificationmail");
            msg.saveChanges();

            // Send the message
            Transport.send(msg);
            Debugger.debug(getClass(), 1, "Notification mail sent succesfully to: " +
m.sender);
        }
        catch(Exception ex)
        {
            Debugger.debug(getClass(), 1, "Collect coin: Could not send notification mail
to: " + m.sender, ex);
        }
    }

    public boolean deleteCoinWithSerial(long serialNumber)
    {
        return getSafe().deleteCoinWithSerial(serialNumber);
    }

    public Coin[] getCoinsToSend(int n, Address[] receivers) throws Exception
    {
        return getSafe().getCoinsToSend(n,receivers);
    }

    public void saveMail(Coin coin, MimeMessage msg) throws Exception
    {
        getSafe().addPendingMail(new MailInformation(coin, msg));
        gui.updateMailList();
    }

    public boolean isOnWhitelist(String address) throws AddressException
    {
        return whitelist.onWhitelist(new InternetAddress(address));
    }

    public void addToWhitelist(String address) throws Exception
    {
        whitelist.add(new InternetAddress(address));
    }

    public void removeFromWhitelist(InternetAddress address) throws AddressException
    {

        massMail(2,address);
        whitelist.remove(address);
    }

    public Whitelist getWhitelist()
    {
        return whitelist;
    }

    public boolean isOnBlackList(Certificate cert)
    {
        return blacklist.onBlacklist(cert);
    }

    //--------------------------------------------------------------------------
    //Methods for interacting with the Currency Exchange Server (CES)
    //--------------------------------------------------------------------------

    public void withdraw(String username, String password, double amount)
    {
        String error = "";
        Exception ex = null;
```

```
        try
        {
            currencyExchangeClient.logonToCurrencyServer(username, password);
            Coin[] withdrawedCoins = currencyExchangeClient.withdraw(amount,
certificate);
            int coinCount = 0;
            for(int i=0;i<withdrawedCoins.length;i++)
            {
                //Verify the transactionlist
                boolean verified =
withdrawedCoins[i].initialVerify(this.RSCertificate,this.CESCertificate,certificate);
                if( ! verified)
                    Debugger.debug(getClass(),1,"Could not verify the transactionlist in
the new coin. Discarding it !");
                else
                    coinCount++;

                //Sign the transactionList since the verification was OK

withdrawedCoins[i].getTransactionlist().signList(getCertificate(),getPrivateKey());

                //Save the new coins
                if(verified)
                    getSafe().addCoins(new Coin[]{withdrawedCoins[i]});
            }

            gui.showMessage("Transaction successful. Withdrawed " + coinCount + " coins
(worth "
                            + withdrawedCoins.length * Config.coinValue + ") from your
account.");
            Debugger.debug(getClass(), 1,
                           "Transaction successful. Withdrawed " + coinCount + " coins
(worth "
                           + withdrawedCoins.length * Config.coinValue + ") from your
account.");
            return;
        }
        catch(ConnectException ex1)
        {
            ex = ex1;
            error = "Withdrawal of " + amount
                + " failed. Could not establish connection with the currency exchange
server.";
        }
        catch(Exception ex2)
        {
            ex = ex2;
            error = "Withdrawal of " + amount + " failed. "+ex.getMessage();
        }
        Debugger.debug(getClass(), 1, error, ex);
        gui.showError(error);
    }

    public void deposit(String username, String password, int numberOfCoins)
    {
        String error = "";
        try
        {
            currencyExchangeClient.logonToCurrencyServer(username, password);
            double depositedAmount =
currencyExchangeClient.deposit(getSafe().getCoins(numberOfCoins));

            gui.showMessage("Transaction successful. Deposited " + numberOfCoins + "
coins (worth " + depositedAmount
                            + ") to your account.");
            Debugger.debug(getClass(), 1,
                           "Transaction successful. Deposited " + numberOfCoins + " coins
(worth " + depositedAmount
                           + ") to your account.");
            return;
        }
```

```
        catch(ConnectException ex)
        {
            error = "Deposit of " + numberOfCoins
                + " failed. Could not establish connection with the currency exchange
server.";
            Debugger.debug(getClass(), 1, error, ex);
        }
        catch(Exception ex)
        {
            //error = "Could not deposit " + numberOfCoins + " coins.";
            error = ex.getMessage();
            Debugger.debug(getClass(), 1, error, ex);
        }
        gui.showError(error);
    }

    //------------------------------------------------------------------------------
---
    //Methods for interacting with the BlackListServer
    //------------------------------------------------------------------------------
---
    private session createBLSSession() throws Exception
    {
        Debugger.debug(getClass(), 3, "Creating new session with the BLS: " +
Config.BLSAddress + ":" + Config.BLSPort);
        Socket socket = new Socket(InetAddress.getByName(Config.BLSAddress),
Config.BLSPort);
        return new session(socket, false, (RSAPublicKey)
BLSCertificate.getPublicKey(),false);
    }

    public void updateBlacklist()
    {
        Debugger.debug(getClass(), 1,"Updating blacklist....");

        if( ! blacklist.needUpdate())
        {
            Debugger.debug(getClass(),1,"No need to update the blacklist.");
            return;
        }

        try
        {
            session serverSession = createBLSSession();

            serverSession.send(new GetBlacklistPacket(blacklist.elements.size()));
            BlacklistPacket responsePacket = (BlacklistPacket) serverSession.receive();
            blacklist.merge(responsePacket.blacklist);
            Debugger.debug(getClass(), 1, "Blacklist updated succesfully. New list:\n" +
blacklist);
        }
        catch(Exception ex1)
        {
            Debugger.debug(getClass(), 1, "Error updating blacklist.", ex1);
        }
    }

    public void reportCoins(Coin a, Coin b)
    {
        Debugger.debug(getClass(), 1, "Reporting coins.... ");
        Debugger.debug(getClass(), 3, "Coin a: "+a);
        Debugger.debug(getClass(), 3, "Coin b: "+b);
        try
        {
            session serverSession = createBLSSession();
            a.encryptTransactionlist((RSAPublicKey)BLSCertificate.getPublicKey());
            b.encryptTransactionlist((RSAPublicKey)BLSCertificate.getPublicKey());
            serverSession.send(new ReportCoinsPacket(a, b));
            Debugger.debug(getClass(), 1, "Reported coins succesfully.");
        }
        catch(Exception ex1)
```

```
        {
            Debugger.debug(getClass(), 1, "Error reporting coin.");
        }
    }

    //------------------------------------------------------------------------
    //Methods for interacting with the Registration Server (RS)
    //------------------------------------------------------------------------

    private session createRSSession() throws Exception
    {
        Debugger.debug(getClass(), 3, "Creating new session with the RS: " +
Config.RSAddress + ":" + Config.RSPort);
        Socket socket = new Socket(InetAddress.getByName(Config.RSAddress),
Config.RSPort);
        return new session(socket, false, (RSAPublicKey)
RSCertificate.getPublicKey(),true);
    }

    public void startRegistration(String CESusername, String CESpassword, String
password, String email) throws Exception
    {
        Debugger.debug(getClass(), 3,
                       "Starting registration using: CESusername = " + CESusername + "
CESpassword = " + CESpassword
                       + " Local password = " + password + " E-mail = " + email);

        this.password = password;
        session RSsession = createRSSession();
        //Keystore-password = key-password = safe-password

        //data = {firstname,lastname,orgUnit,orgName,locality,state,country}
        Random r = new Random();

        if(Config.hashEmailAdresses)
            email = new String(Cryptotools.SHA1(email).getBytes());

        String[] data = { email, "", "", "", "", "", ""};
        boolean ok = Cryptotools.generateKeyPair(Config.name, new
File(Config.keystoreFilePath).getAbsolutePath(),
                                                 "rsa", Config.clientAsymmetricKeySize,
                                                 password, password, data);
        if(!ok)
            throw new Exception("Keypair generation during registration failed.");

        //Save the private key
        getSafe().addPrivateKey((RSAPrivateKey)
Cryptotools.getKey(Config.keystoreFilePath, Config.name, password, password));
        byte[] certReq = Cryptotools.generateCertRequest(Config.name, new
File(Config.keystoreFilePath).getAbsolutePath(),
                                                         password, password);
        //The key has now been saved in the SpamCash-file so we delete the keystore.
        if(new File(Config.keystoreFilePath).delete())
            Debugger.debug(getClass(),2,"Keystore deleted.");
        else
            Debugger.debug(getClass(),2,"Keystore could not be deleted.");

        RSsession.send(new RegistrationRequestPacket(certReq));

        RegistrationInfoPacket registrationInfo = (RegistrationInfoPacket)
RSsession.receive();

        //Withdraw 1 to avoid replayattacks
        registrationInfo.activationNumber--;

        currencyExchangeClient.logonToCurrencyServer(CESusername, CESpassword);
        currencyExchangeClient.send(registrationInfo);
        StatusPacket status = (StatusPacket) currencyExchangeClient.receive();

        if(!status.ok)
        {
```

```
            throw new Exception(
                "An error has occured. Received the following error from the Currency
Exchange Server:\n\n"
                + status.message);
        }

        Debugger.debug(getClass(), 3, "Retrieving certificate...");
        RSsession.send(registrationInfo);

        NetPacket pack = RSsession.receive();
        if(pack instanceof StatusPacket)
        {
            String error = ((StatusPacket) pack).message;
            Debugger.debug(getClass(), 3, "Error completing the registration " + error);
            throw new Exception(error);
        }
        else if(pack instanceof ActivationCompletedPacket)
        {
            certificate = ((ActivationCompletedPacket) pack).certificate;
            FileHandler.export(certificate, new File(Config.clientCertificatePath));
            Debugger.debug(getClass(), 3, "Stored certificate successfully. Starting
withdrawal.");
            gui.hideRegistrationDialog();
            gui.withdraw(Config.registrationCoinValue, CESusername, CESpassword);
        }
        else
            throw new Exception("Error in registration completion. Registration server
responded unexpectedly.");

        byte[] testbytes = {2,99,7,45,56,7,7,89,98,7,78,4,12,3,4,4,23,22,3,4,45,45,32};
        Debugger.debug(getClass(), 1,"Test of the received certificate with the
privatkey..."+((Cryptotools.verify((RSAPublicKey)getCertificate().getPublicKey(),Cryptoto
ols.sign(getPrivateKey(), testbytes),testbytes)) ? "passed" : "failed"));
    }

    //------------------------------------------------------------------------------------
---
    //Methods for the mail-proxy
    //------------------------------------------------------------------------------------
---

    public void setAreAllMailsQueued(boolean b)
    {
        areAllMailsQueued = b;
    }

    public boolean getAreAllMailsQueued()
    {
        return areAllMailsQueued;

    }

    public void incomingMailProcessingCompleted()
    {
        POP3ServerShouldWait = false;
        wakeUptheSleeping();
    }

    /**
     * Monitor to make it possible for other processes to wait
     * until some job is done.
     * @param id int identifies which process made this call
     */
    public synchronized void waitForOrderToGo(int id)
    {
        while(true)
        {
            try
            {
                //Debugger.debug(getClass(), 2, "I am awake !");
```

```
            //Are all the mails retrieved yet? (POP3)
            if(id == ID_POP3 && !POP3ServerShouldWait)
                return;

            //Is the safe open
            if(id == ID_OPENSAFE && this.safe.isOpen())
                return;

            //Debugger.debug(getClass(), 2, "Sleeping again...");
            wait();
        }
        catch(Exception e)
        {
            Debugger.debug(getClass(), 1, "Jobmonitor interruptedexception: " + e);
        }
    }
}

public synchronized void wakeUptheSleeping()
{
    notifyAll();
}

public synchronized void setPOP3ServerShouldWait(boolean b)
{
    POP3ServerShouldWait = b;
}

public SafeHandler getSafe()
{
    if(safe.isOpen())
        return safe;
    //The safe is not loaded we need a promt...
    gui.tryToOpenSafe(OpenSafeDialog.CONTROL_THREAD);
    waitForOrderToGo(ID_OPENSAFE);
    return safe;
}

public void openSafe(String path, String password) throws Exception
{
    safe.openSafe(path, password);
    this.password = password;
}

public String getPassword()
{
    if(password != null)
        return password;
    getSafe();
    return password;
}

public static void showError( String error)
{
    gui.showError(error);
}

public void shutdown()
{
    Debugger.debug(getClass(), 1, "Shutting down...");
    Debugger.debug(getClass(), 1, "Writing safe to disk...");
    try
    {
        if(safe.isOpen())
            safe.saveToFile();
        FileHandler.writeObject(blacklist,new File(Config.blacklistFilePath));
        FileHandler.writeObject(whitelist,new File(Config.whitelistFilePath));
        Debugger.debug(getClass(), 1, "Data was written successfully.");
    }
    catch(Exception ex)
    {
```

```
            Debugger.debug(getClass(), 1, "Could not write data to disk.",ex);
            gui.showError("There was an IO-error when trying to save state.");
            return;
        }
        System.exit(0);
    }


    //--------------------------------------------------------------------------
    //Test functions...
    //--------------------------------------------------------------------------

    public static void makeDelay() throws Exception
    {

        //Sleeptime in seconds
        int sleepTime = 10;
        int sleptTime = 0;

        Debugger.debug(Control.class, 3, "Sleeping " + sleepTime + " seconds...");

        while(sleptTime < sleepTime)
        {
            Thread.sleep(1000);
            sleptTime += 1;
            Debugger.debug(Control.class, 3, ".." + sleptTime);
        }
        Debugger.debug(Control.class, 3, "\ndone. =)");
    }

    public static void main(String[] args)
    {
        try
        {
            Control c = new Control();

            //testCoinCompression();

            //testBlacklistserver();
            //testCoinVerification();
        }
        catch(Exception e)
        {
            System.out.println("Exception...");
            e.printStackTrace();
        }

    }

    private void massMail(int numberOfMails,Address recipient)
    {
        //Send a lot of mails...
        try
        {
            Debugger.debug(getClass(), 1,"Mass-mailing to "+recipient+" started...");
            Properties props = System.getProperties();

            // -- Attaching to default Session, or we could start a new one --

            props.put("mail.smtp.host", Config.configuration.outgoingMailserver);
            Session session = Session.getDefaultInstance(props, null);
            for(int i=0;i<numberOfMails;i++)
            {
                // -- Create a new message --
                MimeMessage msg = new MimeMessage(session);

                // -- Set the FROM and TO fields --
                msg.setFrom(new
InternetAddress(Config.name+"@"+Config.name+"."+Config.name));
                msg.setRecipients(Message.RecipientType.TO, new Address[]
                                  {recipient});
```

```
                msg.setSubject(Config.name + " test mail "+i);
                msg.setText("...");
                msg = MimeTools.addAttachment(msg,Config.mainDirectory +
System.getProperty("file.separator") + "testMessages" +
System.getProperty("file.separator") + "test.bin");
                msg = outgoingMailProcessor.processMail(msg);
                //msg.saveChanges();
                // -- Send the message --
                Transport.send(msg);
                Debugger.debug(getClass(), 1, "Mail "+i+" / "+numberOfMails+" sent
succesfully to: " + recipient);
            }

            Debugger.debug(getClass(), 1,"Mass-mailing to "+recipient+" completed.");
        }
        catch(Exception ex)
        {
            Debugger.debug(getClass(), 1, "Error sending mail to: " + recipient, ex);
        }

    }

    public static void testCoinCompression() throws Exception
    {
        Certificate CEScert = FileHandler.importCertificate(new
File(Config.CESCertificateName));
            Certificate BLScert = FileHandler.importCertificate(new
File(Config.BLSCertificateName));
            Certificate RScert = FileHandler.importCertificate(new
File(Config.RSCertificateName));
            RSAPrivateKey CESprivkey = (RSAPrivateKey) FileHandler.readObject(new
File(Config.CESPrivateKeyName));

            Certificate cert1 = FileHandler.importCertificate(new File(Config.keydir +
"\\user1_test_certificate.crt"));

            RSAPrivateKey privkey1 = (RSAPrivateKey) FileHandler.readObject(new
File(Config.keydir
                                                                                +
"\\user1_test_private.key"));

            // coin stuff
            double value = 1.2;
            long serialnumber1 = 123456789;
            long gc1 = System.currentTimeMillis();
            int maxUsages = 10;


            // 1: creating 2 coins with equal serial-number but different first element
in transactionlist (highly unlikely case)

            Coin a = createCoin(gc1, serialnumber1, value, maxUsages, cert1, CESprivkey,
CEScert);
            System.out.println("Before: "+a);
            Coin b = new Coin(a.getCompressed());
            System.out.println("After: "+b);
            System.out.println("Compress Coin test successful? "+a.equals(b));

    }

    public static void testCoinVerification() throws MessagingException, Exception
    {
        //Testing the verification of coins...
        Control c = new Control();
        System.out.println(((X509Certificate) certificate).getSubjectDN().getName());
        String filename = "data/testMessages/testMessageHTML.msg";
        MimeMessage msg = MimeTools.readMimeFile(filename);

        String processed = "data/testMessages/withCoin.msg";
        MimeTools.writeMimeFile(c.outgoingMailProcessor.processMail(msg), processed);
```

```
        MimeMessage ext2 = MimeTools.readMimeFile(processed);

        c.incomingMailProcessor.processPOP(ext2);
    }

    public static void testBlacklistserver()
    {
        try
        {
            //-------------------------------------------------------------------------
---------------------------

            SpamCash.Servers.BLS.BlackListServer BLS = new
SpamCash.Servers.BLS.BlackListServer();

            Certificate CEScert = FileHandler.importCertificate(new
File(Config.CESCertificateName));
            Certificate BLScert = FileHandler.importCertificate(new
File(Config.BLSCertificateName));
            Certificate RScert = FileHandler.importCertificate(new
File(Config.RSCertificateName));
            RSAPrivateKey CESprivkey = (RSAPrivateKey) FileHandler.readObject(new
File(Config.CESPrivateKeyName));
            RSAPrivateKey BLSprivkey = (RSAPrivateKey) FileHandler.readObject(new
File(Config.BLSPrivateKeyName));
            RSAPrivateKey RSprivkey = (RSAPrivateKey) FileHandler.readObject(new
File(Config.RSPrivateKeyName));

            Certificate cert1 = FileHandler.importCertificate(new File(Config.keydir +
"\\user1_test_certificate.crt"));
            Certificate cert2 = FileHandler.importCertificate(new File(Config.keydir +
"\\user2_test_certificate.crt"));
            Certificate cert3 = FileHandler.importCertificate(new File(Config.keydir +
"\\user3_test_certificate.crt"));
            Certificate cert4 = FileHandler.importCertificate(new File(Config.keydir +
"\\user4_test_certificate.crt"));
            Certificate cert5 = FileHandler.importCertificate(new File(Config.keydir +
"\\user5_test_certificate.crt"));
            Certificate cert6 = FileHandler.importCertificate(new File(Config.keydir +
"\\user6_test_certificate.crt"));
            Certificate cert7 = FileHandler.importCertificate(new File(Config.keydir +
"\\user7_test_certificate.crt"));

            RSAPrivateKey privkey1 = (RSAPrivateKey) FileHandler.readObject(new
File(Config.keydir
                                                                                +
"\\user1_test_private.key"));
            RSAPrivateKey privkey2 = (RSAPrivateKey) FileHandler.readObject(new
File(Config.keydir
                                                                                +
"\\user2_test_private.key"));
            RSAPrivateKey privkey3 = (RSAPrivateKey) FileHandler.readObject(new
File(Config.keydir
                                                                                +
"\\user3_test_private.key"));
            RSAPrivateKey privkey4 = (RSAPrivateKey) FileHandler.readObject(new
File(Config.keydir
                                                                                +
"\\user4_test_private.key"));
            RSAPrivateKey privkey5 = (RSAPrivateKey) FileHandler.readObject(new
File(Config.keydir
                                                                                +
"\\user5_test_private.key"));
            RSAPrivateKey privkey6 = (RSAPrivateKey) FileHandler.readObject(new
File(Config.keydir
                                                                                +
"\\user6_test_private.key"));
            RSAPrivateKey privkey7 = (RSAPrivateKey) FileHandler.readObject(new
File(Config.keydir
                                                                                +
"\\user7_test_private.key"));
```

```
            //Verify certificates...
            BLScert.verify(RScert.getPublicKey());
            CEScert.verify(RScert.getPublicKey());
            cert1.verify(RScert.getPublicKey());
            cert2.verify(RScert.getPublicKey());
            cert3.verify(RScert.getPublicKey());
            cert4.verify(RScert.getPublicKey());
            cert5.verify(RScert.getPublicKey());
            cert6.verify(RScert.getPublicKey());
            cert7.verify(RScert.getPublicKey());

            Address address1[] =
                {new InternetAddress("santa1@greenland.net")};
            Address address2[] =
                {new InternetAddress("santa2@greenland.net")};
            Address address3[] =
                {new InternetAddress("santa3@greenland.net")};
            Address address4[] =
                {new InternetAddress("santa4@greenland.net")};
            Address address5[] =
                {new InternetAddress("santa5@greenland.net")};
            Address address6[] =
                {new InternetAddress("santa6@greenland.net")};
            Address address7[] =
                {new InternetAddress("santa7@greenland.net")};

            // coin stuff
            double value = 1.2;
            long serialnumber1 = 123456789;
            long serialnumber2 = 987654321;
            long gc1 = System.currentTimeMillis()*2+2;
            long gc2 = System.currentTimeMillis()*2;
            int maxUsages = 10;

            // 1: creating 2 coins with equal serial-number but different first element
in transactionlist (highly unlikely case)
            System.out.println("-----------------------------------Test 1-------------
----------------------");
            Coin a = createCoin(gc1, serialnumber1, value, maxUsages, cert1, CESprivkey,
CEScert);
            Coin b = createCoin(gc1, serialnumber1, value, maxUsages, cert2, CESprivkey,
CEScert);

            ReportCoinsPacket packet1 = new ReportCoinsPacket(a, b);

            Coin c = createCoin(gc1, serialnumber1, value, maxUsages, cert1, CESprivkey,
CEScert);

            addType2(c, cert1, privkey1);

            addType1(c, privkey1, address2);
            addType2(c, cert2, privkey2);

            // 2: Reporting 2 identical coins should not result in any blacklisting which
is tested now
            System.out.println("-----------------------------------Test 2-------------
----------------------");
            BLS.reportCoins(new ReportCoinsPacket(c, c));

            // 3: Reporting 2 coins where the person who is on 2nd position in the
transactionlist copied a coin and used it to send to two different people
            System.out.println("-----------------------------------Test 3-------------
----------------------");
            //------------------ (used for test 4:)

            Coin c41 = new Coin(c);
            Coin c42 = new Coin(c);

            // --------------------
```

```
            addType1(c, privkey2, address4);
            addType2(c, cert4, privkey4);

            Coin e = new Coin(c);

            //Test copy constructor....
            /*System.out.println("c.equals(e) = "+c.equals(e));
            System.out.println("hash c :
"+Cryptotools.byteToHexString(Cryptotools.SHA1(((TransactionlistElement)c.transactionList
.list.elementAt(0))).getBytes()));
            System.out.println("hash e :
"+Cryptotools.byteToHexString(Cryptotools.SHA1(((TransactionlistElement)e.transactionList
.list.elementAt(0))).getBytes()));


            System.out.println(" c :
"+((TransactionlistElement)c.transactionList.list.elementAt(0)));
            System.out.println(" e :
"+((TransactionlistElement)e.transactionList.list.elementAt(0)));


            System.out.println("VERIFY e: "+e.verify(RScert,CEScert));
            System.out.println("VERIFY c: "+c.verify(RScert,CEScert));
            if(0==0)
                return;
            */

            addType1(c, privkey4, address5);
            addType1(e, privkey4, address6);

            //now each of the two recipients sign a hash of the transactionlist and tries
to deposit the coins. The CES then reports the copying to BLS.

            addType2(c, cert5, privkey5);
            addType2(e, cert6, privkey6);

            BLS.reportCoins(new ReportCoinsPacket(c, e));

            // 4: different length of transactionlists (someone has deposited a coin and
thereafter used it for sending)
            System.out.println("-----------------------------------Test 4-------------
----------------------");
            addType1(c41, privkey2, address7);
            addType2(c41, cert7, privkey7);

            // now reporting the two coins

            BLS.reportCoins(new ReportCoinsPacket(c41, c42));

            System.out.println("-----------------------------------Test 5-------------
----------------------");

            BLS.reportCoins(new ReportCoinsPacket(c42, c41));

            // Now sending all the reportcoinspackets to BLS to see if it works as
intended :-)

            System.out.println("Initial Blacklist (empty):\n"+BLS.getBlacklist(new
GetBlacklistPacket(0)));
            BLS.reportCoins(packet1);
            System.out.println("packet1 tested successfully:\n"+BLS.getBlacklist(new
GetBlacklistPacket(0)));
            System.out.println("packet5 tested successfully:\n"+BLS.getBlacklist(new
GetBlacklistPacket(0)));
        }
        catch(Exception e)
        {
            System.out.println("Exception...");
            e.printStackTrace();
        }
```

```
    }

    public static Coin createCoin(long gc, long serialnumber, double value, int
maxUsages,
                                  Certificate cert, RSAPrivateKey CESprivkey, Certificate
CEScert) throws Exception
    {
        Fingerprint fp = Fingerprint.create(cert.getEncoded(), "SHA1");
        fp = Cryptotools.merge(new Long(serialnumber), fp);
        org.logi.crypto.sign.Signature sig1 = Cryptotools.sign(CESprivkey, fp);
        Transactionlist tlist = new UnencryptedTransactionlist(new
TransactionlistElement(sig1, CEScert));
        Coin a = new Coin(value, serialnumber, gc, maxUsages, null, tlist);
        a.setCESSignature(Cryptotools.sign(CESprivkey, a.getHash1()));
        return a;
    }

    public static void addType2(Coin c, Certificate cert, RSAPrivateKey privkey) throws
Exception
    {
        Fingerprint listHash = c.getTransactionlist().hashList();
        Signature listSignature = Cryptotools.sign(privkey, listHash);
        c.getTransactionlist().addElement(new TransactionlistElement(listSignature,
cert));
    }

    public static void addType1(Coin c, RSAPrivateKey privkey, Address[] address) throws
Exception
    {
        Fingerprint fp = Cryptotools.merge(new Long(c.getSerialNumber()),
Cryptotools.SHA1(address));
        c.getTransactionlist().addElement(new
TransactionlistElement(Cryptotools.sign(privkey, fp), null));
    }
}
```

10.1.2 CurrencyExchangeClient

```
package SpamCash.Client;

import java.util.Vector;
import java.net.*;

import SpamCash.NetworkPackets.*;
import SpamCash.Utilities.*;
import java.security.Key;
import SpamCash.Utilities.Debugger;
import SpamCash.Servers.BLS.Blacklist;
import java.security.cert.X509Certificate;
import SpamCash.NetworkPackets.*;
import SpamCash.Currency.*;
import java.security.cert.Certificate;
import org.logi.crypto.keys.RSABlindingFactor;
import org.logi.crypto.sign.BlindFingerprint;
import java.security.interfaces.RSAPublicKey;
import java.util.Random;
import java.util.GregorianCalendar;
import java.io.File;
import SpamCash.Currency.BlindedCoin;

public class CurrencyExchangeClient
{
    private session serverSession;
    private Certificate CESCertificate,BLSCertificate;
    private String username, password;
    private final static Class className = CurrencyExchangeClient.class;
```

```
    public CurrencyExchangeClient() throws Exception
    {
        CESCertificate = FileHandler.importCertificate(new
File(Config.CESCertificateName));
        BLSCertificate = FileHandler.importCertificate(new
File(Config.BLSCertificateName));
    }

    /**
     * Method to start a currencyExchange with the server
     * @param username String
     * @param password String
     * @throws Exception An exception is thrown when the connection encounters problems
     */
    public boolean logonToCurrencyServer(String username, String password) throws
Exception
    {
        StatusPacket pack = logon(username, password);
        if(pack.ok)
        {
            this.username = username;
            this.password = password;
        }
        else
            throw new Exception(pack.message);
        return pack.ok;
    }

    private StatusPacket logon(String username, String password) throws Exception
    {
        //If a valid session does not exist..create one.
        checkSession();

        //logon
        serverSession.send(new LogonPacket(username, password));
        StatusPacket responsePacket = (StatusPacket) serverSession.receive();
        return responsePacket;
    }

    private void checkSession() throws Exception
    {
        if(serverSession == null || serverSession.closed)
        {
            createSession();

        }
    }

    private void createSession() throws Exception
    {
        Debugger.debug(getClass(), 3, "Creating new session with the CES: " +
Config.CESAddress + ":" + Config.CESPort);
        Socket socket = new Socket(InetAddress.getByName(Config.CESAddress),
Config.CESPort);
        serverSession = new session(socket, false, (RSAPublicKey)
CESCertificate.getPublicKey(),true);
    }

    public Coin[] withdraw(double amount, Certificate certificate) throws Exception
    {
        Debugger.debug(getClass(), 3, "Withdrawing "+amount+"...");
        int necessaryCoins = necessaryCoins(amount);
        Coin[][] coins = generateCoins(necessaryCoins);

        checkSession();

        //Initialize blindingFactors and blind the coins
        RSABlindingFactor[][] blindingFactors = new
RSABlindingFactor[coins.length][coins[0].length];
        BlindedCoin[][] blindedCoins = new BlindedCoin[coins.length][coins[0].length];
        for(int i = 0; i < coins.length; i++)
```

```
            for(int j = 0; j < coins[0].length; j++)
            {
                Debugger.debug(getClass(), 1, "Blinding coin "+(1+j+i*coins[0].length)+"
of "+coins.length*coins[0].length);
                Thread.sleep(20);
                blindingFactors[i][j] = Cryptotools.createBlindingFactor();
                blindedCoins[i][j] = coins[i][j].getBlindedCoin((RSAPublicKey)
CESCertificate.getPublicKey(),
                                                                blindingFactors[i][j],
certificate);
            }

        serverSession.send(new CurrencyExchangeRequestPacket(blindedCoins));
        RequestBlindingFactorsPacket responsePacket = (RequestBlindingFactorsPacket)
serverSession.receive();

        Coin[] newCoins = new Coin[coins.length];
        RSABlindingFactor[] chosenBlindingFactors = new RSABlindingFactor[coins.length];
        //Remove one blindingfactor (pointed out by the server) in each set
        for(int i = 0; i < blindingFactors.length; i++)
        {
            chosenBlindingFactors[i] = blindingFactors[i][responsePacket.index[i]];
            blindingFactors[i][responsePacket.index[i]] = null;
            newCoins[i] = coins[i][responsePacket.index[i]];
            coins[i][responsePacket.index[i]] = null;
        }

        serverSession.send(new CurrencyBlindingFactorPacket(blindingFactors, coins,
certificate));
        CoinSignaturePacket coinSignaturePacket = (CoinSignaturePacket)
serverSession.receive();

        //Save the signatures in the coins
        for(int i = 0; i < coins.length; i++)
        {

newCoins[i].setCESSignature(coinSignaturePacket.blindedcoins[i].getAndUnblindSignature1((
RSAPublicKey)this.

CESCertificate.

getPublicKey(),
                chosenBlindingFactors[i]));
            newCoins[i].setTransactionlist(new UnencryptedTransactionlist(new
TransactionlistElement(

coinSignaturePacket.blindedcoins[i].getAndUnblindSignature2((RSAPublicKey)this.CESCertifi
cate.

getPublicKey(), chosenBlindingFactors[i]), this.CESCertificate)));
        }

        return newCoins;
    }

    public double deposit(Coin[] coins) throws Exception
    {
        for(int i=0;i< coins.length;i++)
        {

coins[i].encryptTransactionlist((RSAPublicKey)this.BLSCertificate.getPublicKey());
        }

        serverSession.send(new DepositRequestPacket(coins));
        AmountDepositedPacket depositedPacket = (AmountDepositedPacket)
serverSession.receive();
        return depositedPacket.amount;
    }

    private static int necessaryCoins(double amount)
    {
```

```
        return(int) (amount / Config.coinValue);
    }

    private Coin[][] generateCoins(int necessaryCoins)
    {
        Coin[][] coins = new Coin[necessaryCoins][Config.numberOfCoinsPerCoinset];

        //Generate radom serialNumbers
        Random rand = new Random();

        //Expirationtime is calculated
        long expirationTime = System.currentTimeMillis();
        expirationTime += Config.expirationTimeInMilliseconds;

        for(int i = 0; i < coins.length; i++)
            for(int j = 0; j < coins[0].length; j++)
            {
                coins[i][j] = new Coin(Config.coinValue, rand.nextLong(), expirationTime,
Config.maxNumberOfUsages, null, null);
            }

        return coins;
    }

    public void reportCoins(Coin c1, Coin c2) throws Exception
    {
        checkSession();
        serverSession.send(new ReportCoinsPacket(c1, c2));
    }

    public void send(NetPacket pack) throws Exception
    {
        checkSession();
        serverSession.send(pack);
    }

    public NetPacket receive() throws Exception
    {
        checkSession();
        return serverSession.receive();
    }

    public static void main(String[] s)

    {
        double amount = 25.2;
        Debugger.debug(className, 3, "Amount = " + amount + "  Necessary coins = " +
necessaryCoins(amount));
    }
}
```

10.1.3 Whitelist

```
package SpamCash.Client;


import java.util.Vector;
import javax.mail.Address;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.AddressException;

public class Whitelist implements java.io.Serializable
{
    private Vector list;

    public Whitelist()
    {
        list = new Vector();
    }
```

```
    public void add(InternetAddress address) throws Exception
    {
        try
        {
            address.validate();
        }
        catch(AddressException e)
        {
            throw new Exception("Not a valid e-mail address.");
        }
        if(onWhitelist(address))
            throw new Exception(address + " is already on the whitelist.");
        list.add(address);
    }

    public void remove(InternetAddress address)
    {
        list.remove(address);
    }

    public boolean onWhitelist(Address address)
    {
        return list.contains(address);
    }

    public String toString()
    {
        return "Whitelist: " + list.toString();
    }

    public Vector getData()
    {
        return list;
    }

   public static void main(String[] args) throws Exception
   {
       Whitelist white = new Whitelist();
       InternetAddress addr1 = new InternetAddress("ji@mobil.dk");
       InternetAddress addr2 = new InternetAddress("santa@mobil.dk");
       white.add(addr1);
       white.add(addr2);
       System.out.println(white);
       white.remove(addr2);
       System.out.println(white);
       System.out.println("is "+addr2+" on the whitelist? "+white.onWhitelist(addr2));
       System.out.println("is "+addr1+" on the whitelist? "+white.onWhitelist(addr1));
   }


}
```

10.1.4 GUI

10.1.4.a AmountPanel

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.event.*;

public class AmountPanel extends JPanel
{
    private ImageButton plusButton, minusButton;
    private FormPanel leftPanel;
    private JPanel rightPanel;
```

```
    private PlusMinusListener plusMinusListener;
    private boolean integer;
    private double incrementValue;

    public AmountPanel(String label, ActionListener listener, boolean integer, double
incrementValue)
    {
        this.incrementValue = incrementValue;
        this.integer = integer;
        leftPanel = new FormPanel(new String[]
                                  {label}
                                  , new boolean[]
                                  {false}
                                  , listener);

        //Listen for keyevents...
        leftPanel.textFields[0].addKeyListener(new keyListener());

        plusButton = new ImageButton("gradient_small", "+");
        minusButton = new ImageButton("gradient_small", "-");
        plusMinusListener = new PlusMinusListener();
        plusButton.addActionListener(plusMinusListener);
        minusButton.addActionListener(plusMinusListener);
        rightPanel = new PlusMinusPanel(plusButton, minusButton);
        setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
        add(leftPanel);
        add(Box.createHorizontalStrut(5));
        add(rightPanel);
    }

    public String getText()
    {
        return leftPanel.textFields[0].getText();
    }

    public void setText(String s)
    {
        leftPanel.textFields[0].setText(s);
    }

    private String updateField(String text)
    {
        try
        {
            Integer.parseInt(text);
        }
        catch(Exception ex)
        {
            try
            {
                if(!integer)
                {
//                    if(text.indexOf(".") != text.length()-1)
                    Double.parseDouble(text);
                    return text;
                }
            }
            catch(Exception e)
            {
                if(!text.equals(""))
                    return "0.0";
            }

            if(!text.equals(""))
                return "0";
        }
        return text;
    }

    private String addIntValue(String text, int add)
    {
```

```
        if(!updateField(text).equals(""))
            return "" + (((int) Integer.parseInt(text)) + add);
        else
            return "";
    }

    private String addDoubleValue(String text, double add)
    {
        if(!updateField(text).equals(""))
            return "" + ((double)Math.round(100*(((double) Double.parseDouble(text)) +
add)))/100;
        else
            return "";
    }

    private String subIntValue(String text, int add)
    {
        if(!updateField(text).equals("") && (((int) Integer.parseInt(text)) - add) > 0)
            return "" + ((int) Integer.parseInt(text) - add);
        else
            return "0";
    }

    private String subDoubleValue(String text, double add)
    {
        if(!updateField(text).equals("") && (((double) Double.parseDouble(text)) - add) >
0)
            return "" + ((double)Math.round(100*(((double) Double.parseDouble(text)) -
add)))/100;
        else
            return "0.0";
    }

    private class PlusMinusListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            if(getText().equals(""))
                setText("0");
            if(e.getActionCommand().equals("+"))
            {
                if(integer)
                {
                    setText(addIntValue(getText(), (int)incrementValue));
                }
                else
                    setText(addDoubleValue(getText(), incrementValue));
            }
            else
            {
                if(integer)
                {
                    setText(subIntValue(getText(), (int)incrementValue));
                }
                else
                    setText(subDoubleValue(getText(), incrementValue));

            }
        }
    }

    private class keyListener implements KeyListener
    {
        public void keyPressed(KeyEvent e)
        {

            setText(updateField(getText()));
        }

        public void keyReleased(KeyEvent e)
        {
```

```
            setText(updateField(getText()));
        }

        public void keyTyped(KeyEvent e)
        {
        }
    }

    //Just to get the BoxLayout
    private class PlusMinusPanel extends JPanel
    {
        public PlusMinusPanel(ImageButton plusButton, ImageButton minusButton)
        {
            setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
            add(plusButton);
            add(Box.createVerticalStrut(2));
            add(minusButton);
        }
    }


}
```

10.1.4.b ConfigurationPanel

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.event.*;
import java.net.InetAddress;
import javax.mail.internet.InternetAddress;
import java.util.StringTokenizer;

import SpamCash.Utilities.Debugger;
import SpamCash.Utilities.*;

public class ConfigurationPanel extends JPanel
{
    private MainFrame frame;

    private final String informationtext = "Here you can configure the mail-proxy
settings.";
    private final String[] userLabels =
        {" Your Name ", "E-mail Address "}
        ,
        serverLabels =
        {"Outgoing mail server (SMTP) ", "Incoming mail server (IMAP/POP3) "}
        ,
        logonLabels =
        {"Login name ", " Password "}
        ,
        sectionLabels =
        {"Server Information:", "User Information:", "Logon Information:"};
    private final String[] buttonTypeNames =
        {"_normal", "_pressed", "_point"};
    private FormPanel userInformationPanel, serverInformationPanel,
logonInformationPanel;
    private JTextArea informationTextArea;

    protected ConfigurationPanel(MainFrame frame)
    {
        this.frame = frame;

        ActionListener listener = new actionListener();

        informationTextArea = Definitions.getTextArea();
        informationTextArea.setText(informationtext);
```

```
        serverInformationPanel = new FormPanel(new String[]
                                               {serverLabels[0], serverLabels[1]}
                                               , new boolean[]
                                               {false, false}
                                               , listener);
        /*userInformationPanel = new FormPanel(new String[]
                                             {userLabels[0], userLabels[1]}
                                             , new boolean[]
                                             {false, false}
                                             , listener);
        logonInformationPanel = new FormPanel(new String[]
                                              {logonLabels[0], logonLabels[1]}
                                              , new boolean[]
                                              {false, true}
                                              , listener);
*/
        ImageButton button = new ImageButton("gradient", "Update");
        button.addActionListener(listener);

        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        add(informationTextArea);
        add(Box.createVerticalStrut(Definitions.getSpaceL()));
        add(new JLabel(sectionLabels[0]));
        add(Box.createVerticalStrut(Definitions.getSpaceS()));
        add(serverInformationPanel);
        add(Box.createVerticalStrut(Definitions.getSpaceL()));
        /*add(new JLabel(sectionLabels[1]));
        add(Box.createVerticalStrut(Definitions.getSpaceS()));
        add(userInformationPanel);
        add(Box.createVerticalStrut(Definitions.getSpaceL()));
        add(new JLabel(sectionLabels[2]));
        add(Box.createVerticalStrut(Definitions.getSpaceS()));
        add(logonInformationPanel);
        add(Box.createVerticalStrut(Definitions.getSpaceL()));*/
        add(button);

        setBorder(BorderFactory.createMatteBorder(Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                  Definitions.getSpaceL(),
Definitions.getSpaceL(),getBackground()));
    }

    private String checkFields()
    {
        if(!checkIP(serverInformationPanel.textFields[0].getText()))
            return "The field " + serverLabels[0] + "must be a valid DNS-address (eg.
mailserver.com or 123.123.123.123)";
        if(!checkIP(serverInformationPanel.textFields[1].getText()))
            return "The field " + serverLabels[1] + "must be a valid DNS-address (eg.
mailserver.com or 123.123.123.123)";
/*
        if(userInformationPanel.textFields[0].getText().length() == 0)
            return "Please fill in the field \"" + userLabels[0] + "\"";
        if(!checkEmail(userInformationPanel.textFields[1].getText()))
            return "The field " + userLabels[1] + "must be an email-address
(joe@smith.com)";

        if(logonInformationPanel.textFields[0].getText().length() == 0)
            return "Please fill in the field \"" + logonLabels[0] + "\"";

        if(logonInformationPanel.textFields[1].getText().length() == 0)
            return "Please fill in the field \"" + logonLabels[1] + "\"";*/
        return "";
    }

    private boolean checkIP(String ip)
    {
        try
        {
            InetAddress.getByName(ip);
        }
```

```
        catch(Exception e)
        {
            return false;
        }
        return true;
    }

    private boolean checkEmail(String email)
    {
        try
        {
            new InternetAddress(email,true);
        }
        catch(Exception e)
        {
            return false;
        }
        return true;
    }

    protected void setConfigInfo(ProxyConfiguration config)
    {
        serverInformationPanel.textFields[0].setText(config.outgoingMailserver);
        serverInformationPanel.textFields[1].setText(config.incomingMailserver);
        /*userInformationPanel.textFields[0].setText(config.name);
        userInformationPanel.textFields[1].setText(config.email);
        logonInformationPanel.textFields[0].setText(config.loginName);
        logonInformationPanel.textFields[1].setText(config.password);*/
    }

    private class actionListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            //Filled in correctly?
            String check = checkFields();
            if(check.length() != 0)
            {
                frame.showError(check);
                //Debugger.debug(getClass(),3,"Typing error: "+check);
                return;
            }

            frame.saveConfigurationInfo(new
ProxyConfiguration(serverInformationPanel.textFields[0].getText(),
                                        serverInformationPanel.textFields[1].getText()/*,
                                        userInformationPanel.textFields[0].getText(),
                                        userInformationPanel.textFields[1].getText(),
                                        logonInformationPanel.textFields[0].getText(),

logonInformationPanel.textFields[1].getText()*/));

        }
    }


}
```

10.1.4.c Definitions

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.*;
import javax.swing.plaf.metal.*;
import java.util.*;
import javax.swing.plaf.*;
```

```
public class Definitions extends MetalLookAndFeel
{
    private final static int largeSpace = 20, smallSpace = 5;

    private final static Color backgroundColor = new Color(0, 0, 0);
    private final static Color borderColor1 = new Color(161, 248, 237);
    private final static Color borderColor2 = new Color(111, 248, 237);
    private final static Color borderColor3 = new Color(77, 177, 169);
    private final static Color panelBackground = new Color(30, 30, 30);
    private final static Color textColor = new Color(200, 200, 200);
    private final static FontUIResource font1 = new FontUIResource("Arial", Font.PLAIN,
12);

    private final static ColorUIResource textFieldBackgroundColor = new
ColorUIResource(0, 0, 0);
    private final static int textFieldBorderSize = 1;
    private final static Color textFieldForegroundColor = borderColor2;
    private final static Color textFieldBorderColor = new Color(36, 97, 197);
    private final static Color textFieldForegroundColorInactive = new
ColorUIResource(160, 160, 160);
    private final static FontUIResource textFieldFont = font1;


    private final static Color comboBoxBackgroundColor = backgroundColor;
    private final static Color comboBoxForegroundColor = textColor;
    private final static Color comboBoxSelectionBackgroundColor = panelBackground;
    private final static Color comboBoxSelectionForegroundColor = borderColor2;

    private final static Color textAreaForegroundColor = textColor;

    public final static Color buttonTextColor = textColor;

    private final Object[] defaults =
        {
//"activeCaption", new ColorUIResource(0,0,0),
//"activeCaptionBorder", new ColorUIResource(0,0,0),
//"activeCaptionText", new ColorUIResource(0,0,0),
//"AuditoryCues.allAuditoryCues", [Ljava.lang.Object;@64dc11
//"AuditoryCues.cueList", [Ljava.lang.Object;@64dc11
//"AuditoryCues.defaultCueList", [Ljava.lang.Object;@1a626f
//"AuditoryCues.noAuditoryCues", [Ljava.lang.Object;@763f5d
//"Button.background", new ColorUIResource(0,0,0),
//"Button.border", javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@1546e25
//"Button.darkShadow", new ColorUIResource(0,0,0),
//"Button.disabledText", new ColorUIResource(0,0,0),
//"Button.focus", new ColorUIResource(0,0,0),
//"Button.focusInputMap", javax.swing.plaf.InputMapUIResource@89cf1e
//"Button.font", font1,
//"Button.foreground", new ColorUIResource(0,0,0),
//"Button.highlight", new ColorUIResource(0,0,0),
//"Button.light", new ColorUIResource(0,0,0),
//"Button.margin", javax.swing.plaf.InsetsUIResource[top=2,left=14,bottom=2,right=14]
//"Button.select", new ColorUIResource(0,0,0),
//"Button.shadow", new ColorUIResource(0,0,0),
//"Button.textIconGap", 4
//"Button.textShiftOffset", 0
//"ButtonUI", javax.swing.plaf.metal.MetalButtonUI
//"CheckBox.background", new ColorUIResource(0,0,0),
//"CheckBox.border", javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@7bd9f2
//"CheckBox.disabledText", new ColorUIResource(0,0,0),
//"CheckBox.focus", new ColorUIResource(0,0,0),
//"CheckBox.focusInputMap", javax.swing.plaf.InputMapUIResource@10e790c
//"CheckBox.font", font1,
//"CheckBox.foreground", new ColorUIResource(0,0,0),
//"CheckBox.icon", javax.swing.plaf.metal.MetalIconFactory$CheckBoxIcon@c68c3
//"CheckBox.margin", javax.swing.plaf.InsetsUIResource[top=2,left=2,bottom=2,right=2]
//"Checkbox.select", new ColorUIResource(0,0,0),
//"CheckBox.textIconGap", 4
//"CheckBox.textShiftOffset", 0
//"CheckBoxMenuItem.acceleratorFont", font1,
//"CheckBoxMenuItem.acceleratorForeground", new ColorUIResource(0,0,0),
```

```
//"CheckBoxMenuItem.acceleratorSelectionForeground", new ColorUIResource(0,0,0),
//"CheckBoxMenuItem.arrowIcon",
javax.swing.plaf.metal.MetalIconFactory$MenuItemArrowIcon@1e9cb75
//"CheckBoxMenuItem.background", new ColorUIResource(0,0,0),
//"CheckBoxMenuItem.border", javax.swing.plaf.metal.MetalBorders$MenuItemBorder@12a3722
//"CheckBoxMenuItem.borderPainted", true
//"CheckBoxMenuItem.checkIcon",
javax.swing.plaf.metal.MetalIconFactory$CheckBoxMenuItemIcon@641e9a
//"CheckBoxMenuItem.commandSound", sounds/MenuItemCommand.wav
//"CheckBoxMenuItem.disabledForeground", new ColorUIResource(0,0,0),
//"CheckBoxMenuItem.font", font1,
//"CheckBoxMenuItem.foreground", new ColorUIResource(0,0,0),
//"CheckBoxMenuItem.margin",
javax.swing.plaf.InsetsUIResource[top=2,left=2,bottom=2,right=2]
//"CheckBoxMenuItem.selectionBackground", new ColorUIResource(0,0,0),
//"CheckBoxMenuItem.selectionForeground", new ColorUIResource(0,0,0),
//"CheckBoxMenuItemUI", javax.swing.plaf.basic.BasicCheckBoxMenuItemUI
//"CheckBoxUI", javax.swing.plaf.metal.MetalCheckBoxUI

//"ColorChooser.background", new ColorUIResource(0,0,0),
//"ColorChooser.font", font1,
//"ColorChooser.foreground", new ColorUIResource(0,0,0),
//"ColorChooser.rgbBlueMnemonic", 66
//"ColorChooser.rgbGreenMnemonic", 78
//"ColorChooser.rgbRedMnemonic", 68
//"ColorChooser.swatchesDefaultRecentColor", new ColorUIResource(0,0,0),
//"ColorChooser.swatchesRecentSwatchSize", java.awt.Dimension[width=10,height=10]
//"ColorChooser.swatchesSwatchSize", java.awt.Dimension[width=10,height=10]
//"ColorChooserUI", javax.swing.plaf.basic.BasicColorChooserUI
//"ComboBox.ancestorInputMap", javax.swing.plaf.InputMapUIResource@1ac1fe4
        "ComboBox.background", comboBoxBackgroundColor,
        "ComboBox.buttonBackground", comboBoxBackgroundColor,
        "ComboBox.buttonDarkShadow", new ColorUIResource(0, 0, 0),
        "ComboBox.buttonHighlight", new ColorUIResource(0, 0, 0),
        "ComboBox.buttonShadow", new ColorUIResource(0, 0, 0),
//"ComboBox.disabledBackground", new ColorUIResource(0,0,0),
//"ComboBox.disabledForeground", new ColorUIResource(0,0,0),
        "ComboBox.font", font1,
        "ComboBox.foreground", comboBoxForegroundColor,
        "ComboBox.selectionBackground", comboBoxSelectionBackgroundColor,
        "ComboBox.selectionForeground", comboBoxSelectionForegroundColor,
//"ComboBoxUI", javax.swing.plaf.metal.MetalComboBoxUI
//"control", new ColorUIResource(0,0,0),
//"controlDkShadow", new ColorUIResource(0,0,0),
//"controlHighlight", new ColorUIResource(0,0,0),
//"controlLtHighlight", new ColorUIResource(0,0,0),
//"controlShadow", new ColorUIResource(0,0,0),
//"controlText", new ColorUIResource(0,0,0),
//"desktop", new ColorUIResource(0,0,0),
//"Desktop.ancestorInputMap", javax.swing.plaf.InputMapUIResource@12558d6
//"Desktop.background", new ColorUIResource(0,0,0),
//"DesktopIcon.background", new ColorUIResource(0,0,0),
//"DesktopIcon.border",
javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@16672d6
//"DesktopIcon.font", font1,
//"DesktopIcon.foreground", new ColorUIResource(0,0,0),
//"DesktopIcon.width", 160
//"DesktopIconUI", javax.swing.plaf.metal.MetalDesktopIconUI
//"DesktopPaneUI", javax.swing.plaf.basic.BasicDesktopPaneUI
//"EditorPane.background", new ColorUIResource(0,0,0),
//"EditorPane.border", javax.swing.plaf.basic.BasicBorders$MarginBorder@e1d5ea
//"EditorPane.caretBlinkRate", 500
//"EditorPane.caretForeground", new ColorUIResource(0,0,0),
//"EditorPane.focusInputMap", javax.swing.plaf.InputMapUIResource@10f6d3
//"EditorPane.font", font1,
//"EditorPane.foreground", new ColorUIResource(0,0,0),
//"EditorPane.inactiveForeground", new ColorUIResource(0,0,0),
//"EditorPane.margin", javax.swing.plaf.InsetsUIResource[top=3,left=3,bottom=3,right=3]
//"EditorPane.selectionBackground", new ColorUIResource(0,0,0),
//"EditorPane.selectionForeground", new ColorUIResource(0,0,0),
//"EditorPaneUI", javax.swing.plaf.basic.BasicEditorPaneUI
```

```
//"FileChooser.ancestorInputMap", javax.swing.plaf.InputMapUIResource@1c9a690
//"FileChooser.cancelButtonMnemonic", 67
//"FileChooser.detailsViewIcon",
javax.swing.plaf.metal.MetalIconFactory$FileChooserDetailViewIcon@1968e23
//"FileChooser.directoryOpenButtonMnemonic", 79
//"FileChooser.fileNameLabelMnemonic", 78
//"FileChooser.filesOfTypeLabelMnemonic", 84
//"FileChooser.helpButtonMnemonic", 72
//"FileChooser.homeFolderIcon",
javax.swing.plaf.metal.MetalIconFactory$FileChooserHomeFolderIcon@1dfc547
//"FileChooser.listViewIcon",
javax.swing.plaf.metal.MetalIconFactory$FileChooserListViewIcon@c5c3ac
//"FileChooser.lookInLabelMnemonic", 73
//"FileChooser.newFolderIcon",
javax.swing.plaf.metal.MetalIconFactory$FileChooserNewFolderIcon@1db699b
//"FileChooser.openButtonMnemonic", 79
//"FileChooser.saveButtonMnemonic", 83
//"FileChooser.updateButtonMnemonic", 85
//"FileChooser.upFolderIcon",
javax.swing.plaf.metal.MetalIconFactory$FileChooserUpFolderIcon@16cd7d5
//"FileChooserUI", javax.swing.plaf.metal.MetalFileChooserUI
//"FileView.computerIcon",
javax.swing.plaf.metal.MetalIconFactory$TreeComputerIcon@18385e3
//"FileView.directoryIcon", javax.swing.plaf.metal.MetalIconFactory$TreeFolderIcon@e0b6f5
//"FileView.fileIcon", javax.swing.plaf.metal.MetalIconFactory$TreeLeafIcon@d19bc8
//"FileView.floppyDriveIcon",
javax.swing.plaf.metal.MetalIconFactory$TreeFloppyDriveIcon@b2a2d8
//"FileView.hardDriveIcon",
javax.swing.plaf.metal.MetalIconFactory$TreeHardDriveIcon@111a3a4
//"FocusManagerClassName", javax.swing.DefaultFocusManager
//"FormattedTextField.background", new ColorUIResource(0,0,0),
//"FormattedTextField.border",
javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@df8ff1
//"FormattedTextField.caretBlinkRate", 500
//"FormattedTextField.caretForeground", new ColorUIResource(0,0,0),
//"FormattedTextField.focusInputMap", javax.swing.plaf.InputMapUIResource@a17083
//"FormattedTextField.font", font1,
//"FormattedTextField.foreground", new ColorUIResource(0,0,0),
//"FormattedTextField.inactiveBackground", new ColorUIResource(0,0,0),
//"FormattedTextField.inactiveForeground", new ColorUIResource(0,0,0),
//"FormattedTextField.margin",
javax.swing.plaf.InsetsUIResource[top=0,left=0,bottom=0,right=0]
//"FormattedTextField.selectionBackground", new ColorUIResource(0,0,0),
//"FormattedTextField.selectionForeground", new ColorUIResource(0,0,0),
//"FormattedTextFieldUI", javax.swing.plaf.basic.BasicFormattedTextFieldUI
//"inactiveCaption", new ColorUIResource(0,0,0),
//"inactiveCaptionBorder", new ColorUIResource(0,0,0),
//"inactiveCaptionText", new ColorUIResource(0,0,0),
//"info", new ColorUIResource(0,0,0),
//"infoText", new ColorUIResource(0,0,0),
//"InternalFrame.activeTitleBackground", new ColorUIResource(0,0,0),
//"InternalFrame.activeTitleForeground", new ColorUIResource(0,0,0),
//"InternalFrame.border", javax.swing.plaf.metal.MetalBorders$InternalFrameBorder@12a54f9
//"InternalFrame.borderColor", new ColorUIResource(0,0,0),
//"InternalFrame.borderDarkShadow", new ColorUIResource(0,0,0),
//"InternalFrame.borderHighlight", new ColorUIResource(0,0,0),
//"InternalFrame.borderLight", new ColorUIResource(0,0,0),
//"InternalFrame.borderShadow", new ColorUIResource(0,0,0),
//"InternalFrame.closeIcon",
javax.swing.plaf.metal.MetalIconFactory$InternalFrameCloseIcon@1b9ce4b
//"InternalFrame.closeSound", sounds/FrameClose.wav
//"InternalFrame.icon",
javax.swing.plaf.metal.MetalIconFactory$InternalFrameDefaultMenuIcon@4fce71
//"InternalFrame.iconifyIcon",
javax.swing.plaf.metal.MetalIconFactory$InternalFrameMinimizeIcon@1d05c81
//"InternalFrame.inactiveTitleBackground", new ColorUIResource(0,0,0),
//"InternalFrame.inactiveTitleForeground", new ColorUIResource(0,0,0),
//"InternalFrame.maximizeIcon",
javax.swing.plaf.metal.MetalIconFactory$InternalFrameMaximizeIcon@19dfbff
//"InternalFrame.maximizeSound", sounds/FrameMaximize.wav
```

```
//"InternalFrame.minimizeIcon",
javax.swing.plaf.metal.MetalIconFactory$InternalFrameAltMaximizeIcon@2808b3
//"InternalFrame.minimizeSound", sounds/FrameMinimize.wav
//"InternalFrame.optionDialogBorder",
javax.swing.plaf.metal.MetalBorders$OptionDialogBorder@11b9fb1
//"InternalFrame.paletteBorder",
javax.swing.plaf.metal.MetalBorders$PaletteBorder@1386000
//"InternalFrame.paletteCloseIcon",
javax.swing.plaf.metal.MetalIconFactory$PaletteCloseIcon@1f78ef1
//"InternalFrame.paletteTitleHeight", 11
//"InternalFrame.restoreDownSound", sounds/FrameRestoreDown.wav
//"InternalFrame.restoreUpSound", sounds/FrameRestoreUp.wav
//"InternalFrame.titleFont", font1,
//"InternalFrameUI", javax.swing.plaf.metal.MetalInternalFrameUI
        "Label.background", new ColorUIResource(0, 0, 0),
//"Label.disabledForeground", new ColorUIResource(0,0,0),
//"Label.disabledShadow", new ColorUIResource(0,0,0),
        "Label.font", font1,
        "Label.foreground", textColor,
//"LabelUI", javax.swing.plaf.metal.MetalLabelUI
        "List.background", comboBoxBackgroundColor,
//"List.cellRenderer",
javax.swing.DefaultListCellRenderer$UIResource[,0,0,0x0,invalid,alignmentX=0.0,alignmentY
=null,border=javax.swing.border.EmptyBorder@139eeda,flags=8,maximumSize=,minimumSize=,pre
ferredSize=,defaultIcon=,disabledIcon=,horizontalAlignment=LEADING,horizontalTextPosition
=TRAILING,iconTextGap=4,labelFor=,text=,verticalAlignment=CENTER,verticalTextPosition=CEN
TER]
//"List.focusCellHighlightBorder",
javax.swing.plaf.BorderUIResource$LineBorderUIResource@913fe2
//"List.focusInputMap", javax.swing.plaf.InputMapUIResource@147c5fc
//"List.focusInputMap.RightToLeft", javax.swing.plaf.InputMapUIResource@161d36b
//"List.font", font1,
        "List.foreground", comboBoxForegroundColor,
        "List.selectionBackground", comboBoxSelectionBackgroundColor,
        "List.selectionForeground", comboBoxSelectionForegroundColor,
//"ListUI", javax.swing.plaf.basic.BasicListUI
//"menu", new ColorUIResource(0,0,0),
//"Menu.acceleratorFont", font1,
//"Menu.acceleratorForeground", new ColorUIResource(0,0,0),
//"Menu.acceleratorSelectionForeground", new ColorUIResource(0,0,0),
//"Menu.arrowIcon", javax.swing.plaf.metal.MetalIconFactory$MenuArrowIcon@1e4457d
//"Menu.background", new ColorUIResource(0,0,0),
//"Menu.border", javax.swing.plaf.metal.MetalBorders$MenuItemBorder@b2fd8f
//"Menu.borderPainted", true
//"Menu.crossMenuMnemonic", true
//"Menu.disabledForeground", new ColorUIResource(0,0,0),
//"Menu.font", font1,
//"Menu.foreground", new ColorUIResource(0,0,0),
//"Menu.margin", javax.swing.plaf.InsetsUIResource[top=2,left=2,bottom=2,right=2]
//"Menu.menuPopupOffsetX", 0
//"Menu.menuPopupOffsetY", 0
//"Menu.selectionBackground", new ColorUIResource(0,0,0),
//"Menu.selectionForeground", new ColorUIResource(0,0,0),
//"Menu.shortcutKeys", [I@1292d26
//"Menu.submenuPopupOffsetX", -4
//"Menu.submenuPopupOffsetY", -3
//"MenuBar.background", new ColorUIResource(0,0,0),
//"MenuBar.border", javax.swing.plaf.metal.MetalBorders$MenuBarBorder@544ec1
//"MenuBar.font", font1,
//"MenuBar.foreground", new ColorUIResource(0,0,0),
//"MenuBar.highlight", new ColorUIResource(0,0,0),
//"MenuBar.shadow", new ColorUIResource(0,0,0),
//"MenuBar.windowBindings", [Ljava.lang.Object;@10a2d64
//"MenuBarUI", javax.swing.plaf.basic.BasicMenuBarUI
//"MenuItem.acceleratorDelimiter", -
//"MenuItem.acceleratorFont", font1,
//"MenuItem.acceleratorForeground", new ColorUIResource(0,0,0),
//"MenuItem.acceleratorSelectionForeground", new ColorUIResource(0,0,0),
//"MenuItem.arrowIcon", javax.swing.plaf.metal.MetalIconFactory$MenuItemArrowIcon@1e9cb75
//"MenuItem.background", new ColorUIResource(0,0,0),
//"MenuItem.border", javax.swing.plaf.metal.MetalBorders$MenuItemBorder@79717e
```

```
//"MenuItem.borderPainted", true
//"MenuItem.commandSound", sounds/MenuItemCommand.wav
//"MenuItem.disabledForeground", new ColorUIResource(0,0,0),
//"MenuItem.font", font1,
//"MenuItem.foreground", new ColorUIResource(0,0,0),
//"MenuItem.margin", javax.swing.plaf.InsetsUIResource[top=2,left=2,bottom=2,right=2]
//"MenuItem.selectionBackground", new ColorUIResource(0,0,0),
//"MenuItem.selectionForeground", new ColorUIResource(0,0,0),
//"MenuItemUI", javax.swing.plaf.basic.BasicMenuItemUI
//"menuText", new ColorUIResource(0,0,0),
//"MenuUI", javax.swing.plaf.basic.BasicMenuUI
//"OptionPane.background", new ColorUIResource(0,0,0),
//"OptionPane.border", javax.swing.plaf.BorderUIResource$EmptyBorderUIResource@1b09468
//"OptionPane.buttonAreaBorder",
javax.swing.plaf.BorderUIResource$EmptyBorderUIResource@e94e92
//"OptionPane.buttonClickThreshhold", 500
//"OptionPane.errorDialog.border.background", new ColorUIResource(0,0,0),
//"OptionPane.errorDialog.titlePane.background", new ColorUIResource(0,0,0),
//"OptionPane.errorDialog.titlePane.foreground", new ColorUIResource(0,0,0),
//"OptionPane.errorDialog.titlePane.shadow", new ColorUIResource(0,0,0),
//"OptionPane.errorIcon", javax.swing.plaf.IconUIResource@166a22b
//"OptionPane.errorSound", sounds/OptionPaneError.wav
//"OptionPane.font", font1,
//"OptionPane.foreground", new ColorUIResource(0,0,0),
//"OptionPane.informationIcon", javax.swing.plaf.IconUIResource@1f6f296
//"OptionPane.informationSound", sounds/OptionPaneInformation.wav
//"OptionPane.messageAreaBorder",
javax.swing.plaf.BorderUIResource$EmptyBorderUIResource@8bdcd2
//"OptionPane.messageForeground", new ColorUIResource(0,0,0),
//"OptionPane.minimumSize", javax.swing.plaf.DimensionUIResource[width=262,height=90]
//"OptionPane.questionDialog.border.background", new ColorUIResource(0,0,0),
//"OptionPane.questionDialog.titlePane.background", new ColorUIResource(0,0,0),
//"OptionPane.questionDialog.titlePane.foreground", new ColorUIResource(0,0,0),
//"OptionPane.questionDialog.titlePane.shadow", new ColorUIResource(0,0,0),
//"OptionPane.questionIcon", javax.swing.plaf.IconUIResource@3570b0
//"OptionPane.questionSound", sounds/OptionPaneQuestion.wav
//"OptionPane.warningDialog.border.background", new ColorUIResource(0,0,0),
//"OptionPane.warningDialog.titlePane.background", new ColorUIResource(0,0,0),
//"OptionPane.warningDialog.titlePane.foreground", new ColorUIResource(0,0,0),
//"OptionPane.warningDialog.titlePane.shadow", new ColorUIResource(0,0,0),
//"OptionPane.warningIcon", javax.swing.plaf.IconUIResource@62937c
//"OptionPane.warningSound", sounds/OptionPaneWarning.wav
//"OptionPane.windowBindings", [Ljava.lang.Object;@1e893df
//"OptionPaneUI", javax.swing.plaf.basic.BasicOptionPaneUI
        "Panel.background", panelBackground,
//"Panel.font", font1,
//"Panel.foreground", new ColorUIResource(0,0,0),
//"PanelUI", javax.swing.plaf.basic.BasicPanelUI

        "PasswordField.background", textFieldBackgroundColor,
        "PasswordField.border",
        BorderFactory.createMatteBorder(textFieldBorderSize, textFieldBorderSize,
textFieldBorderSize,
                                        textFieldBorderSize, textFieldBorderColor),
        "PasswordField.caretBlinkRate", new Integer(150),
        "PasswordField.caretForeground", textFieldForegroundColor,
//"PasswordField.focusInputMap", javax.swing.plaf.InputMapUIResource@a1807c
//"PasswordField.font", font1,
        "PasswordField.foreground", textFieldForegroundColor,
//"PasswordField.inactiveBackground", new ColorUIResource(0,0,0),
//"PasswordField.inactiveForeground", new ColorUIResource(0,0,0),
//"PasswordField.margin",
javax.swing.plaf.InsetsUIResource[top=0,left=0,bottom=0,right=0]
//"PasswordField.selectionBackground", new ColorUIResource(0,0,0),
//"PasswordField.selectionForeground", new ColorUIResource(0,0,0),
//"PasswordFieldUI", javax.swing.plaf.basic.BasicPasswordFieldUI
//"PopupMenu.background", new ColorUIResource(0,0,0),
//"PopupMenu.border", javax.swing.plaf.metal.MetalBorders$PopupMenuBorder@861f24
//"PopupMenu.font", font1,
//"PopupMenu.foreground", new ColorUIResource(0,0,0),
//"PopupMenu.popupSound", sounds/PopupMenuPopup.wav
```

```
//"PopupMenu.selectedWindowInputMapBindings", [Ljava.lang.Object;@1d4c61c
//"PopupMenu.selectedWindowInputMapBindings.RightToLeft", [Ljava.lang.Object;@121cc40
//"PopupMenuSeparatorUI", javax.swing.plaf.metal.MetalPopupMenuSeparatorUI
//"PopupMenuUI", javax.swing.plaf.basic.BasicPopupMenuUI
"ProgressBar.background", backgroundColor,
"ProgressBar.border", new
BorderUIResource(BorderFactory.createMatteBorder(1,1,1,1,borderColor2)),
//"ProgressBar.cellLength", new Integer(5),
//"ProgressBar.cellSpacing", new Integer(5),
//"ProgressBar.cycleTime", 3000
"ProgressBar.font", font1,
"ProgressBar.foreground", textFieldBorderColor,
//"ProgressBar.repaintInterval", 50
"ProgressBar.selectionBackground", textFieldBorderColor,
"ProgressBar.selectionForeground", backgroundColor,
//"ProgressBarUI", javax.swing.plaf.metal.MetalProgressBarUI
//"RadioButton.background", new ColorUIResource(0,0,0),
//"RadioButton.border", javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@3eca90
//"RadioButton.darkShadow", new ColorUIResource(0,0,0),
//"RadioButton.disabledText", new ColorUIResource(0,0,0),
//"RadioButton.focus", new ColorUIResource(0,0,0),
//"RadioButton.focusInputMap", javax.swing.plaf.InputMapUIResource@1f14ceb
//"RadioButton.font", font1,
//"RadioButton.foreground", new ColorUIResource(0,0,0),
//"RadioButton.highlight", new ColorUIResource(0,0,0),
//"RadioButton.icon", javax.swing.plaf.metal.MetalIconFactory$RadioButtonIcon@503429
//"RadioButton.light", new ColorUIResource(0,0,0),
//"RadioButton.margin", javax.swing.plaf.InsetsUIResource[top=2,left=2,bottom=2,right=2]
//"RadioButton.select", new ColorUIResource(0,0,0),
//"RadioButton.shadow", new ColorUIResource(0,0,0),
//"RadioButton.textIconGap", 4
//"RadioButton.textShiftOffset", 0
//"RadioButtonMenuItem.acceleratorFont", font1,
//"RadioButtonMenuItem.acceleratorForeground", new ColorUIResource(0,0,0),
//"RadioButtonMenuItem.acceleratorSelectionForeground", new ColorUIResource(0,0,0),
//"RadioButtonMenuItem.arrowIcon",
javax.swing.plaf.metal.MetalIconFactory$MenuItemArrowIcon@1e9cb75
//"RadioButtonMenuItem.background", new ColorUIResource(0,0,0),
//"RadioButtonMenuItem.border",
javax.swing.plaf.metal.MetalBorders$MenuItemBorder@17ee8b8
//"RadioButtonMenuItem.borderPainted", true
//"RadioButtonMenuItem.checkIcon",
javax.swing.plaf.metal.MetalIconFactory$RadioButtonMenuItemIcon@1815859
//"RadioButtonMenuItem.commandSound", sounds/MenuItemCommand.wav
//"RadioButtonMenuItem.disabledForeground", new ColorUIResource(0,0,0),
//"RadioButtonMenuItem.font", font1,
//"RadioButtonMenuItem.foreground", new ColorUIResource(0,0,0),
//"RadioButtonMenuItem.margin",
javax.swing.plaf.InsetsUIResource[top=2,left=2,bottom=2,right=2]
//"RadioButtonMenuItem.selectionBackground", new ColorUIResource(0,0,0),
//"RadioButtonMenuItem.selectionForeground", new ColorUIResource(0,0,0),
//"RadioButtonMenuItemUI", javax.swing.plaf.basic.BasicRadioButtonMenuItemUI
//"RadioButtonUI", javax.swing.plaf.metal.MetalRadioButtonUI
//"RootPane.colorChooserDialogBorder",
javax.swing.plaf.metal.MetalBorders$QuestionDialogBorder@13caecd
//"RootPane.defaultButtonWindowKeyBindings", [Ljava.lang.Object;@166aa18
//"RootPane.errorDialogBorder",
javax.swing.plaf.metal.MetalBorders$ErrorDialogBorder@11a64ed
//"RootPane.fileChooserDialogBorder",
javax.swing.plaf.metal.MetalBorders$QuestionDialogBorder@1174b07
//"RootPane.frameBorder", javax.swing.plaf.metal.MetalBorders$FrameBorder@18f6235
//"RootPane.informationDialogBorder",
javax.swing.plaf.metal.MetalBorders$DialogBorder@197a37c
//"RootPane.plainDialogBorder", javax.swing.plaf.metal.MetalBorders$DialogBorder@115273a
//"RootPane.questionDialogBorder",
javax.swing.plaf.metal.MetalBorders$QuestionDialogBorder@1f934ad
//"RootPane.warningDialogBorder",
javax.swing.plaf.metal.MetalBorders$WarningDialogBorder@19b5393
//"RootPaneUI", javax.swing.plaf.metal.MetalRootPaneUI
//"scrollbar", new ColorUIResource(0,0,0),
//"ScrollBar.allowsAbsolutePositioning", true
```

```
        "ScrollBar.background", backgroundColor,
        "ScrollBar.darkShadow", backgroundColor,
//"ScrollBar.focusInputMap", javax.swing.plaf.InputMapUIResource@17ce4e7
//"ScrollBar.focusInputMap.RightToLeft", javax.swing.plaf.InputMapUIResource@1c282a1
        "ScrollBar.foreground", textColor,
        "ScrollBar.highlight", backgroundColor,
//"ScrollBar.maximumThumbSize",
javax.swing.plaf.DimensionUIResource[width=4096,height=4096]
//"ScrollBar.minimumThumbSize", javax.swing.plaf.DimensionUIResource[width=8,height=8]
//"ScrollBar.shadow", new ColorUIResource(0,0,0),
        "ScrollBar.thumb", textFieldBorderColor,
//"ScrollBar.thumbDarkShadow", backgroundColor,
//"ScrollBar.thumbHighlight",backgroundColor,
//"ScrollBar.thumbShadow", textFieldBorderColor,
        "ScrollBar.track", panelBackground,
        "ScrollBar.trackHighlight", borderColor2,
        "ScrollBar.width", new Integer(14),
//"ScrollBarUI", javax.swing.plaf.metal.MetalScrollBarUI
//"ScrollPane.ancestorInputMap", javax.swing.plaf.InputMapUIResource@6e3d60
//"ScrollPane.ancestorInputMap.RightToLeft", javax.swing.plaf.InputMapUIResource@17f1ba3
//"ScrollPane.background", new ColorUIResource(0,0,0),
//"ScrollPane.border", javax.swing.plaf.metal.MetalBorders$ScrollPaneBorder@2a4983
//"ScrollPane.font", font1,
//"ScrollPane.foreground", new ColorUIResource(0,0,0),
//"ScrollPaneUI", javax.swing.plaf.metal.MetalScrollPaneUI
//"Separator.background", new ColorUIResource(0,0,0),
//"Separator.foreground", new ColorUIResource(0,0,0),
//"Separator.highlight", new ColorUIResource(0,0,0),
//"Separator.shadow", new ColorUIResource(0,0,0),
//"SeparatorUI", javax.swing.plaf.metal.MetalSeparatorUI
//"Slider.background", new ColorUIResource(0,0,0),
//"Slider.focus", new ColorUIResource(0,0,0),
//"Slider.focusInputMap", javax.swing.plaf.InputMapUIResource@1ef8cf3
//"Slider.focusInputMap.RightToLeft", javax.swing.plaf.InputMapUIResource@b66cc
//"Slider.focusInsets", javax.swing.plaf.InsetsUIResource[top=0,left=0,bottom=0,right=0]
//"Slider.foreground", new ColorUIResource(0,0,0),
//"Slider.highlight", new ColorUIResource(0,0,0),
//"Slider.horizontalThumbIcon",
javax.swing.plaf.metal.MetalIconFactory$HorizontalSliderThumbIcon@1786e64
//"Slider.majorTickLength", 6
//"Slider.shadow", new ColorUIResource(0,0,0),
//"Slider.trackWidth", 7
//"Slider.verticalThumbIcon",
javax.swing.plaf.metal.MetalIconFactory$VerticalSliderThumbIcon@bfea1d
//"SliderUI", javax.swing.plaf.metal.MetalSliderUI
//"Spinner.ancestorInputMap", javax.swing.plaf.InputMapUIResource@535b58
//"Spinner.arrowButtonBorder",
javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@1546e25
//"Spinner.arrowButtonInsets", java.awt.Insets[top=0,left=0,bottom=0,right=0]
//"Spinner.arrowButtonSize", java.awt.Dimension[width=16,height=5]
//"Spinner.background",F new ColorUIResource(0,0,0),
//"Spinner.border", javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@df8ff1
//"Spinner.editorBorderPainted", false
//"Spinner.font", font1,
//"Spinner.foreground", new ColorUIResource(0,0,0),
//"SpinnerUI", javax.swing.plaf.basic.BasicSpinnerUI
//"SplitPane.ancestorInputMap", javax.swing.plaf.InputMapUIResource@17a8a02
//"SplitPane.background", new ColorUIResource(0,0,0),
//"SplitPane.border", javax.swing.plaf.basic.BasicBorders$SplitPaneBorder@1662dc8
//"SplitPane.darkShadow", new ColorUIResource(0,0,0),
//"SplitPane.dividerSize", 10
//"SplitPane.highlight", new ColorUIResource(0,0,0),
//"SplitPane.shadow", new ColorUIResource(0,0,0),
//"SplitPaneDivider.border",
javax.swing.plaf.basic.BasicBorders$SplitPaneDividerBorder@1db7df8
//"SplitPaneUI", javax.swing.plaf.metal.MetalSplitPaneUI
//"Pane.ancestorInputMap", javax.swing.plaf.InputMapUIResource@cf40f5
//"TabbedPane.background", new ColorUIResource(0,0,0),
//"TabbedPane.contentBorderInsets",
javax.swing.plaf.InsetsUIResource[top=2,left=2,bottom=3,right=3]
        "TabbedPane.darkShadow", borderColor3,
```

```
//"TabbedPane.focusInputMap", javax.swing.plaf.InputMapUIResource@1aa57fb
        "TabbedPane.font", font1,
        "TabbedPane.foreground", textColor,
//"TabbedPane.highlight", new Color(0,200,0),
        "TabbedPane.light", borderColor1,
        "TabbedPane.focus", panelBackground,
        "TabbedPane.selected", panelBackground,
//"TabbedPane.selectedTabPadInsets",
javax.swing.plaf.InsetsUIResource[top=2,left=2,bottom=2,right=1]
        "TabbedPane.selectHighlight", borderColor1,
        "TabbedPane.shadow", borderColor2,
//"TabbedPane.tabAreaBackground", new ColorUIResource(200,0,0),
//"TabbedPane.tabAreaInsets",
javax.swing.plaf.InsetsUIResource[top=4,left=2,bottom=0,right=6]
//"TabbedPane.tabInsets",
javax.swing.plaf.InsetsUIResource[top=0,left=9,bottom=1,right=9]
//"TabbedPane.tabRunOverlay", 2
//"TabbedPane.textIconGap", 4
//"TabbedPaneUI", javax.swing.plaf.metal.MetalTabbedPaneUI
//"Table.ancestorInputMap", javax.swing.plaf.InputMapUIResource@5a9de6
//"Table.ancestorInputMap.RightToLeft", javax.swing.plaf.InputMapUIResource@b2002f
//"Table.background", new ColorUIResource(0,0,0),
//"Table.focusCellBackground", new ColorUIResource(0,0,0),
//"Table.focusCellForeground", new ColorUIResource(0,0,0),
//"Table.focusCellHighlightBorder",
javax.swing.plaf.BorderUIResource$LineBorderUIResource@1abab88
//"Table.font", font1,
//"Table.foreground", new ColorUIResource(0,0,0),
//"Table.gridColor", new ColorUIResource(0,0,0),
//"Table.scrollPaneBorder", javax.swing.plaf.metal.MetalBorders$ScrollPaneBorder@176c74b
//"Table.selectionBackground", new ColorUIResource(0,0,0),
//"Table.selectionForeground", new ColorUIResource(0,0,0),
//"TableHeader.background", new ColorUIResource(0,0,0),
//"TableHeader.cellBorder", javax.swing.plaf.metal.MetalBorders$TableHeaderBorder@1194a4e
//"TableHeader.font", font1,
//"TableHeader.foreground", new ColorUIResource(0,0,0),
//"TableHeaderUI", javax.swing.plaf.basic.BasicTableHeaderUI
//"TableUI", javax.swing.plaf.basic.BasicTableUI
//"text", new ColorUIResource(0,0,0),
        "TextArea.background", panelBackground,
//"TextArea.border", javax.swing.plaf.basic.BasicBorders$MarginBorder@1ee3914
//"TextArea.caretBlinkRate", 500
//"TextArea.caretForeground", new ColorUIResource(0,0,0),
//"TextArea.focusInputMap", javax.swing.plaf.InputMapUIResource@540408
        "TextArea.font", font1,
        "TextArea.foreground", textAreaForegroundColor,
//"TextArea.inactiveForeground", textFieldForegroundColor2,
//"TextArea.margin", javax.swing.plaf.InsetsUIResource[top=0,left=0,bottom=0,right=0]
        "TextArea.selectionBackground", panelBackground,
        "TextArea.selectionForeground", textFieldForegroundColor,
//"TextAreaUI", javax.swing.plaf.basic.BasicTextAreaUI
        "TextField.background", textFieldBackgroundColor,
        "TextField.border",
        BorderFactory.createMatteBorder(textFieldBorderSize, textFieldBorderSize,
textFieldBorderSize,
                                        textFieldBorderSize, textFieldBorderColor),
        "TextField.caretBlinkRate", new Integer(150),
        "TextField.caretForeground", textFieldForegroundColor,
//"TextField.darkShadow", new ColorUIResource(0,0,0),
//"TextField.focusInputMap", javax.swing.plaf.InputMapUIResource@14a8cd1
        "TextField.font", textFieldFont,
        "TextField.foreground", textFieldForegroundColor,
//"TextField.highlight", borderColor1,
//"TextField.inactiveBackground", new ColorUIResource(0,0,0),
        "TextField.inactiveForeground", textFieldForegroundColorInactive,
        "TextField.light", borderColor1,
//"TextField.margin", javax.swing.plaf.InsetsUIResource[top=0,left=0,bottom=0,right=0]
//"TextField.selectionBackground", new ColorUIResource(0,0,0),
//"TextField.selectionForeground", new ColorUIResource(0,0,0),
        "TextField.shadow", borderColor3,
//"TextFieldUI", javax.swing.plaf.metal.MetalTextFieldUI
```

```
//"textHighlight", new ColorUIResource(0,0,0),
//"textHighlightText", new ColorUIResource(0,0,0),
//"textInactiveText", new ColorUIResource(0,0,0),
//"TextPane.background", new ColorUIResource(0,0,0),
//"TextPane.border", javax.swing.plaf.basic.BasicBorders$MarginBorder@18a7efd
//"TextPane.caretBlinkRate", 500
//"TextPane.caretForeground", new ColorUIResource(0,0,0),
//"TextPane.focusInputMap", javax.swing.plaf.InputMapUIResource@1d6776d
//"TextPane.font", font1,
//"TextPane.foreground", new ColorUIResource(0,0,0),
//"TextPane.inactiveForeground", new ColorUIResource(0,0,0),
//"TextPane.margin", javax.swing.plaf.InsetsUIResource[top=3,left=3,bottom=3,right=3]
//"TextPane.selectionBackground", new ColorUIResource(0,0,0),
//"TextPane.selectionForeground", new ColorUIResource(0,0,0),
//"TextPaneUI", javax.swing.plaf.basic.BasicTextPaneUI
//"textText", new ColorUIResource(0,0,0),
//"TitledBorder.border", javax.swing.plaf.BorderUIResource$LineBorderUIResource@107ebe1
//"TitledBorder.font", font1,
//"TitledBorder.titleColor", new ColorUIResource(0,0,0),
//"ToggleButton.background", new ColorUIResource(0,0,0),
//"ToggleButton.border",
javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@1c39a2d
//"ToggleButton.darkShadow", new ColorUIResource(0,0,0),
//"ToggleButton.disabledText", new ColorUIResource(0,0,0),
//"ToggleButton.focus", new ColorUIResource(0,0,0),
//"ToggleButton.focusInputMap", javax.swing.plaf.InputMapUIResource@1f26605
//"ToggleButton.font", font1,
//"ToggleButton.foreground", new ColorUIResource(0,0,0),
//"ToggleButton.highlight", new ColorUIResource(0,0,0),
//"ToggleButton.light", new ColorUIResource(0,0,0),
//"ToggleButton.margin",
javax.swing.plaf.InsetsUIResource[top=2,left=14,bottom=2,right=14]
//"ToggleButton.select", new ColorUIResource(0,0,0),
//"ToggleButton.shadow", new ColorUIResource(0,0,0),
//"ToggleButton.textIconGap", 4
//"ToggleButton.textShiftOffset", 0
//"ToggleButtonUI", javax.swing.plaf.metal.MetalToggleButtonUI
//"ToolBar.ancestorInputMap", javax.swing.plaf.InputMapUIResource@1630ab9
//"ToolBar.background", new ColorUIResource(0,0,0),
//"ToolBar.border", javax.swing.plaf.metal.MetalBorders$ToolBarBorder@1f436f5
//"ToolBar.darkShadow", new ColorUIResource(0,0,0),
//"ToolBar.dockingBackground", new ColorUIResource(0,0,0),
//"ToolBar.dockingForeground", new ColorUIResource(0,0,0),
//"ToolBar.floatingBackground", new ColorUIResource(0,0,0),
//"ToolBar.floatingForeground", new ColorUIResource(0,0,0),
//"ToolBar.font", font1,
//"ToolBar.foreground", new ColorUIResource(0,0,0),
//"ToolBar.highlight", new ColorUIResource(0,0,0),
//"ToolBar.light", new ColorUIResource(0,0,0),
//"ToolBar.separatorSize", javax.swing.plaf.DimensionUIResource[width=10,height=10]
//"ToolBar.shadow", new ColorUIResource(0,0,0),
//"ToolBarSeparatorUI", javax.swing.plaf.basic.BasicToolBarSeparatorUI
//"ToolBarUI", javax.swing.plaf.metal.MetalToolBarUI
//"ToolTip.background", new ColorUIResource(0,0,0),
//"ToolTip.backgroundInactive", new ColorUIResource(0,0,0),
//"ToolTip.border", javax.swing.plaf.BorderUIResource$LineBorderUIResource@e5855a
//"ToolTip.borderInactive",
javax.swing.plaf.BorderUIResource$LineBorderUIResource@1e13d52
//"ToolTip.font", font1,
//"ToolTip.foreground", new ColorUIResource(0,0,0),
//"ToolTip.foregroundInactive", new ColorUIResource(0,0,0),
//"ToolTip.hideAccelerator", false
//"ToolTipUI", javax.swing.plaf.metal.MetalToolTipUI
//"Tree.ancestorInputMap", javax.swing.plaf.InputMapUIResource@15d56d5
//"Tree.background", new ColorUIResource(0,0,0),
//"Tree.changeSelectionWithFocus", true
//"Tree.closedIcon", javax.swing.plaf.metal.MetalIconFactory$TreeFolderIcon@12b3d53
//"Tree.collapsedIcon", javax.swing.plaf.metal.MetalIconFactory$TreeControlIcon@1975b59
//"Tree.drawsFocusBorderAroundIcon", false
//"Tree.editorBorder", javax.swing.plaf.BorderUIResource$LineBorderUIResource@1ef9f1d
//"Tree.expandedIcon", javax.swing.plaf.metal.MetalIconFactory$TreeControlIcon@78a212
```

```
//"Tree.focusInputMap", javax.swing.plaf.InputMapUIResource@1551f60
//"Tree.focusInputMap.RightToLeft", javax.swing.plaf.InputMapUIResource@27e353
//"Tree.font", font1,
//"Tree.foreground", new ColorUIResource(0,0,0),
//"Tree.hash", new ColorUIResource(0,0,0),
//"Tree.leafIcon", javax.swing.plaf.metal.MetalIconFactory$TreeLeafIcon@4e79f1
//"Tree.leftChildIndent", 7
//"Tree.line", new ColorUIResource(0,0,0),
//"Tree.openIcon", javax.swing.plaf.metal.MetalIconFactory$TreeFolderIcon@1ff7a1e
//"Tree.rightChildIndent", 13
//"Tree.rowHeight", 0
//"Tree.scrollsOnExpand", true
//"Tree.selectionBackground", new ColorUIResource(0,0,0),
//"Tree.selectionBorderColor", new ColorUIResource(0,0,0),
//"Tree.selectionForeground", new ColorUIResource(0,0,0),
//"Tree.textBackground", new ColorUIResource(0,0,0),
//"Tree.textForeground", new ColorUIResource(0,0,0),
//"TreeUI", javax.swing.plaf.metal.MetalTreeUI
//"Viewport.background", new ColorUIResource(0,0,0),
//"Viewport.font", font1,
//"Viewport.foreground", new ColorUIResource(0,0,0),
//"ViewportUI", javax.swing.plaf.basic.BasicViewportUI
//"window", new ColorUIResource(0,0,0),
//"windowBorder", new ColorUIResource(0,0,0),
//"windowText", new ColorUIResource(0,0,0),
    };

    /*        borderColor = new Color(0, 0, 0),
            //JTabbedPane
            tabbackground = new Color(58, 135, 135),
            tabforeground = new Color(158, 235, 235),
            //TextArea
            typedTextColor = new Color(18, 41, 87),
            textBackgroundColor = new Color(158, 235, 235),
            typedSelectedTextColor = new Color(150, 150, 150),
            textSelectedBackgroundColor = new Color(50, 50, 50),
            //TextField
            fieldtypedTextColor = new Color(158, 235, 235),
            fieldtextBackgroundColor = new Color(18, 41, 87),
            //Labels
            labelTextColor = new Color(18, 41, 87),
            labelBackgroundColor = backgroundColor,
            labelTextColor2 = new Color(0, 0, 0),
            labelBackgroundColor2 = new Color(255, 255, 255),
            //Queue
            queueActiveColor = new Color(80, 0, 20),
            queueInactiveColor = new Color(50, 50, 50),
            queueWriterColor = new Color(20, 80, 0),
            //Buttons
            buttonTextColor = new Color(0, 0, 0),
            buttonBackgroundColor = new Color(120, 120, 120),
            //Menu
            menuBackgroundColor = new Color(0, 0, 0),
            menuForegroundColor = new Color(200, 200, 200),
            //GroupList
            listForegroundColor = new Color(200, 200, 200),
            listBackgroundColor = new Color(10, 0, 40),
            listSelectBackgroundColor = new Color(10, 0, 80),
            //Border (GroupList)
            lightBorderColor = new Color(200, 200, 200);


         private final static Font typedTextFont = new Font("", Font.PLAIN, 14),
        labelTextFont = new Font("", Font.BOLD, 12),
        labelTextFont2 = new Font("", Font.BOLD, 12),
        menuTextFont = new Font("", Font.BOLD, 12),
        buttonTextFont = new Font("", Font.PLAIN, 12);
         private final static String picturePath = "/Gui/images/";
     */
    private final static int defaultXsize = 100
        , defaultYsize = 350;
```

```
public String getName()
{

    //Debugger.debug(getClass(),3,"getName");
    return "Definitions_LookAndFeel";
}

public String getDescription()
{
    //Debugger.debug(getClass(),3,"getDescription");
    return "The Definitions My Look and Feel";

}

public boolean isNativeLookAndFeel()
{
    return false;
}

public boolean isSupportedLookAndFeel()
{
    return true;
}

public String getID()
{
    return "Definitions_LookAndFeel";
}

protected void initClassDefaults(UIDefaults table)
{
    super.initClassDefaults(table);
    table.putDefaults(defaults);
}

protected void initComponentDefaults(UIDefaults table)
{
    super.initComponentDefaults(table);
    table.putDefaults(defaults);
}

protected static int getSpaceS()
{
    return smallSpace;
}

protected static int getSpaceL()
{
    return largeSpace;
}

protected static Color getBackground()
{
    return backgroundColor;
}

protected static JTextArea getTextArea()
{
    JTextArea T = new JTextArea();
    T.setEditable(false);
    T.setLineWrap(true);
    T.setWrapStyleWord(true);
    T.setCursor(MainFrame.cursor);
    return T;
}

//The main method generates the color properties code for the top of this file.
public static void main(String[] a)
{
```

```
        UIDefaults uiDefaults = UIManager.getDefaults();
        Enumeration en = uiDefaults.keys();
        String temp = "";
        Vector v = new Vector();
        while(en.hasMoreElements())
        {
            Object key = en.nextElement();
            Object value = uiDefaults.get(key);
            if(value != null)
            {
                if(value.toString().indexOf("ColorUIResource") != -1)
                {
                    temp = "\", new ColorUIResource(0,0,0),";
                }
                else if(value.toString().indexOf("FontUIResource") != -1)
                {
                    temp = "\", font1,";
                }

                else
                    temp = "\", " + value.toString();
                v.add("//\"" + key.toString() + temp);
            }
        }

        Object[] thearray = v.toArray();

        Arrays.sort(thearray, new Comparator()
        {
            public int compare(Object o1, Object o2)
            {
                return(((String) o1).toLowerCase().
                       compareTo(((String) o2).toLowerCase()));
            }
        });

    }


}
```

10.1.4.d DepositPanel

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.event.*;
import SpamCash.Utilities.Debugger;
import java.awt.BorderLayout;
import SpamCash.Utilities.Config;

public class DepositPanel extends JPanel
{
    private MainFrame frame;

    private final String informationtext = "Please enter the number of coins you wish to
deposit.";
    private final String[] labels =
        {"Amount ", "Login name ", " Password "};

    private final String CESLabel = "Currency server information:";

    private FormPanel logonInformationPanel;
    private AmountPanel amountPanel;
    private JTextArea informationTextArea;
    private Class className;

    protected DepositPanel(MainFrame frame)
    {
        this.frame = frame;
```

```
        className = getClass();

        ActionListener listener = new textFieldListener();
        informationTextArea = Definitions.getTextArea();
        informationTextArea.setText(informationtext);
        amountPanel = new AmountPanel(labels[0],listener,true,1);
        logonInformationPanel = new FormPanel(new String[]
                                              {labels[1], labels[2]}
                                              , new boolean[]
                                              {false, true}
                                              , listener);

        logonInformationPanel.textFields[0].setText(Config.defaultCESUsername);
        logonInformationPanel.textFields[1].setText(Config.defaultCESPassword);

        ImageButton button = new ImageButton("gradient", "Deposit...");
        button.addActionListener(listener);

        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        add(informationTextArea);
        add(Box.createVerticalStrut(Definitions.getSpaceL()));
        add(amountPanel);
        add(Box.createVerticalStrut(Definitions.getSpaceL()));
        add(new JLabel(CESLabel));
        add(Box.createVerticalStrut(Definitions.getSpaceS()));
        add(logonInformationPanel);
        add(Box.createVerticalStrut(Definitions.getSpaceL()));
        add(button);

        setBorder(BorderFactory.createMatteBorder(Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                  Definitions.getSpaceL(),
Definitions.getSpaceL(), getBackground()));

    }



    private String checkFields()
    {
        if(amountPanel.getText().length() == 0)
            return "Please fill in the field \"" + labels[0] + "\"";

        if(logonInformationPanel.textFields[0].getText().length() == 0)
            return "Please fill in the field \"" + labels[1] + "\"";

        if(logonInformationPanel.textFields[1].getText().length() == 0)
            return "Please fill in the field \"" + labels[2] + "\"";
        return "";
    }

    private class textFieldListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            try
            {
                //Filled in correctly?
                String check = checkFields();
                if(check.length() != 0)
                {
                    frame.showError(check);
                    return;
                }

                frame.deposit(Integer.parseInt(amountPanel.getText()),
                              logonInformationPanel.textFields[0].getText(),
                              logonInformationPanel.textFields[1].getText());
            }
            catch(Exception ex)
            {
```

```
                Debugger.debug(className, 3, "Could not deposit.", ex);
            }
        }
    }
}
```

10.1.4.e ErrorDialog

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class ErrorDialog extends JDialog
{
    private ImageButton okButton;

    //The textarea containing the text of the dialog
    private JTextArea info;

    //An errormessage upon a JFrame
    ErrorDialog(JFrame parent) throws Exception
    {
        super(parent,"Error",true);
        init();
    }
    //An errormessage upon a JDialog (LogonDialog)
    ErrorDialog(JDialog parent) throws Exception
    {
        super(parent,"Error",true);
        init();
    }

    private void init() throws Exception
    {
        super.dialogInit();
        getContentPane().add(new backgroundPanel());
        setCursor(MainFrame.cursor);
        setSize(500, 300);
        setResizable(true);
        center();
    }

    private class backgroundPanel extends JPanel
    {
        public backgroundPanel()
        {
            info = Definitions.getTextArea();
            JScrollPane infoScroll = new JScrollPane(info);
            infoScroll.setWheelScrollingEnabled(true);

infoScroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED );
            okButton = new ImageButton("gradient", "Ok");
            okButton.addActionListener(new listener());
            JPanel okPanel = new JPanel();
            okPanel.add(okButton);
            setLayout(new BorderLayout());
            add(infoScroll, "Center");
            add(okPanel, "South");
            setBorder(BorderFactory.createMatteBorder(Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                      Definitions.getSpaceL(),
Definitions.getSpaceL(),getBackground()));
        }
    }

    public void showError(String message)
    {
        //setSize(440,message.length()*5/13+125);
```

```
        //center();
        super.setTitle("Error");
        info.setText(message);
        show();
    }

    public void showMessage(String message)
    {
        //setSize(440,message.length()*5/13+125);
        //center();
        super.setTitle("Information");
        info.setText(message);
        show();
    }

    //Centers the window on the screen
    private void center()
    {
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = getSize();
        int x = ( screenSize.width - frameSize.width)/2;
        int y = ( screenSize.height - frameSize.height)/2;
        setLocation(x,y);
    }

    private class listener extends WindowAdapter implements ActionListener
    {
        public void windowClosing(WindowEvent e)
        {
            hide();
        }

        public void actionPerformed(ActionEvent e)
        {
            hide();
        }
    }

}
```

10.1.4.f FormPanel

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.event.ActionListener;
import java.awt.*;

public class FormPanel extends JPanel
{
    private String[] strings;
    private JLabel[] labels;
    protected JTextField[] textFields;
    private JPanel[] panels;

    public FormPanel(String[] strings, boolean[] password,ActionListener listener)
    {
        this.strings = new String[strings.length];
        labels = new JLabel[strings.length];
        textFields = new JTextField[strings.length];
        panels = new JPanel[strings.length];
        JPanel temp;

        setLayout(new BoxLayout(this,BoxLayout.Y_AXIS));

        for(int i=0;i<strings.length;i++)
        {
            this.strings[i] = strings[i];
            labels[i] = new JLabel(strings[i]);
```

```
            labels[i].setHorizontalTextPosition(JLabel.RIGHT);
            //Should the typed text be hidden in this field?
            if(password[i])
                textFields[i] = new JPasswordField();
            else
                textFields[i] = new JTextField();
            textFields[i].addActionListener(listener);

            panels[i] = new JPanel();
            panels[i].setLayout(new GridLayout(1, 2));
            temp = new JPanel();
            temp.setLayout(new BorderLayout());
            temp.add(labels[i],"East");
            panels[i].add(temp);
            panels[i].add(textFields[i]);
            panels[i].setPreferredSize(new Dimension(4000,18));
            panels[i].setMaximumSize(new Dimension(4000,18));


            add(panels[i]);
            add(Box.createVerticalStrut(Definitions.getSpaceS()));
        }
    }

    public void clear()
    {
        for(int i=0;i<strings.length;i++)
        {
            textFields[i].setText("");
        }
    }


}
```

10.1.4.g ImageButton

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
import java.awt.font.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.event.EventListenerList;
import SpamCash.Utilities.Config;

public class ImageButton extends JPanel
{
    private final String[] buttonTypeNames =
        {"_normal", "_pressed", "_point"};
    private JLabel buttonLabel;
    private ImageIcon[] buttonIcons;
    private String buttonName, buttonText;

    //Graphics properties
    private int buttonWidth = 0;
    private int buttonHeight = 0;
    private Font font;
    private final int space = 5;
    private final double percentOfHeight = 0.40;
    private final Color textColor = Definitions.buttonTextColor;

    private boolean setTextCoords = false;
    private int x = 0, y = 0;

    private boolean pressed = false;

    public ImageButton(String buttonName, String buttonText, int buttonHeight)
```

```
    {
        this.buttonHeight = buttonHeight;
        initImageButton(buttonName, buttonText);
    }

    public ImageButton(String buttonName, String buttonText, int buttonHeight, int
buttonWidth)
    {
        this.buttonHeight = buttonHeight;
        this.buttonWidth = buttonWidth;
        initImageButton(buttonName, buttonText);
    }


    public ImageButton(String buttonName, String buttonText)
    {
        initImageButton(buttonName, buttonText);
    }

    public void initImageButton(String buttonName, String buttonText)
    {
        this.buttonName = buttonName;
        this.buttonText = buttonText;

        buttonIcons = new ImageIcon[buttonTypeNames.length];

        for(int t = 0; t < buttonTypeNames.length; t++)
            buttonIcons[t] = new
ImageIcon(Toolkit.getDefaultToolkit().getImage(Config.imagedir
                                                                        +
System.getProperty("file.separator")
                                                                        +
buttonName + buttonTypeNames[t]
                                                                        +
".gif"));

        buttonLabel = new JLabel(buttonIcons[0]);

        if(buttonHeight == 0)
        {
            buttonHeight =
buttonIcons[0].getImage().getHeight(buttonIcons[0].getImageObserver());
        }
        buttonWidth =
buttonIcons[0].getImage().getWidth(buttonIcons[0].getImageObserver());


        //At least 10 in fontsize
        font = new Font("font", Font.PLAIN,
                        (int) (buttonHeight * percentOfHeight) < 12 ? 12 : (int)
(buttonHeight * percentOfHeight));

        setLayout(new BorderLayout());
        setPreferredSize(new Dimension(buttonWidth, buttonHeight));
        setMaximumSize(new Dimension(buttonWidth, buttonHeight));
        add(buttonLabel, "Center");
        addMouseListener(new buttonListener());
    }

    public void paint(Graphics g)
    {
        g.fillRect(0, 0, getWidth(), getHeight());
        this.paintChildren(g);
        g.setColor(textColor);
        g.setFont(font);

        if(!setTextCoords)
        {
            FontMetrics metrics = g.getFontMetrics();
            Rectangle2D bounds = metrics.getStringBounds(buttonText, g);
            x = (buttonWidth - (int) bounds.getWidth()) / 2;
```

```
            y = (buttonHeight - (int) bounds.getHeight()) / 2 + (int) bounds.getHeight()
- metrics.getDescent();
            setTextCoords = true;
        }
        g.drawString(buttonText, x, y);
    }

    private class buttonListener implements MouseListener
    {
        public void mousePressed(MouseEvent e)
        {
            buttonLabel.setIcon(buttonIcons[1]);
            pressed = true;
        }

        public void mouseReleased(MouseEvent e)
        {

            pressed = false;

            if(e.getX() < buttonWidth && e.getX() > 0 && e.getY() < buttonHeight &&
e.getY() > 0)
            {
                buttonLabel.setIcon(buttonIcons[2]);
                fireAction(new ActionEvent(this, 1, buttonText));
            }
            else
                buttonLabel.setIcon(buttonIcons[0]);
        }

        public void mouseClicked(MouseEvent e)
        {
        }

        public void mouseEntered(MouseEvent e)
        {
            if(pressed)
                buttonLabel.setIcon(buttonIcons[1]);
            else
                buttonLabel.setIcon(buttonIcons[2]);
        }

        public void mouseExited(MouseEvent e)
        {
            buttonLabel.setIcon(buttonIcons[0]);
        }
    }

    EventListenerList actionListeners = new EventListenerList();

    public void addActionListener(ActionListener listener)
    {
        actionListeners.add(ActionListener.class, listener);
    }

    public void removeActionListener(ActionListener listener)
    {
        actionListeners.remove(ActionListener.class, listener);
    }

    protected void fireAction(ActionEvent actionEvent)
    {
        Object[] listeners = actionListeners.getListenerList();

        // loop through each listener and pass on the event
        for(int i = 0; i < listeners.length; i++)
        {
            if(listeners[i] instanceof ActionListener)
                ((ActionListener) listeners[i]).actionPerformed(actionEvent);
        }
    }
```

```
}
```

10.1.4.h MainFrame

```
package SpamCash.Client.GUI;


import SpamCash.Utilities.Config;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

import SpamCash.Utilities.Debugger;
import SpamCash.Utilities.*;
import SpamCash.Client.Control;
import SpamCash.Currency.Coin;
import SpamCash.Client.MailHandler.CoinHeader;

public class MainFrame extends JFrame
{
    public static Cursor cursor;
    public static ImageIcon cornerImage;

    private OpenSafeDialog safeDialog;
    private ConfigurationPanel configPanel;
    private SpamPanel spamPanel;
    private String safeKeyFilePath = "", safePassword = "";
    private boolean safeInfoExists = false;
    private static ErrorDialog errorDialog;
    private RegistrationFrame registrationDialog;
    private SplashScreen splash;
    private Control control;

    public MainFrame()
    {
        splash = new SplashScreen();
        //Look and feel
        try
        {
            UIManager.setLookAndFeel(
                "SpamCash.Client.GUI.Definitions");
        }
        catch(Exception e)
        {
            Debugger.debug(getClass(), 1, "Couldn't load graphics.");
        }
    }

    public void init(Control control)
    {
        this.control = control;
        try
        {
            //The Cursor
            ImageIcon cursorImage = new
ImageIcon(Toolkit.getDefaultToolkit().getImage(Config.imagedir
                                                                                +
System.getProperty(
                "file.separator")
                                                                                +
"cursor.gif"));
            Toolkit toolKit = Toolkit.getDefaultToolkit();
            cursor = toolKit.createCustomCursor(cursorImage.getImage(), new Point(0, 0),
"cursor");
            setCursor(cursor);
            cornerImage = new
ImageIcon(Toolkit.getDefaultToolkit().getImage(Config.imagedir
                                                                                +
System.getProperty(
```

```
                "file.separator")
                                                                                +
"corner.gif"));
            setIconImage(cornerImage.getImage());

            UIDefaults defaults = UIManager.getDefaults();
            defaults.put("TabbedPane.background", Definitions.getBackground());

            safeDialog = new OpenSafeDialog(this);
            errorDialog = new ErrorDialog(this);
            registrationDialog = new RegistrationFrame(this,control);

            addWindowListener(new WindowIsClosing());
            setSize(550, 600);
            center();
            setTitle(Config.name);

            //Add the components
            JTabbedPane pane = new JTabbedPane();
            pane.setOpaque(true);

            pane.addChangeListener(new changeListener());
            configPanel = new ConfigurationPanel(this);
            pane.addTab("Configuration", configPanel);
            pane.addTab("Bank Deposit", new DepositPanel(this));
            pane.addTab("Bank Withdraw", new WithdrawPanel(this));
            spamPanel = new SpamPanel(this);
            pane.addTab("SPAM Withdraw", spamPanel);
            pane.addTab("Whitelist", new WhitelistPanel(control,this));
            pane.setBorder(BorderFactory.createMatteBorder(Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                           Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                           Definitions.getBackground()));


            getContentPane().setLayout(new BorderLayout());
            getContentPane().add(pane, "Center");
            getContentPane().add(new buttonPanel(),"South");

            setVisible(false);
            setResizable(true);
        }
        catch(Exception ex)
        {
            Debugger.debug(getClass(), 1, "Initialization error.", ex);
        }
    }

    private class buttonPanel extends JPanel

    {
        public buttonPanel()
        {

            ImageButton exitButton = new ImageButton("gradient","Exit");
            exitButton.addActionListener(new listener());

            ImageButton hideButton = new ImageButton("gradient","Hide");
            hideButton.addActionListener(new listener());


            ImageButton statusButton = new ImageButton("gradient","Money Status");
            statusButton.addActionListener(new listener());


            //ImageButton generateButton = new ImageButton("gradient","Generate Key-
file");
            //generateButton.addActionListener(new listener());

            setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
```

```
        add(Box.createHorizontalGlue());
        add(statusButton);
        add(Box.createHorizontalStrut(5));
        //add(generateButton);
        //add(Box.createHorizontalStrut(5));
        add(hideButton);
        add(Box.createHorizontalStrut(5));
        add(exitButton);
        add(Box.createHorizontalGlue());
        setBorder(BorderFactory.createMatteBorder(0,0,Definitions.getSpaceL(),0,
                                                  Definitions.getBackground()));
        setBackground(Definitions.getBackground());
    }
}

public void showRegistrationDialog()
{
    splash.hide();
    registrationDialog.show();
}

public void hideRegistrationDialog()
{
    splash.hide();
    registrationDialog.hide();
}

public void show()
{
    splash.hide();
    super.show();
}

public void showError(String message)
{
    show();
    errorDialog.showError(message);
}

public void showMessage(String message)
{
    show();
    errorDialog.showMessage(message);
}

protected void collectCoin(MailInformation m) throws Exception
{
    control.collectCoin(m);
}

protected void saveConfigurationInfo(ProxyConfiguration config)
{
    Debugger.debug(getClass(), 1, "Updating the configuration file...");
    try
    {
        control.saveConfiguration(config);
    }
    catch(Exception e)
    {
        Debugger.debug(getClass(), 1, "Error updating config file.",e);
        errorDialog.showError("The configuration could not be saved.");
    }
    showMessage("The configuration has been saved.");
    Debugger.debug(getClass(), 1, "Updated the configuration file succesfully.");
}

protected ProxyConfiguration loadConfigurationInfo() throws Exception
{
    Debugger.debug(getClass(), 1, "Loading the configuration file...");
    return control.loadConfiguration();
}
```

```
    public void updateMailList()
    {
        if(safeInfoExists && control.getSafe().isOpen())
        {
            spamPanel.update(control.getSafe());
        }
        else
            safeDialog.show(OpenSafeDialog.UPDATE_ID, 0,null,null);
    }

    public void withdraw(double amount, String CESLogonName, String CESpassword)
    {
        if(safeInfoExists && control.getSafe().isOpen())
        {
            Debugger.debug(getClass(), 1, "Making withdrawal using:");
            Debugger.debug(getClass(), 1, "   amount: " + amount);
            Debugger.debug(getClass(), 1, "   CES login name: " + CESLogonName);
            Debugger.debug(getClass(), 1, "   CES password: " + CESpassword);
            control.withdraw(CESLogonName, CESpassword, amount);
            Debugger.debug(getClass(), 1, "Withdrawal done.");
        }
        else
            safeDialog.show(OpenSafeDialog.WITHDRAW_ID, amount, CESLogonName,
CESpassword);
    }

    protected void deposit(int numberOfCoins, String CESLogonName, String CESpassword)
    {
        if(safeInfoExists && control.getSafe().isOpen())
        {
            Debugger.debug(getClass(), 1, "Making deposit using:");
            Debugger.debug(getClass(), 1, "   number of coins: " + numberOfCoins);
            Debugger.debug(getClass(), 1, "   CES login name: " + CESLogonName);
            Debugger.debug(getClass(), 1, "   CES password: " + CESpassword);
            control.deposit(CESLogonName, CESpassword, numberOfCoins);
            Debugger.debug(getClass(), 1, "Deposit done.");
        }
        else
            safeDialog.show(OpenSafeDialog.DEPOSIT_ID, numberOfCoins, CESLogonName,
CESpassword);
    }

    private class changeListener implements ChangeListener
    {
        public void stateChanged(ChangeEvent e)
        {
            //Debugger.debug(getClass(),3,"StateChanged source = " +
((JTabbedPane)e.getSource()).getSelectedIndex());

            //The configurationTab has been selected
            if(((JTabbedPane) e.getSource()).getSelectedIndex() == 0)
            {
                Debugger.debug(getClass(), 1, "Loading config...");
                try
                {
                    configPanel.setConfigInfo(loadConfigurationInfo());
                }
                catch(Exception ex)
                {
                    Debugger.debug(getClass(), 1, "The configuration file could not be
loaded.");
                    //errorDialog.showError("The configuration file could not be
loaded.");
                }
                Debugger.debug(getClass(), 1, "Loading config done.");
            }
            //The SPAMTab has been selected
            else if(((JTabbedPane) e.getSource()).getSelectedIndex() == 3)
            {
                updateMailList();
```

```
        }
    }
}

public void tryToOpenSafe(int id)
{
    try
    {
        safeDialog.show(id);
    }
    catch(Exception e)
    {
        Debugger.debug(getClass(),1,e.getMessage());
    }
}

protected void openSafe(String safeKeyFilePath, String safePassword) throws Exception
{
    Debugger.debug(getClass(), 1, "Opening safe using...");
    Debugger.debug(getClass(), 1, "   Filepath: " + safeKeyFilePath);

    Debugger.debug(getClass(), 1, "   Password: " + safePassword);

    control.openSafe(safeKeyFilePath, safePassword);
    this.safeKeyFilePath = safeKeyFilePath;
    this.safePassword = safePassword;

    Debugger.debug(getClass(), 1, "   Successful.");
    safeInfoExists = true;
}

protected void wakeUpControlThreads()
{
    control.wakeUptheSleeping();
}

protected void showStatus()
{
    showMessage("Money status:\n " + getSafeStatus());
}

protected String getSafeStatus()
{
    return control.getSafe().toString();
}

//Centers the window on the screen
private void center()
{
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = getSize();
    int x = (screenSize.width - frameSize.width) / 2;
    int y = (screenSize.height - frameSize.height) / 2;
    setLocation(x, y);
}

private void shutdown()
{
    control.shutdown();
}

private class listener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        if(e.getActionCommand().equals("Money Status"))
        {
            Debugger.debug(getClass(),3,"Money status...");
            if(safeInfoExists && control.getSafe().isOpen())
            {
                showStatus();
```

```
                }
                else
                    safeDialog.show(OpenSafeDialog.SAFE_STATUS_ID, 0.0,null,null);

            }
            else if(e.getActionCommand().equals("Generate Key-file"))
            {
                Debugger.debug(getClass(),3,"Generate pressed...");
                String s="";
                try
                {
                    s = control.generateKeyFile(Config.defaultSafeKeyFilePath);
                }
                catch(Exception ex)
                {
                    errorDialog.showError("A new key for the safe could not be
generated.");
                    return;
                }
               showMessage("A new key for the safe has been generated and saved in the
file: "+s);
            }
            else if(e.getActionCommand().equals("Hide"))
                setState(Frame.ICONIFIED);
            else
                shutdown();
        }
    }

    private class WindowIsClosing extends WindowAdapter
    {
        public void windowClosing(WindowEvent e)
        {
            shutdown();
        }
    }
}
```

10.1.4.i OpenSafeDialog

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;

import SpamCash.Utilities.Debugger;
import SpamCash.Utilities.*;

public class OpenSafeDialog extends JDialog
{
    private MainFrame frame;
    private final String informationtext =
        "Please provide the path for the key-file and the password to access the safe-
file. This is needed to get access to the e-cash and the private-key.";
    private final String[] labels =
        {"Path of the key-file ", "Password "};

    private final String[] buttonTypeNames =
        {"_normal", "_pressed", "_point"};
    private FormPanel logonPanel;
    private TopPanel topPanel;
    private JTextArea informationTextArea;

    private double amount;

    //To specify if it is a deposit or withdrawal
```

```
    private int jobID;
    private String CESLogonName, CESpassword;

    private ErrorDialog errorDialog;

    public final static int DEPOSIT_ID = 1, WITHDRAW_ID = 2, SAFE_STATUS_ID = 3,
UPDATE_ID = 4, CONTROL_THREAD = 5;

    protected OpenSafeDialog(MainFrame frame) throws Exception
    {
        super(frame, "Safe access", true);
        super.dialogInit();

        this.frame = frame;

        setCursor(MainFrame.cursor);
        getContentPane().add(new backgroundPanel());
        setSize(500, 230);
        setResizable(false);
        center();
        errorDialog = new ErrorDialog(this);
    }

    private class backgroundPanel extends JPanel
    {
        public backgroundPanel()
        {
            ActionListener listener = new actionListener();

            informationTextArea = Definitions.getTextArea();
            informationTextArea.setText(informationtext);

            ImageButton okButton = new ImageButton("gradient", "Ok");
            okButton.addActionListener(listener);
            ImageButton cancelButton = new ImageButton("gradient", "Cancel");
            cancelButton.addActionListener(listener);

            JPanel buttonPanel = new JPanel();
            buttonPanel.add(okButton);
            buttonPanel.add(cancelButton);

            topPanel = new TopPanel(listener);

            setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
            add(informationTextArea);
            add(Box.createVerticalStrut(Definitions.getSpaceL()));
            add(topPanel);
            add(Box.createVerticalStrut(Definitions.getSpaceL()));
            add(buttonPanel);

            setBorder(BorderFactory.createMatteBorder(Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                      Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                      getBackground()));

        }
    }

    private class TopPanel extends JPanel
    {
        public TopPanel(ActionListener listener)
        {
            logonPanel = new FormPanel(new String[]
                                       {labels[0], labels[1]}
                                       , new boolean[]
                                       {false, true}
                                       , listener);
            ImageButton browseButton = new ImageButton("gradient_medium", "Browse...");
            browseButton.addActionListener(listener);
            JPanel browsePanel = new JPanel();
```

```
            browsePanel.setLayout(new BorderLayout());
            browsePanel.add(browseButton, "North");
            setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
            add(logonPanel);
            add(Box.createHorizontalStrut(Definitions.getSpaceS()));
            add(browsePanel);
        }
    }

    public void show(int jobID) throws Exception
    {
        //frame.show();
        //jobID indocates wether a deposit,withdrawal or sendmail is underway...
        if(jobID == this.WITHDRAW_ID || jobID == this.DEPOSIT_ID)
            throw new Exception("Necessary information is missing. Cannot show
safedialog.");
        logonPanel.clear();
        logonPanel.textFields[0].setText(Config.defaultSafeKeyFilePath);
        logonPanel.textFields[1].setText(Config.defaultLocalPassword);
        this.jobID = jobID;
        super.show();
    }

    public void show(int jobID, double amount, String CESLogonName, String CESpassword)
    {
        //jobID indocates wether a deposit,withdrawal or sendmail is underway...

        frame.show();
        logonPanel.clear();
        logonPanel.textFields[0].setText(Config.defaultSafeKeyFilePath);
        logonPanel.textFields[1].setText(Config.defaultLocalPassword);
        this.jobID = jobID;
        this.amount = amount;
        this.CESLogonName = CESLogonName;
        this.CESpassword = CESpassword;
        super.show();
    }


    private String checkFields()
    {
        if(logonPanel.textFields[0].getText().length() == 0)
            return "Please fill in the field \"" + labels[0] + "\"";

        if(logonPanel.textFields[1].getText().length() == 0)
            return "Please fill in the field \"" + labels[1] + "\"";
        return "";
    }

    protected void showError(String message)
    {
        errorDialog.showError(message);
    }

    private class actionListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            //The cancel button?
            if(e.getActionCommand().equals("Cancel"))
            {
                hide();
                return;
            }
            else if(e.getActionCommand().equals("Browse..."))
            {
                //Debugger.debug(getClass(),3,"Browse...");
                JFileChooser fc = new JFileChooser();
                int returnVal = fc.showOpenDialog(frame);

                if(returnVal == JFileChooser.APPROVE_OPTION)
```

```
                {
                    //Update the path in the textfield
logonPanel.textFields[0].setText(fc.getSelectedFile().getAbsolutePath());
                }
                return;
            }

            //Filled in correctly?
            String check = checkFields();
            if(check.length() != 0)
            {
                showError(check);
                //Debugger.debug(getClass(),3,"Typing error: "+check);
                return;
            }


            try
            {
                //deposit
                if(jobID == DEPOSIT_ID)
                {
                    frame.openSafe(logonPanel.textFields[0].getText(),
                                   logonPanel.textFields[1].getText());
                    frame.deposit((int) amount, CESLogonName, CESpassword);
                }
                //Withdrawal
                else if(jobID == WITHDRAW_ID)
                {
                    frame.openSafe(logonPanel.textFields[0].getText(),
                                   logonPanel.textFields[1].getText());
                    frame.withdraw(amount, CESLogonName, CESpassword);
                }
                //Status
                else if(jobID == SAFE_STATUS_ID)

                {
                    frame.openSafe(logonPanel.textFields[0].getText(),
                                   logonPanel.textFields[1].getText());
                    frame.showStatus();
                }
                else if(jobID == UPDATE_ID)
                {
                    frame.openSafe(logonPanel.textFields[0].getText(),
                                   logonPanel.textFields[1].getText());
                    frame.updateMailList();
                }

                else if(jobID == CONTROL_THREAD)
                {
                    frame.openSafe(logonPanel.textFields[0].getText(),
                                   logonPanel.textFields[1].getText());
                    frame.wakeUpControlThreads();
                }
                else
                {
                    Debugger.debug(getClass(), 3, " Unknown jobID !!!");
                }
            }
            catch(Exception ex)
            {
                errorDialog.showError("Could not open safe. "+ex.getMessage());
                return;
            }
            hide();

        }
    }

    private void center()
```

```
    {
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = getSize();
        int x = (screenSize.width - frameSize.width) / 2;
        int y = (screenSize.height - frameSize.height) / 2;
        setLocation(x, y);
    }
}
```

10.1.4.j Progress

```
package SpamCash.Client.GUI;


import SpamCash.Utilities.Debugger;
import javax.swing.*;
import java.awt.*;
import SpamCash.Utilities.Config;

public class Progress extends JWindow
{
    private double totalSize;
    private double passed;
    private double numberOfProgreesSteps;
    private double currentStep;

    private JProgressBar progress;
    private JLabel label1,label2,label3;
    private String print;
    private ImageIcon cornerImage;

    public Progress()
    {
        //Look and feel
        try
        {
            UIManager.setLookAndFeel(
                "SpamCash.Client.GUI.Definitions");
        }
        catch(Exception e)
        {
            Debugger.debug(getClass(), 1, "Couldn't load graphics.");
        }

        cornerImage = new ImageIcon(Toolkit.getDefaultToolkit().getImage(Config.imagedir
                                                                         + System.getProperty(
    "file.separator")
                                                                         + "corner.gif"));


        Container c = getContentPane();
        JPanel progressPanel = new ProgressPanel();
        JPanel outerPanel = new JPanel();
        outerPanel.setBorder(BorderFactory.createMatteBorder(1,1,1,1,new
Color(200,200,200)));
        outerPanel.setLayout(new BorderLayout());
        outerPanel.add(progressPanel,"Center");
        c.add(outerPanel);
        setSize(220,110);
        position();
    }

    private class ProgressPanel extends JPanel
    {
        public ProgressPanel()
        {
            label1 = new JLabel(Config.name,cornerImage,JLabel.LEFT);
            label2 = new JLabel("",JLabel.CENTER);
            label3 = new JLabel("",JLabel.CENTER);
```

```
            progress = new JProgressBar();
            progress.setStringPainted(true);
            setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
            add(label1);
            add(Box.createVerticalGlue());
            add(label2);
            add(Box.createVerticalStrut(3));
            add(label3);
            add(Box.createVerticalStrut(10));
            add(progress);
            setBorder(BorderFactory.createMatteBorder(10,10,10,10,this.getBackground()));
        }
    }

    public void init(String print, long totalSize, int numberOfProgreesSteps)
    {
        this.print = print;
        this.totalSize = (double)totalSize;
        this.passed = 0;
        this.numberOfProgreesSteps = numberOfProgreesSteps;
        this.currentStep = 0;
        progress.setMaximum((int)totalSize);
        progress.setValue(0);
        label2.setText(print);
        label3.setText("");
        show();
        toFront();
    }

    public void update(long pass)
    {
        /*Debugger.debug(getClass(),3,"passed="+passed);
        Debugger.debug(getClass(),3,"b1="+passed/totalSize);
        Debugger.debug(getClass(),3,"b2="+currentStep/numberOfProgreesSteps);
        */
        passed = passed + (double)pass;
        if(passed/totalSize  >= currentStep/numberOfProgreesSteps)
        {
            int percentage = (int)(passed/totalSize*100);
            progress.setValue((int)passed);
            String printed = print+"\n  "+percentage+"% complete.";
            label3.setText(passed/1000+" / "+totalSize/1000 + " Kb ");
//          Debugger.debug(getClass(),3,printed);
            currentStep++;
            if(percentage >= 100)
                hide();
        }
    }

    //Centers the window on the screen
    private void position()
    {
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = getSize();
        int x = (screenSize.width - frameSize.width);
        int y = (screenSize.height - frameSize.height);
        setLocation(x -3, y - 33);
    }

    public static void main(String[] s)
    {
        try
        {

            //Look and feel
            try
            {
                UIManager.setLookAndFeel(
                    "SpamCash.Client.GUI.Definitions");
            }
            catch(Exception e)
```

```
            {
                System.out.println("Couldn't load graphics.");
            }


            //Test
            Progress p = new Progress();
            System.out.println("Contructed.");
            Thread.sleep(1000);
            System.out.println("Doing test job 1...");
            doTestJob(p);
            System.out.println("Job 1 Done.");
            System.out.println("Doing test job 2...");
            doTestJob(p);
            System.out.println("Job 2 Done.");
        }
        catch(Exception e)
        {
        }
        System.exit(0);
    }

    private static void doTestJob(Progress p)
    {
        p.init("Sending mail",10000,100);

        for(long i=0;i<10000;i += 100)
        {
            try
            {
                Thread.sleep(50);
            }
            catch(InterruptedException ex)
            {
            }
            p.update(100);
        }
        System.out.println("done.");
    }
}
```

10.1.4.k RegistrationFrame

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;

import SpamCash.Utilities.Debugger;
import SpamCash.Utilities.*;
import SpamCash.Client.Control;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.AddressException;

public class RegistrationFrame extends JFrame
{
    private Control control;
    private MainFrame frame;
    private final String informationtext =
        "You have to register in the " + Config.name+" system to be able to send mail
with it."+
        "\n\nBelow you must provide the necessary logon information for the currency
exchange server (CES),"+
        " choose a password for the local encryption and type in your emailadress.";
        private final String[] labels =
        {"CES Username", "CES password", "Local password", "Emailaddress:"};

    private final String[] buttonTypeNames =
```

```
        {"_normal", "_pressed", "_point"};
    private FormPanel logonPanel;
    private JTextArea informationTextArea;
    private ErrorDialog errorDialog;

    public RegistrationFrame(MainFrame frame, Control control) throws Exception
    {
        setTitle(Config.name+" Registration");
        this.frame = frame;
        this.control = control;

        addWindowListener(new WindowIsClosing());

        setCursor(MainFrame.cursor);
        setIconImage(MainFrame.cornerImage.getImage());
        getContentPane().add(new backgroundPanel());
        setSize(400, 315);
        setResizable(false);
        center();
        errorDialog = new ErrorDialog(this);
        logonPanel.textFields[0].setText(Config.defaultCESUsername);
        logonPanel.textFields[1].setText(Config.defaultCESPassword);
        logonPanel.textFields[2].setText(Config.defaultLocalPassword);
        //Just to make a random email appear
       // String[] emails =
{"santa@greenland","lerkenfeld@gmail.com","krabo@gmail.com","bo@boesen.dk","lille@lise.dk
"};
        logonPanel.textFields[3].setText("user1@127.0.0.1");
//emails[(int)(Math.random()*emails.length)]);
    }

    private class backgroundPanel extends JPanel
    {
        public backgroundPanel()
        {
            ActionListener listener = new actionListener();

            informationTextArea = Definitions.getTextArea();
            informationTextArea.setText(informationtext);

            ImageButton registerButton = new ImageButton("gradient", "Register...");
            registerButton.addActionListener(listener);

            JPanel buttonPanel = new JPanel();
            buttonPanel.add(registerButton);

            logonPanel = new FormPanel(labels
                                       , new boolean[]
                                       {false, true,true,false}
                                       , listener);

            setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
            add(informationTextArea);
            add(Box.createVerticalStrut(Definitions.getSpaceL()));
            add(logonPanel);
            add(Box.createVerticalStrut(Definitions.getSpaceL()));
            add(buttonPanel);

            setBorder(BorderFactory.createMatteBorder(Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                      Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                      getBackground()));

        }
    }

    private String checkFields()
    {
        if(logonPanel.textFields[0].getText().length() == 0)
            return "Please fill in the field \"" + labels[0] + "\"";
```

```
        if(logonPanel.textFields[1].getText().length() == 0)
            return "Please fill in the field \"" + labels[1] + "\"";

        if(logonPanel.textFields[2].getText().length() < 6)
            return "Please fill in the field \"" + labels[2] + "\" correctly. The
password must be at least 6 characters.";

        try
        {
            InternetAddress address = new
InternetAddress(logonPanel.textFields[3].getText());
            address.validate();
        }
        catch(AddressException e)
        {
            return "Not a valid e-mail address.";
        }
        return "";
    }

    protected void showError(String message)
    {
        errorDialog.showError(message);
    }

    private class actionListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            try
            {
                //Filled in correctly?
                String check = checkFields();
                if(check.length() != 0)
                {
                    showError(check);
                    Debugger.debug(getClass(),3,"Typing error");
                    return;
                }

                Debugger.debug(getClass(),3,"Calling Control.startRegistration(...)");
                control.startRegistration(logonPanel.textFields[0].getText(),
logonPanel.textFields[1].getText(),

logonPanel.textFields[2].getText(),logonPanel.textFields[3].getText());
            }
            catch(Exception ex)
            {
                StackTraceElement[] s = ex.getStackTrace();
                String m = "";
                for(int i=0;i<s.length;i++)
                    m += s[i].toString()+"\n";
                errorDialog.showError(ex.toString()+"\n"+m);
            }
        }
    }

    private void center()
    {
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = getSize();
        int x = (screenSize.width - frameSize.width) / 2;
        int y = (screenSize.height - frameSize.height) / 2;
        setLocation(x, y);
    }

    private class WindowIsClosing extends WindowAdapter
    {
        public void windowClosing(WindowEvent e)
        {
```

```
            control.shutdown();
        }
    }
}
```

10.1.4.l SpamPanel

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.net.InetAddress;
import javax.mail.internet.InternetAddress;
import java.util.StringTokenizer;

import SpamCash.Utilities.Debugger;
import SpamCash.Utilities.*;
import SpamCash.Client.Control;
import SpamCash.Client.Safe.*;
import SpamCash.Currency.Coin;
import SpamCash.Client.MailHandler.CoinHeader;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;

public class SpamPanel extends JPanel
{
        private MainFrame frame;
        private JTextArea informationTextArea,mailInformationTextArea;
        private final String informationtext = "Here you can retrive the money from the
SPAM you have received.";

        private JList list;

        protected SpamPanel(MainFrame frame)
        {
            this.frame = frame;

            mailInformationTextArea = Definitions.getTextArea();

            JPanel buttonPanel = new buttonPanel();

            setLayout(new BorderLayout());
            add(new SelectionPanel(),"North");
            add(mailInformationTextArea,"Center");
            add(buttonPanel,"South");

            setBorder(BorderFactory.createMatteBorder(Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                      Definitions.getSpaceL(),
Definitions.getSpaceL(),getBackground()));
        }

        private class SelectionPanel extends JPanel
        {
            public SelectionPanel()
            {
                informationTextArea = Definitions.getTextArea();
                informationTextArea.setText(informationtext);

                list = new JList();
                list.addListSelectionListener(new selectListener());

                setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
                add(Box.createVerticalStrut(Definitions.getSpaceL()));
                add(informationTextArea);
                add(Box.createVerticalStrut(Definitions.getSpaceL()));
                add(new JScrollPane(list));
```

```
                add(Box.createVerticalStrut(Definitions.getSpaceL()));
            }
        }

        private class buttonPanel extends JPanel
        {
            public buttonPanel()
            {

                ImageButton getCoinButton = new ImageButton("gradient", "Get Coin");
                getCoinButton.addActionListener(new actionListener());

                ImageButton updateButton = new ImageButton("gradient", "Update List");
                updateButton.addActionListener(new actionListener());

                setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
                add(Box.createHorizontalGlue());
                add(updateButton);
                add(Box.createHorizontalStrut(5));
                add(getCoinButton);
                add(Box.createHorizontalGlue());
            }
        }

        public void update(SafeHandler safe)
        {
            try
            {
                list.setListData(safe.getPendingMails());
            }
            catch(Exception ex)
            {
                Debugger.debug(getClass(),1,"Could not update the pending mails
list.",ex);
            }
        }

        protected void updateList()
        {
            //Debugger.debug(getClass(),3,"Update pressed.");
            frame.updateMailList();
            list.clearSelection();
            this.mailInformationTextArea.setText("");
        }

        private class actionListener implements ActionListener
        {
            public void actionPerformed(ActionEvent e)
            {
                if(e.getActionCommand().equals("Update List"))
                {
                    updateList();
                }
                else // "Get coin"
                {
                    //Debugger.debug(getClass(), 3, "Get Coin pressed.");
                    if(list.isSelectionEmpty())
                    {
                        frame.showError("Please select which mail to collect the coin
from, before pressing 'Collect Coin'");
                        return;
                    }

                    try
                    {
                        Object[] mails = list.getSelectedValues();
                        for(int i=0;i<mails.length;i++)
                        {
                            frame.collectCoin(((MailInformation) mails[i]));
                            updateList();
                        }
```

```
                        frame.showMessage(mails.length+" coin(s) collected succesfully.
The Safe status can be seen below. \n\n"+frame.getSafeStatus());
                    }
                    catch(Exception ex)
                    {
                        Debugger.debug(getClass(),1,"The coin could not be
collected.",ex);
                        frame.showError("The coin could not be collected.");
                    }
                }
            }
        }

        private class selectListener implements ListSelectionListener
        {
            public void valueChanged(ListSelectionEvent evt)
            {
                if(evt.getValueIsAdjusting())
                    return;
                if(((MailInformation)list.getSelectedValue()) != null)

mailInformationTextArea.setText(((MailInformation)list.getSelectedValue()).showContents()
);
            }
        }
}
```

10.1.4.m SplashScreen

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

import SpamCash.Utilities.Config;

public class SplashScreen extends JWindow
{
    private ImageIcon splashImage;
    private JLabel imageLabel;

    public SplashScreen()
    {
        splashImage = new ImageIcon(Toolkit.getDefaultToolkit().getImage(Config.imagedir
                                                                                +
System.getProperty("file.separator")
                                                                                +
"splash.gif"));

        setSize(new Dimension(450,118));
        imageLabel = new JLabel(splashImage);
        getContentPane().add(imageLabel);
        center();
        setVisible(true);
        try
        {
            Thread.sleep(100);
        }
        catch(InterruptedException ex)
        {
        }
    }

    //Centers the window on the screen
    private void center()
    {
```

```
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = getSize();
        int x = ( screenSize.width - frameSize.width)/2;
        int y = ( screenSize.height - frameSize.height)/2;
        setLocation(x,y);
    }

}
```

10.1.4.n WhitelistPanel

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

import SpamCash.Utilities.Debugger;
import SpamCash.Utilities.*;
import SpamCash.Client.Control;
import SpamCash.Client.Safe.SafeHandler;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import SpamCash.Client.Whitelist;
import javax.mail.internet.*;

public class WhitelistPanel extends JPanel
{
    private MainFrame frame;
    private Control control;

    private JTextArea informationTextArea, mailInformationTextArea;
    private final String informationtext = "E-mails from the addresses entered here will
never be blocked by the proxy.";

    private JTextField addressTextField;
    private JList list;

    protected WhitelistPanel(Control control,MainFrame frame)
    {
        this.frame = frame;
        this.control = control;

        mailInformationTextArea = Definitions.getTextArea();

        JPanel buttonPanel = new buttonPanel();

        setLayout(new BorderLayout());
        add(new SelectionPanel(), "North");
        add(mailInformationTextArea, "Center");
        add(buttonPanel, "South");
        list.setListData(control.getWhitelist().getData());
        setBorder(BorderFactory.createMatteBorder(Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                  Definitions.getSpaceL(),
Definitions.getSpaceL(), getBackground()));
    }

    private class SelectionPanel extends JPanel
    {
        public SelectionPanel()
        {
            informationTextArea = Definitions.getTextArea();
            informationTextArea.setText(informationtext);

            list = new JList();
            list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
            setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
            add(Box.createVerticalStrut(Definitions.getSpaceL()));
            add(informationTextArea);
            add(Box.createVerticalStrut(Definitions.getSpaceL()));
            add(new JScrollPane(list));
            add(Box.createVerticalStrut(Definitions.getSpaceL()));
        }
    }

    private class buttonPanel extends JPanel
    {
        public buttonPanel()
        {
            addressTextField = new JTextField();

            ImageButton addButton = new ImageButton("gradient", "Add");
            addButton.addActionListener(new actionListener());

            ImageButton removeButton = new ImageButton("gradient", "Remove");
            removeButton.addActionListener(new actionListener());

            setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
            add(Box.createHorizontalGlue());
            add(addressTextField);
            add(Box.createHorizontalStrut(5));
            add(addButton);
            add(Box.createHorizontalStrut(5));
            add(removeButton);
            add(Box.createHorizontalGlue());
        }
    }

    private class actionListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            if(e.getActionCommand().equals("Add"))
            {
                try
                {
                    control.addToWhitelist(addressTextField.getText());
                    addressTextField.setText("");
                }
                catch(Exception ex)
                {
                    Debugger.debug(getClass(), 1, ex.getMessage(), ex);
                    frame.showError(ex.getMessage());
                }
            }
            else
            {
                //Debugger.debug(getClass(), 3, "Get Coin pressed.");
                if(list.isSelectionEmpty())
                {
                    frame.showError("Please select which address to remove, before
pressing 'Remove'");
                    return;
                }

                try
                {

control.removeFromWhitelist((InternetAddress)list.getSelectedValue());
                }
                catch(Exception ex)
                {
                    Debugger.debug(getClass(), 1, "Could not remove address.", ex);
                    frame.showError("Could not remove address.");
                }
            }
```

```
            list.setListData(control.getWhitelist().getData());
        }
    }
}
```

10.1.4.o WithdrawPanel

```
package SpamCash.Client.GUI;


import javax.swing.*;
import java.awt.event.*;
import SpamCash.Utilities.Debugger;
import SpamCash.Utilities.Config;

public class WithdrawPanel extends JPanel
{
    private MainFrame frame;

    private final String informationtext = "Please enter the amount you wish to
withdraw.";
    private final String[] labels =
        {"Amount ", "Login name ", " Password "};

    private final String CESLabel = "Currency server information:";

    private FormPanel logonInformationPanel;
    private AmountPanel amountPanel;
    private JTextArea informationTextArea;
    private Class className;

    protected WithdrawPanel(MainFrame frame)
    {
        this.frame = frame;
        className = getClass();

        ActionListener listener = new textFieldListener();
        informationTextArea = Definitions.getTextArea();
        informationTextArea.setText(informationtext);
        amountPanel = new AmountPanel(labels[0],listener,false,0.05);
        logonInformationPanel = new FormPanel(new String[]
                                              {labels[1], labels[2]}
                                              , new boolean[]
                                              {false, true}
                                              , listener);
        logonInformationPanel.textFields[0].setText(Config.defaultCESUsername);
        logonInformationPanel.textFields[1].setText(Config.defaultCESPassword);
        ImageButton button = new ImageButton("gradient", "Withdraw...");
        button.addActionListener(listener);

        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        add(informationTextArea);
        add(Box.createVerticalStrut(Definitions.getSpaceL()));
        add(amountPanel);
        add(Box.createVerticalStrut(Definitions.getSpaceL()));
        add(new JLabel(CESLabel));
        add(Box.createVerticalStrut(Definitions.getSpaceS()));
        add(logonInformationPanel);
        add(Box.createVerticalStrut(Definitions.getSpaceL()));
        add(button);

        setBorder(BorderFactory.createMatteBorder(Definitions.getSpaceL(),
Definitions.getSpaceL(),
                                                  Definitions.getSpaceL(),
Definitions.getSpaceL(),getBackground()));
    }

    private String checkFields()
```

```
    {
        if(amountPanel.getText().length() == 0)
            return "Please fill in the field \"" + labels[0] + "\"";

        if(logonInformationPanel.textFields[0].getText().length() == 0)
            return "Please fill in the field \"" + labels[1] + "\"";

        if(logonInformationPanel.textFields[1].getText().length() == 0)
            return "Please fill in the field \"" + labels[2] + "\"";
        return "";
    }

    private class textFieldListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            try
            {
                //Filled in correctly?
                String check = checkFields();
                if(check.length() != 0)
                {
                    frame.showError(check);
                    return;
                }

                frame.withdraw(Double.parseDouble(amountPanel.getText()),
                               logonInformationPanel.textFields[0].getText(),
                               logonInformationPanel.textFields[1].getText());
            }
            catch(Exception ex)
            {
                Debugger.debug(className, 3, "Could not withdraw.", ex);
            }

        }
    }
}
```

10.1.5 MailHandler

10.1.5.a CoinHeader

```
package SpamCash.Client.MailHandler;


import org.logi.crypto.sign.Signature;
import java.security.cert.Certificate;
import SpamCash.Currency.Coin;
import java.io.IOException;
import java.security.cert.CertificateEncodingException;

public class CoinHeader implements java.io.Serializable
{
    //Signatures
    private byte[] s1, s2;
    private String hashFunc;

    public Coin coin;
    public Certificate cert;

    public CoinHeader(Signature s1,Signature s2,Coin coin,Certificate cert) throws
IOException
    {
        this.s1 = s1.getBytes();
        this.s2 = s2.getBytes();
        this.hashFunc = s1.getHashFunc();
        this.coin = coin;
        this.cert = cert;
```

```
    }

    public Signature getRecipientSignature()
    {
        return new Signature(hashFunc,s1);
    }

    public Coin getCoin() throws Exception
    {
        return coin;
    }

    public Signature getMailSignature()
    {
        return new Signature(hashFunc,s2);
    }

    public String toString()
    {
        String s ="CoinHeader contents:\n";

        try
        {
            s += printSizes();
            s += "  Coin:\n";
            s += coin.printSizes();
            s+= "Total:
"+(s1.length+s2.length+cert.getEncoded().length+coin.size())+"\n";
        }
        catch(Exception ex)
        {
        }
        return s;
    }

    private String printSizes() throws CertificateEncodingException
    {
        String s="";
        s += "  Signature 1: "+s1.length+" bytes\n";
        s += "  Signature 2: "+s2.length+" bytes\n";
        s += "  Certificate: "+cert.getEncoded().length+" bytes\n";
        return s;
    }
}
```

10.1.5.b IncomingMailProcessor

```
package SpamCash.Client.MailHandler;


import SpamCash.Client.Control;
import SpamCash.NetworkPackets.*;
import SpamCash.Utilities.*;
import SpamCash.Utilities.Debugger;

import java.io.*;
import SpamCash.Currency.Coin;
import javax.mail.MessagingException;
import javax.mail.internet.MimeMessage;
import org.logi.crypto.sign.Signature;
import java.security.cert.Certificate;
import java.security.interfaces.RSAPublicKey;
import SpamCash.Currency.TransactionlistElement;
import org.logi.crypto.sign.Fingerprint;
import javax.mail.Address;
import javax.mail.Message;
import javax.mail.internet.InternetAddress;

public class IncomingMailProcessor extends Thread
```

```
{
    private Control control;

    public IncomingMailProcessor(Control control)
    {
        this.control = control;
    }

    public void run()
    {
        MimePacket packet;
        boolean lastPacket = false;

        while(true)
        {
            try
            {
                if(lastPacket)
                {
                    Debugger.debug(getClass(), 2,
                                   "The last packet is done processing finished.");
                    control.incomingMailProcessingCompleted();
                }
                packet = (MimePacket) control.incomingMailQueue.getPacket();
                lastPacket = packet.lastPacket();
                processPOP(packet.getMessage());
            }
            catch(Throwable e)
            {
                Debugger.debug(getClass(), 1, "Processor interrupted.", e);
            }
        }
    }

    public MimeMessage processIMAP(MimeMessage message) throws Exception
    {
        if(keepMail(message))
            return message;
        return null;
    }

    //Should be private
    public void processPOP(MimeMessage message) throws Exception
    {
        //Here we process the mail...
        Debugger.debug(getClass(), 2, "Processing mail with ID = " +
message.getMessageID());

        if(keepMail((MimeMessage)message))
        {
            MimeTools.writeMimeFile(message,
FileHandler.findNewFile(Config.incomingMailDirectory));
            Debugger.debug(getClass(), 2, "Done processing mail. Saved mail with ID = " +
message.getMessageID());
        }
        else
            Debugger.debug(getClass(), 2, "Done processing mail. Did not save mail with
ID = " + message.getMessageID());
    }

    /**
     * @param sender String
     * @param coin Coin
     * @return true if the message should be kept and false if the message
     * should be deleted.
     */
    private boolean keepData(String sender,Object data, MimeMessage message) throws
Exception
    {
        Debugger.debug(getClass(), 1,"Testing message from: "+sender+"...");
```

```
        //Updating Blacklist
        control.updateBlacklist();

        //Read data
        if( ! (data instanceof CoinHeader))
            throw new Exception("Could not read CoinHeader.");
        CoinHeader header = (CoinHeader)data;

        Signature recipientSignature = header.getRecipientSignature();
        Signature mailSignature = header.getMailSignature();
        Coin coin = header.getCoin();
        Certificate cert = header.cert;

        //If the sender is on the blacklist we should delete the mail.
        if(control.isOnBlackList(cert))
             throw new Exception("Sender found on blacklist.");

        //MaxUsages exceeded...?
        if(coin.getMaxNumberOfUsages() <= (coin.getTransactionlist().length()-2)/2)
            throw new Exception("MaxNumberOfUsages exceeded.
"+coin.getMaxNumberOfUsages()+" <= "+(coin.getTransactionlist().length()-2)/2);

        //Testing if the coin was received before and therefore is in the pendingMails-
list
        if(control.getSafe().checkInboxCoins(coin.getSerialNumber()))
            throw new Exception("Coin already on the pendingMails list.");

        //Verify recipient signature
        Fingerprint recipientHash = Coin.getHash2(coin.getSerialNumber(),
((InternetAddress)message.getRecipients(Message.RecipientType.TO)[0]).getAddress());
        if( !
Cryptotools.verify((RSAPublicKey)cert.getPublicKey(),recipientSignature,recipientHash))
            throw new Exception("Could not verify recipient signature.");

        //Verify mail signature
        if( !
Cryptotools.verify((RSAPublicKey)cert.getPublicKey(),mailSignature,MailFunctions.createMa
ilHash(message)))
            throw new Exception("Could not verify mail signature.");

        //Verify CES signature
        if( ! coin.verifyCES(control.CESCertificate))
            throw new Exception("Could not verify CES signature.");

        //Verify transactionlist
        if( ! coin.verify(control.RSCertificate,control.CESCertificate))
            throw new Exception("Could not verify transactionlist.");

        //Coin seen before...(earnedCoinsList)?
        //Here it is assumed that the coin was verified earlier.
        if(!control.getSafe().checkEarnedCoins(coin, control.getCertificate()))
        {
            control.reportCoins(coin,
control.getSafe().getCoinWithSerial(coin.getSerialNumber()));
            throw new Exception("Coin has been used before. The coins where reported to
the Black list server.");
        }

        // Add the senders signature to the transactionlist (type 1)
        coin.getTransactionlist().addElement(new
TransactionlistElement(recipientSignature,null)); //null = recipienthash
        control.saveMail(coin, message);
        Debugger.debug(getClass(), 1,"Message passed all verifications. Processing of
message comleted.");
        return true;
    }

    /**
     *
     * @param message MimeMessage
     * @return true if the message should be kept and false if the message
```

```
     * should be deleted.
     */
    public boolean keepMail(MimeMessage message)
    {
        try
        {
            try
            {
                //MimeTools.writeMimeFile(message,"d:\\incomingstart.msg");

                Debugger.debug(getClass(), 1, "Testing if it is a notificationmail....");
                String[] note = message.getHeader(Config.headerNotificationName);
                if(note == null || note.length == 0)
                    throw new Exception("Could not read notificationinfo.");
                try
                {
                    Debugger.debug(getClass(), 1, "It is a notificationmail.");

                    NotificationHeader header =
(NotificationHeader)MimeTools.getCoinsHeader(message);

                    long serial = header.serialNumber;

                    //Verify signature...
                    Fingerprint hash = Coin.getHash2(header.serialNumber,
(message.getRecipients(Message.RecipientType.TO)[0]).toString());

if(Cryptotools.verify((RSAPublicKey)control.getCertificate().getPublicKey(),header.getSig
nature(),hash))
                    {
                        Debugger.debug(getClass(), 1, "Verifying signature...success");
                        if(control.deleteCoinWithSerial(header.serialNumber))
                        {
                            Debugger.debug(getClass(), 1, "C-coin deleted
successfully.");
                        }
                        else
                        {
                            Debugger.debug(getClass(), 1, "Could not delete C-coin.");
                        }
                    }
                    else
                        Debugger.debug(getClass(), 1, "Verifying signature...failed");
                    return false;
                }
                catch(Exception ex)
                {
                    Debugger.debug(getClass(), 3, "Error reading notificationdata.",ex);
                    return false;
                }
            }
            catch(Exception ex)
            {
                Debugger.debug(getClass(), 3, "Error handling notificationmail.",ex);
            }

            Debugger.debug(getClass(), 1, "It is NOT a notificationmail.");
            String[] from = message.getHeader("From");
            if(from.length > 1)
            {
                Debugger.debug(getClass(), 1,
                               "FATAL! more than one sender. Unable to filter mail
correctly. Deleting mail!");
                return false;
            }


            //If the sender is on the whitelist we should keep the mail.
            if(control.isOnWhitelist(from[0]))
            {
```

```
                Debugger.debug(getClass(), 1,"Sender '"+from[0]+"' is on the whitelist.
Keeping the mail.");
                Debugger.debug(getClass(), 2,"Trying to remove coin...");
                try
                {
                    MimeTools.removeAttachmentHeader(message);
                    Debugger.debug(getClass(), 2,"Coin removed successfully.");
                }
                catch(Exception e)
                {
                    Debugger.debug(getClass(), 2,"Could not remove the coin, maby no coin
was attached.");
                }
                Debugger.debug(getClass(), 1,"Processing of message comleted.");
                return true;
            }
            return keepData(from[0], MimeTools.getCoinsHeader(message),message);
            // return MimeTools.removeAttachmentHeader(message);
        }
        catch(MessagingException e)
        {
            Debugger.debug(getClass(), 1,"Could not read data from message.",e);
        }
        catch(Exception e)
        {
            Debugger.debug(getClass(), 1,"Exception testing message.",e);
        }
        Debugger.debug(getClass(), 1,"Deleting message.");
        return false;
    }
}
```

10.1.5.c MailFunctions

```
package SpamCash.Client.MailHandler;


import org.logi.crypto.sign.Signature;
import javax.mail.*;
import org.logi.crypto.sign.Fingerprint;
import SpamCash.Utilities.Debugger;
import SpamCash.Utilities.Cryptotools;
import javax.mail.internet.MimeMessage;
import SpamCash.Currency.Coin;

public class MailFunctions
{
    protected static Fingerprint createMailHash(MimeMessage extm) throws Exception
    {
        //Subject
        Fingerprint hash = Cryptotools.SHA1(extm.getSubject().toLowerCase());
        //To
        /*hash =
Cryptotools.merge(extm.getRecipients(Message.RecipientType.TO)[0].toString().toLowerCase(
), hash);
        //From
        hash = Cryptotools.merge(extm.getFrom().toString().toLowerCase(), hash);
*/
        //Here all parts of the message should be added....

        return hash;
    }

    //Handled by Coin.getHash2()
   /* protected static Fingerprint createRecipientHash(long serialNumber,MimeMessage
extm) throws Exception
    {
```

```
        return
createRecipientHash(serialNumber,(extm.getRecipients(Message.RecipientType.TO)[0]).toStri
ng());
    }

    private static Fingerprint createRecipientHash(long serialNumber,String to) throws
Exception
    {
        return Coin.getHash2(serialNumber,to);
    }*/
}
```

10.1.5.d NotificationHeader

```
package SpamCash.Client.MailHandler;


import java.security.cert.Certificate;
import org.logi.crypto.sign.Signature;
import javax.mail.Address;

public class NotificationHeader implements java.io.Serializable
{
    //Signatures
    private byte[] ns;
    private String hashFunc;

    public long serialNumber;
    public Address sender;
    public Certificate cert;

    public NotificationHeader(Signature ns,long serial,Address sender, Certificate cert)
    {
        this.ns = ns.getBytes();
        this.hashFunc = ns.getHashFunc();
        this.serialNumber = serial;
        this.sender = sender;
        this.cert = cert;
    }

    public Signature getSignature()
    {
        return new Signature(hashFunc,ns);
    }
}
```

10.1.5.e OutgoingMailProcessor

```
package SpamCash.Client.MailHandler;


import SpamCash.Utilities.*;
import java.io.ByteArrayOutputStream;
import SpamCash.Utilities.Debugger;
import SpamCash.Currency.Coin;
import java.util.GregorianCalendar;
import SpamCash.Client.Control;
import org.logi.crypto.sign.Signature;
import javax.mail.*;
import org.logi.crypto.sign.Fingerprint;
import java.security.interfaces.RSAPublicKey;
import javax.mail.internet.*;

public class OutgoingMailProcessor
{
    private Control control;
```

```
    public OutgoingMailProcessor(Control control)
    {
        this.control = control;
    }

    public MimeMessage processMail(MimeMessage extm) throws MessagingException, Exception
    {
        MimeTools.writeMimeFile(extm,"d:\\original.msg");
        Address[] to = extm.getRecipients(Message.RecipientType.TO);
        String receivers = "";
        for(int i = 0; i < to.length; i++)
            receivers += to[i].toString();
        Debugger.debug(getClass(), 1, "Processing mail to: " + receivers);

        //try
        //{

            Debugger.debug(getClass(), 2, "Attaching coin...");

            //Just using 1 coin at the moment
            Coin[] coins = control.getCoinsToSend(1,to);

            //Create signature 1
            Signature recipientSignature =
createRecipientSignature(coins[0].getSerialNumber(), extm);

            //Create signature 2
            Signature mailSignature = createMailSignature(extm);

            //Add all the parts to the email...
            CoinHeader header = new CoinHeader(recipientSignature, mailSignature,
coins[0], control.getCertificate());

            MimeMessage extm2 = MimeTools.addAttachmentHeader(extm, header);

            Debugger.debug(getClass(), 2, "Done.");

            //MimeTools.writeMimeFile(extm2,"d:\\outgoing.msg");

            return extm2;

        // Allow sending without attachment
        /*}
        catch(Exception e)
        {
            Debugger.debug(getClass(), 2, "Exception...cannot attach.", e);
            Debugger.debug(getClass(), 2, "Sending without attachment...");
            return extm;
        }*/
    }

    private Signature createRecipientSignature(long serialNumber, MimeMessage msg) throws
Exception
    {
        Debugger.debug(getClass(), 2, "Creating Recipient signature...");
        Fingerprint recipientHash = Coin.getHash2(serialNumber,
((InternetAddress)msg.getRecipients(Message.RecipientType.TO)[0]).getAddress());
        return Cryptotools.sign(control.getPrivateKey(), recipientHash);
    }

    private Signature createMailSignature(MimeMessage extm) throws MessagingException,
Exception
    {
        Debugger.debug(getClass(), 2, "Creating Mail signature...");

        return Cryptotools.sign(control.getPrivateKey(),
MailFunctions.createMailHash(extm));
    }
}
```

10.1.6 Proxy

10.1.6.a AbstractBlokingProxy

```
package SpamCash.Client.Proxy;


import SpamCash.Utilities.*;
import SpamCash.Utilities.Debugger;
import SpamCash.Client.Control;

import java.util.*;
import java.net.*;
import java.io.*;

public class AbstractBlockingProxy extends Thread
{
    /* Pointer to control class */
    protected Control control;

    /** Writer to sent data to the client/server */
    private PrintWriter clientOut, serverOut;

    /** Reader to read data from the client/server */
    private BufferedReader clientIn;
    private BufferedReader serverIn;

    /** Socket connection to the server */
    private Socket serverSocket, clientSocket;

    /** The server socket used to listen for incoming connections */
    protected ServerSocket listenSocket;

    /** The IP address of the client */
    private String clientIp, serverIp;
    protected int port;

    protected AbstractBlockingProxy(Control control, int port) throws Exception
    {
        this.control = control;
        this.port = port;
        startProxy();
    }

    public void startProxy() throws Exception
    {
        Debugger.debug(getClass(), 1, "Proxy is initializing...");

        try
        {
            this.listenSocket = new ServerSocket(port);
        }
        catch(Exception e)
        {
            throw new Exception(" Could not create Serversocket - port " + port
                                + " is in use.");
        }
        Debugger.debug(getClass(), 1,
                       "Proxy is ready on port " + port);
    }

    protected void initClientStreams(Socket clientSocket) throws Exception
    {
        this.clientSocket = clientSocket;
        this.clientSocket.setSoTimeout(200);
        //Prepare the input and output streams.
        clientOut = new PrintWriter(clientSocket.getOutputStream(), true);
        clientIn = new BufferedReader(new InputStreamReader(
            clientSocket.getInputStream()));
```

```
        InetAddress remoteAddress = clientSocket.getInetAddress();
        clientIp = remoteAddress.getHostAddress();
        Debugger.debug(getClass(), 2," Client "+
                       remoteAddress.getHostName() + "(" + clientIp
                       + ") socket connected on port " + port);

    }

    private void initServerStreams(Socket serverSocket) throws Exception
    {

        serverSocket.setSoTimeout(200);
        //Prepare the input and output streams.
        serverOut = new PrintWriter(serverSocket.getOutputStream(), true);
        serverIn = new BufferedReader(new InputStreamReader(
            serverSocket.getInputStream()));

        InetAddress remoteAddress = serverSocket.getInetAddress();
        serverIp = remoteAddress.getHostAddress();
        Debugger.debug(getClass(), 2,"Server "+
                       remoteAddress.getHostName() + " (" + serverIp
                       + ") socket connected on port " + port);
    }

    protected void initServerConnection()
    {
        //indsæt kode her som læser host og port fra fil ("config.server" eller lign.) og
benytter disse i næste linie.

        try
        {
            Debugger.debug(getClass(),1,"Connecting to server
"+Config.configuration.incomingMailserver+":"+port);
            serverSocket = new Socket(Config.configuration.incomingMailserver, port);
            initServerStreams(serverSocket);
        }
        catch(UnknownHostException ex)
        {
            Debugger.debug(getClass(), 1, "Unknown host Exception", ex);
        }
        catch(Exception ex)
        {

            Debugger.debug(getClass(), 1, "IOException", ex);
        }
    }

    protected String parseArgument(String inputString, int argumentIndex)
    {

        StringTokenizer st = new StringTokenizer(inputString);
        int index = -1;

        String argument = "";
        while(index < argumentIndex && st.hasMoreTokens())
        {
            argument = st.nextToken();
            index++;
        }
        return argument;
    }

    protected String lastArgument(String inputString)
    {
        StringTokenizer st = new StringTokenizer(inputString);
        String result = "";
        while(st.hasMoreTokens())
            result = st.nextToken();
        return result;
    }
```

```
protected String clientRead() throws IOException
{
    int total =10,times=1;
    while(!clientIn.ready())
    {
        Debugger.debug(getClass(), 3, "Readstream not ready "+times+" / "+total);
        if(times == total)
            break;
        times++;
        if(!this.clientSocket.isConnected())
        {
            throw new InterruptedIOException("Connection closed.");
        }
        try
        {
            Thread.sleep(100);
        }
        catch(InterruptedException ex)
        {
            Debugger.debug(getClass(), 3,
                           "Interrupted waiting for read().");
        }
    }
    String inputLine = clientIn.readLine();
    Debugger.debug(getClass(), 3, "read: " + inputLine);
    return inputLine;
}

protected void clientWrite(String message)
{
    Debugger.debug(getClass(), 3, "write: " + message);
    clientOut.print(message + "\r\n");
    clientOut.flush();
}

protected String serverRead() throws IOException
{
    while(!serverIn.ready())
    {
        try
        {
            Thread.sleep(100);
        }
        catch(InterruptedException ex)
        {
            Debugger.debug(getClass(), 3,
                           "Interrupted waiting for read().");
        }
    }
    String inputLine = serverIn.readLine();
    Debugger.debug(getClass(), 3, "read: " + inputLine);
    return inputLine;
}

protected void serverWrite(String message)
{
        Debugger.debug(getClass(), 3, "write: " + message);
    serverOut.print(message + "\r\n");
    serverOut.flush();
}

protected void closeServerConnection()
{
    try
    {

        //Closing connections to be ready for the next customer.
        serverOut.close();
        serverIn.close();
        serverSocket.close();
        Debugger.debug(getClass(), 2, "Server disconnected on port " + port);
```

```
        }
        catch(IOException ex)
        {
            Debugger.debug(getClass(), 2,
                           "Could not close server connection on IP "
                           + serverIp + " : " + port);
        }
    }

    protected void closeClientConnection()
    {
        try
        {
            clientIn.close();
            clientOut.close();
            clientSocket.close();
            Debugger.debug(getClass(), 2, "Client disconnected on port " + port);
        }
        catch(IOException ex)
        {
            Debugger.debug(getClass(), 2,
                           "Could not close client connection on IP "
                           + clientIp + " : " + port);
        }
    }
}
```

10.1.6.b AbstractNonBlokingProxy

```
package SpamCash.Client.Proxy;


import java.util.*;
import java.net.*;
import java.nio.channels.*;

import SpamCash.Client.Control;
import SpamCash.Utilities.*;
import SpamCash.Utilities.Debugger;

public abstract class AbstractNonBlockingProxy extends Thread
{
    private boolean running = true;

    private static Class className;
    private int idCounter = 1;

    private volatile SocketChannel keyChannel;
    private volatile SelectionKey key;

    private Vector connections;
    private int maxConnections;

    protected Control control;
    protected int port;

    public AbstractNonBlockingProxy(Control control, int port,int maxConnections) throws
Exception
    {
        this.control = control;
        this.port = port;
        this.maxConnections = maxConnections;

        this.className = getClass();
        connections = new Vector();
    }

    public void run()
    {
```

```
        try
        {
            SocketChannel ch;

            Selector selector = Selector.open();

            ServerSocketChannel listenChannel = ServerSocketChannel.open();
            listenChannel.configureBlocking(false);
            InetSocketAddress isa = new InetSocketAddress(port);
            listenChannel.socket().bind(isa);
            listenChannel.register(selector, SelectionKey.OP_ACCEPT);

            // Wait for something of interest to happen
            while(selector.select() > 0)
            {
                 //Debugger.debug(className, 3, "Some channels are ready...");
                // Get set of ready objects
                Set readyKeys = selector.selectedKeys();
                Iterator readyItor = readyKeys.iterator();

                // Walk through set
                while(readyItor.hasNext())
                {
                    try
                    {
                        key = (SelectionKey) readyItor.next();
                        readyItor.remove();

                        if(key.isAcceptable())
                        {

                            // Get the new channel ready
                            ServerSocketChannel keyServerChannel = (ServerSocketChannel)
key.channel();
                            SocketChannel newChannel = keyServerChannel.accept();

                            // Block connections which is not from localhost...
                            byte[] address =
newChannel.socket().getInetAddress().getAddress();
                            if( ! (address[0] == 127 && address[1] == 0 && address[2] ==
0 && address[3] == 1))
                            {
                                Debugger.debug(className, 1, "Rejecting connection since
it is not from 'localhost' !");
                                newChannel.close();
                                continue;
                            }

                            connections.add(new
NonBlockingProxyConnection(this,getClass().toString(), idCounter, selector, newChannel,
this.getServerAddress(), port));
                            idCounter++;

                            // Listen for the next connection
                            if(connections.size() < maxConnections)
                                listenChannel.register(selector, SelectionKey.OP_ACCEPT);
                        }
                        else if(key.isConnectable())
                        {
                            ch = (SocketChannel) key.channel();
                            ((NonBlockingProxyConnection)
connections.elementAt(findConnectionIndex(ch))).handleConnect(ch);
                        }
                        else if(key.isReadable())
                        {
                            ch = (SocketChannel) key.channel();
                            int index = findConnectionIndex(ch);
                            int status = ((NonBlockingProxyConnection)
connections.elementAt(index)).handleRead(ch);

                            // The connection is not needed anymore
```

```
                            if(status == 2)
                            {
                                connections.removeElementAt(index);
                            }
                        }
                        else if(key.isWritable())
                        {
                            ch = (SocketChannel) key.channel();
                            int index = findConnectionIndex(ch);
                            ((NonBlockingProxyConnection)
connections.elementAt(index)).handleWrite(ch);
                        }

                    }
                    catch(Exception ex)
                    {
                        Debugger.debug(className, 3, "Exception... ", ex);
                    }

                }
            }
        }
        catch(Exception e)
        {
            Debugger.debug(className, 3, "Initialization Exception... ", e);
        }
    }

    private int findConnectionIndex(SocketChannel ch)
    {
        for(int i = 0; i < connections.size(); i++)
        {
            if(((NonBlockingProxyConnection)
connections.elementAt(i)).containsChannel(ch))
                return i;
        }
        return -1;
    }

    protected String parseArgument(String inputString, int argumentIndex)
    {

        StringTokenizer st = new StringTokenizer(inputString);
        int index = -1;
        String argument = "";
        while(index < argumentIndex && st.hasMoreTokens())
        {
            argument = st.nextToken();
            index++;
        }
        return argument;
    }

    protected String parseArgument2(String inputString, int argumentIndex)
    {

        StringTokenizer st = new StringTokenizer(inputString,"\"");
        int index = -1;
        String argument = "";
        while(index < argumentIndex && st.hasMoreTokens())
        {
            argument = st.nextToken();
            index++;
        }
        return argument;
    }

    protected abstract String[] testServerCommand(String command);
    protected abstract Object testClientCommand(String command) throws Exception;
    protected abstract String getServerAddress();
}
```

10.1.6.c IMAPProxy

```
package SpamCash.Client.Proxy;


import javax.mail.*;
import javax.mail.search.*;
import javax.mail.internet.*;
import com.sun.mail.imap.IMAPFolder;
import java.util.*;

import SpamCash.Client.Control;
import SpamCash.Utilities.*;
import SpamCash.Utilities.*;
import SpamCash.Utilities.Debugger;

public class IMAPProxy extends AbstractNonBlockingProxy
{
    private static Class className;
    private static String username, password;

    private static Vector inboxCheckedUids;
    private static String currentFolder="",commandForTheClient="";
    private static boolean idle;

    public IMAPProxy(Control control, int port) throws Exception
    {
        super(control, port, Config.maxIMAPConnections);
        this.className = getClass();
        inboxCheckedUids = new Vector();
        idle = false;
        Debugger.debug(className, 1, "Proxy is ready on port " + port);
    }

    protected Object testClientCommand(String command)
    {
        //Debugger.debug(className, 2, "Testing command...: " + command);

        if(parseArgument(command, 1).equalsIgnoreCase("LOGIN"))
        {
            username = parseArgument(command, 2);
            password = parseArgument(command, 3);

            //remove the "'s
            username = username.substring(1, username.length() - 1);
            password = password.substring(1, password.length() - 1);

            Debugger.debug(className, 2,
                           "LOGIN detected. Saved username(" + username + ") and
password(" + password + ").");
        }

        if(parseArgument(command, 1).equalsIgnoreCase("IDLE"))
            idle = true;
        else
            idle = false;
//        Debugger.debug(className, 3, "testClientCommand: set IDLE = "+idle);


        /*if( -1 != command.indexOf("LOGOUT"))
        {
            username = "";
            password = "";
            Debugger.debug(className, 2, "LOGOUT detected. Deleted username and
password.");
        }*/

        //The client is changing the folder...
```

```
        if(parseArgument(command, 1).equalsIgnoreCase("SELECT"))
        {
            currentFolder = parseArgument2(command, 1);
            //Remove the " from the input...
            //currentFolder = currentFolder.substring(1,currentFolder.length()-1);
            Debugger.debug(getClass(),2,"Currentfolder set to: "+currentFolder);

            filterFolder(currentFolder);
        }


        //Debugger.debug(className, 2, "Done processing command: " + command);
        return command;
    }

    protected String[] testServerCommand(String command)
    {
        //The server says new mail has arrived
        if(parseArgument(command, 0).equalsIgnoreCase("*") &&
           (parseArgument(command, 2).equalsIgnoreCase("EXISTS") ||
parseArgument(command, 2).equalsIgnoreCase("RECENT")) &&
           idle)
        {
            if(!filterFolder(currentFolder))
            {
                //No new messages !
                Debugger.debug(getClass(), 1, "New messages has arrived, but they where
all removed by the filter.");
                Debugger.debug(getClass(), 1, "The client will never know =).");
                return new String[]{};
            }
            else
            {
                //Let the client receive the next commands...
                idle = false;
                Debugger.debug(className, 3, "testServerCommand: set IDLE = " + idle);
                return new String[]{command};
            }
        }
        //Debugger.debug(className, 2, "Done processing command: " + command);
        return new String[]{command};
    }

    private boolean filterFolder(String folder)
    {
        try
        {
            Debugger.debug(className, 2, "Checking "+folder+"...");

            Properties props = new Properties();
            props.put("mail.imap.host", Config.configuration.incomingMailserver);
            props.put("mail.imap.user", username);

            Session session = Session.getDefaultInstance(props);

            Store store = session.getStore("imap");

            store.connect(Config.configuration.incomingMailserver, username, password);
            IMAPFolder inbox = (IMAPFolder) store.getFolder(folder);

            inbox.open(Folder.READ_WRITE);
            int count = inbox.getMessageCount();
            //Debugger.debug(getClass(),3,"Number of Messages: " + count);

            MimeMessage newMessage;
            //Debugger.debug(className, 3,"Getting messages");
            javax.mail.Message[] messages = inbox.getMessages();
//(MimeMessage[])inbox.search(searchCriteria);

            int newmessages = 0;
            //Debugger.debug(className, 3,"Checking messages");
```

```
            for(int j = 0; j < messages.length; j++)
            {
                if(inboxCheckedUids.indexOf(((MimeMessage) messages[j]).getMessageID())
!= -1)
                {
                    //Message already checked...dont delete
                    messages[j].setFlag(Flags.Flag.DELETED, false);
                }
                else
                {
                    //Message should be checked
                    newMessage = control.incomingMailProcessor.processIMAP((MimeMessage)
messages[j]);

                    if(newMessage == null)
                    {
                        Debugger.debug(className, 2, "Deleting
message..."+j+"/"+messages.length);
                        messages[j].setFlag(Flags.Flag.DELETED, true);
                    }
                    else
                    {
                        Debugger.debug(className, 2, "Keeping
message..."+j+"/"+messages.length);
                        newmessages++;
                        //messages[j].setFlag(Flags.Flag.DELETED, true);
                        // Save UID to check nexttime
                        //inbox.appendMessages(new javax.mail.Message[]{newMessage});
                        inbox.expunge();
                        inboxCheckedUids.add(new String(newMessage.getMessageID()));
                        //Remove the coin
                        //MIMEReadWrite.removeAttachment(new
ExtMimeMessage((MimeMessage)messages[j]));               }
                    }
                }
            }
            //Messages are actually deleted here...
            inbox.close(true);
            Debugger.debug(className, 2, "Done checking folder " + folder);
            if(newmessages > 0)
                return true;
        }
        catch(Throwable e)
        {
            Debugger.debug(className, 2, "Exception checking folder " + folder + ": ",
e);
        }
        return false;
    }

    protected String getServerAddress()
    {
        return Config.configuration.incomingMailserver;
    }
}
```

10.1.6.d NonBlokingProxyConnection

```
package SpamCash.Client.Proxy;


import java.util.Vector;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
import java.io.*;

import SpamCash.Utilities.Debugger;
```

```
import SpamCash.Utilities.*;

import java.util.StringTokenizer;
import java.io.StringReader;
import SpamCash.Client.GUI.Progress;
import SpamCash.Client.GUI.MainFrame;
import SpamCash.Client.Control;

public class NonBlockingProxyConnection
{
    private Selector selector;
    private SocketChannel serverChannel, clientChannel;
    private Vector clientMessageQueue, serverMessageQueue;
    private String className;
    private boolean disconnecting = false;
    private int channelId;
    private AbstractNonBlockingProxy proxy;

    //Encoder and decoder
    private Charset charset;
    private CharsetDecoder decoder;
    private CharsetEncoder encoder;

    // Allocate buffers
    private ByteBuffer buffer;
    private CharBuffer charBuffer;

    private FileReader fileReader;
    private BufferedReader bufferedReader;
    private Progress progress = new Progress();

    private final int bufferSize = 512;

    protected NonBlockingProxyConnection(AbstractNonBlockingProxy proxy, String
className, int channelId, Selector selector, SocketChannel clientChannel, String
serverAddress, int serverPort) throws
        Exception
    {
        this.className = className + ":" + "NonBlokingProxyConnection";
        this.proxy = proxy;
        this.channelId = channelId;
        this.selector = selector;
        this.clientChannel = clientChannel;
        clientMessageQueue = new Vector();
        serverMessageQueue = new Vector();

        //Encoder and decoder
        charset = Charset.forName("US-ASCII");
        decoder = charset.newDecoder();
        encoder = charset.newEncoder();

        // Allocate buffers
        buffer = ByteBuffer.allocate(bufferSize);
        charBuffer = CharBuffer.allocate(bufferSize);

        clientChannel.configureBlocking(false);
        clientChannel.register(selector, SelectionKey.OP_READ | SelectionKey.OP_WRITE);

        // Make connection to server
        InetSocketAddress socketAddress = new InetSocketAddress(serverAddress,
serverPort);
        serverChannel = SocketChannel.open();
        serverChannel.configureBlocking(false);
        serverChannel.connect(socketAddress);
        serverChannel.register(selector, SelectionKey.OP_CONNECT | SelectionKey.OP_READ |
SelectionKey.OP_WRITE);
    }

    protected boolean containsChannel(SocketChannel channel)
    {
        return channel.equals(clientChannel) || channel.equals(serverChannel);
```

```
    }

    protected void handleConnect(SocketChannel channel)
    {
        if(!channel.equals(serverChannel))
        {
            Debugger.debug(className, 2, "Unknown channel received in handleConnect");
            return;
        }
        if(channel.isConnectionPending())
        {
            try
            {
                channel.finishConnect();
            }
            catch(IOException ex)
            {
                Debugger.debug(className, 2, "Exception trying to finnish
serverconnection");
            }
        }
    }

    /**
     * Handles a read event from either server or client

     *
     * @param channel SocketChannel
     * @throws Exception
     * @return boolean
     * 0: the channel is useable
     * 1: the channel is disconnecting (one party has disconnected)
     * 2: the channel is disconnected and can be destroyed
     */
    protected int handleRead(SocketChannel channel) throws Exception
    {
        buffer.clear();
        charBuffer.clear();

        int readStatus;
        try
        {
            readStatus = channel.read(buffer);
        }
        catch(Exception ex)
        {
            Debugger.debug(getClass(),3,"Exception reading channel...",ex);
            readStatus = -1;
            disconnecting = true;
        }
        // If read() returns -1, it indicates end-of-stream, which
        // means the client has disconnected.
        if(readStatus == -1)
        {
            //The other part is disconnecting to...
            if(disconnecting)
            {
                serverChannel.close();
                Debugger.debug(className, 3, "Closed serverchannel");
                clientChannel.close();
                Debugger.debug(className, 3, "Closed clientchannel");
                return 2;
            }

            disconnecting = true;
            Debugger.debug(className, 3, "Disconnecting...");
            return 1;
        }

        buffer.flip();
```

```
        // Decode buffer
        decoder.decode(buffer, charBuffer, false);
        charBuffer.flip();
        String s = channel.equals(serverChannel) ? "server" : "client";
        //Debugger.debug(className, 3, "Channel " + channelId + " read from " + s + ": "
+ charBuffer.toString());

        String message = charBuffer.toString();

        //Read from the client ?
        if(!channel.equals(serverChannel))
        {
            try
            {
                //Should the proxy react?
                Object result = proxy.testClientCommand(message);

                if(result instanceof File)
                {
                    fileReader = new FileReader((File) result);
                    bufferedReader = new BufferedReader(fileReader);

                    long filesize = ((File) result).length();
                    progress.init("Sending message (SMTP) ", filesize,
Config.numberOfUpdates);
                    Debugger.debug(className, 3, "Initialized outgoing file. Size " +
filesize);

                    //Be ready to write the message later
                    serverChannel.register(selector, SelectionKey.OP_READ |
SelectionKey.OP_WRITE);
                }
                else if(result instanceof String && !result.equals(""))
                {
                    //Add the message to the serverQueue
                    serverMessageQueue.add(result);
                    //Be ready to write the message later
                    serverChannel.register(selector, SelectionKey.OP_READ |
SelectionKey.OP_WRITE);
                }
            }
            catch(Exception e)
            {
                String error = "554 "+e.getMessage();
                Debugger.debug(className, 1, "Exception handling mail. Sending error to
the client and the server: " + error);
                write(error,clientChannel);
                write(error,serverChannel);
                serverChannel.close();
                clientChannel.close();
                Debugger.debug(className, 1, "Closed connections.");
                Control.showError(error);
                Debugger.debug(className, 1, "Error given to GUI.");
            }
        }
        else
        {
            //Should the proxy react?
            String [] messages = proxy.testServerCommand(message);

            if(messages.length != 0)
            {
                for(int i=0;i< messages.length;i++)
                {
                    //Add the message to the queue
                    clientMessageQueue.add(messages[i]);
                }
                //Be ready to write the message later
                clientChannel.register(selector, SelectionKey.OP_READ |
SelectionKey.OP_WRITE);
            }
```

```
        }
        return 0;
    }

    protected void handleWrite(SocketChannel channel) throws Exception
    {
        //Debugger.debug(getClass(),3,"Write-event. server?
"+channel.equals(serverChannel));
        String message = "";
        if(channel.equals(serverChannel))
        {
            if(serverMessageQueue.size() > 0)
            {
                message = (String) serverMessageQueue.remove(0);
                write(message,channel);
                //Printing
                String message2 = message;
                if(message.length()>30)
                    message2 =
message.substring(0,30)+"....."+message.substring(message.length()-30,message.length());
                Debugger.debug(className, 3, "Channel " + channelId + ": client -->
server: " + message2);
            }
            else if(fileReader != null)
            {
                buffer.clear();
                charBuffer.clear();
                char[] chars = new char[bufferSize];

                //Read as much as possible...
                int readChars = 0;
                while(bufferedReader.ready() && readChars < bufferSize)
                {
                    chars[readChars] = (char)bufferedReader.read();
                    readChars++;
                }

                if(readChars > 0)
                {
                    message = CharBuffer.wrap(chars, 0, readChars).toString();
                    write(message, channel);
                    progress.update(readChars);
                    //Debugger.debug(className, 3, "Channel " + channelId + ": client -->
server: " + message);
                }

                if(readChars != bufferSize)
                {
                    //Debugger.debug(getClass(),3,"Closing file stream.");
                    fileReader.close();
                    fileReader = null;
                    bufferedReader.close();
                    FileHandler.clearDirectory(Config.outgoingMailDirectory);
                }
            }

            //Dont listen to WRITE events if there is nothing to write...
            if(serverMessageQueue.size() == 0 && fileReader == null)
                serverChannel.register(selector, SelectionKey.OP_READ);
        }
        else if(clientMessageQueue.size() > 0)
        {
            message = (String) clientMessageQueue.remove(0);

            write(message,channel);
            Debugger.debug(className, 3, "Channel " + channelId + ": server --> client: "
+ message);

            //Dont listen to WRITE events if there is nothing to write...
            if(clientMessageQueue.size() == 0)
                clientChannel.register(selector, SelectionKey.OP_READ);
```

```
        }

    }

    private String getName(SocketChannel channel)
    {
        return channel.equals(serverChannel) ? "server" : "client";
    }

    protected String parseArgument(String inputString, int argumentIndex)
    {
        StringTokenizer st = new StringTokenizer(inputString);
        int index = -1;
        String argument = "";
        while(index < argumentIndex && st.hasMoreTokens())
        {
            argument = st.nextToken();
            index++;
        }
        return argument;
    }

    protected void write(String message,SocketChannel channel) throws Exception
    {
        //Debugger.debug(className,3,"Writing message:");
        int written = 0,end;
        int buffers = 0;
        while(written < message.length())
        {
            end = written + bufferSize;
            end = Math.min(end, message.length());
            write(CharBuffer.wrap(message.substring(written, end)),channel);
            written += end;
            buffers++;
            //Debugger.debug(className,3,"Writing used "+buffers+" buffers.");
        }
    }

    protected void write(CharBuffer b,SocketChannel channel) throws Exception
    {
        channel.write(encoder.encode(b));
    }
}
```

10.1.6.e POP3Proxy

```
package SpamCash.Client.Proxy;


//Java imports
import java.net.*;
import java.io.*;

import SpamCash.Client.Control;
import SpamCash.Client.Proxy.*;
import SpamCash.NetworkPackets.*;
import SpamCash.Utilities.*;
import SpamCash.Utilities.Debugger;
import SpamCash.Client.MailHandler.IncomingMailProcessor;
import SpamCash.Client.GUI.Progress;

public class POP3Proxy extends AbstractBlockingProxy
{
    private static final boolean deleteMessages = true;

    //General Message
    private static final String MESSAGE_OK = "+OK";
    private static final String MESSAGE_INVALID_COMMAND =
        "-ERR Unknown command: ";
```

```
    private static final String MESSAGE_TOO_FEW_ARGUMENTS =
        "-ERR Too few arguments for this command.";

    //Other Messages
    private static final String MESSAGE_NOT_A_NUMBER =
        "-ERR Command requires a valid number as an argument.";
    private static final String MESSAGE_NO_SUCH_MESSAGE =
        "-ERR No such message.";
    private static final String MESSAGE_ALREADY_DELETED =
        "-ERR Message already deleted.";
    private static final String UNABLE_TO_CONNECT =
        "-ERR Unable to connect to server ";

    //Command Constants
    private static final String COMMAND_QUIT = "QUIT";
    private static final String COMMAND_STAT = "STAT";
    private static final String COMMAND_LIST = "LIST";
    private static final String COMMAND_RETR = "RETR";
    private static final String COMMAND_DELE = "DELE";
    private static final String COMMAND_NOOP = "NOOP";
    private static final String COMMAND_RSET = "REST";
    private static final String COMMAND_TOP = "TOP";
    private static final String COMMAND_UIDL = "UIDL";

    /** Indicates if this thread should continue to run or shut down */
    private boolean running = true;

    private Progress progress = new Progress();

//    private static Class className;

    private Message[] messages;

    //To hold the servers quit-response in the proxy
    private String quitMessage = "";

    public POP3Proxy(Control control, int port) throws Exception
    {
        super(control, port);
        //className = getClass();
    }

    public void run()
    {

        while(running)
        {
            try
            {
                super.initClientStreams(listenSocket.accept());

                waitForAuthentication();
                quitMessage = waitForMailToGetReady();
                messages = getMessages();
                super.closeServerConnection();
                handleCommands();
            }
            catch(InterruptedIOException iioe)
            {
                Debugger.debug(getClass(), 2, "Interrupted...", iioe);
                super.closeClientConnection();
            }
            catch(Throwable e)
            {
                //Client said QUIT...
                super.closeClientConnection();
            }
        }
        Debugger.debug(getClass(), 2, "Pop3Processor shut down gracefully");
    }
```

```
private void waitForAuthentication() throws Exception
{
    String result = "", clientInput = "";
    boolean authenticated = false;

    initServerConnection();

    String welcomeMessage = serverRead();

    if(welcomeMessage == null)
    {
        clientWrite(UNABLE_TO_CONNECT);
        return;
    }
    clientWrite(welcomeMessage);

    while(!authenticated)
    {
        clientInput = clientRead();
        serverWrite(clientInput);
        result = serverRead();

        if((clientInput.substring(0, 4).equalsIgnoreCase("PASS"))
           && result.substring(0, 3).equalsIgnoreCase("+OK"))
            authenticated = true;
        clientWrite(result);
    }

    Debugger.debug(getClass(), 2,
                   "Server authenticated user succesfully. Getting the mail...");
}

private String waitForMailToGetReady() throws Exception
{
    //Make sure the client does not receive the mails
    // Before the processing is comleted.
    control.setPOP3ServerShouldWait(true);
    String quitMessage = retrieveMail();

    //Wait for the IncomingMailProcessor to complete the processing
    // of the received mails
    Debugger.debug(getClass(), 2,
                   "Done fetching mail waiting for the processor.");
    control.setAreAllMailsQueued(true);
    control.waitForOrderToGo(control.ID_POP3);

    return quitMessage;
}

private synchronized String retrieveMail() throws Exception
{
    String result = "";
    String quitMessage = "";
    String mimeMessage = "";
    int index;
    int numberOfMails;
    int sizeInBytes;

    serverWrite("STAT");
    result = serverRead();

    if(!parseArgument(result, 0).equals("+OK"))
        throw new Exception("Received error response from mailserver.");

    numberOfMails = Integer.parseInt(parseArgument(result, 1));

    serverWrite("LIST");
    result = serverRead();

    int[] messageSizes = new int[numberOfMails];
    index = 0;
```

```
        while(index <= numberOfMails)
        {
            result = serverRead();
            if(result.equals("."))
            {
                if(index < numberOfMails)
                    return quitMessage;
                break;
            }
            messageSizes[index] = Integer.parseInt(parseArgument(result, 1));
            index++;
        }

        //Getting the mails..
        String filename = "";
        File tempFile;
        FileWriter oFile;
        BufferedWriter bout;
        PrintWriter out;
        index = 1;
        while(index <= numberOfMails)
        {
            serverWrite("RETR " + index);
            result = serverRead();
            if(!parseArgument(result, 0).equals("+OK"))
            {
                Debugger.debug(getClass(), 1,
                               "Could not retrieve mail " + index);
                return quitMessage;
            }

            filename = FileHandler.findNewFile(Config.tempIncomingMailDirectory);
            tempFile = new File(filename);

            oFile = new FileWriter(tempFile);
            bout = new BufferedWriter(oFile);
            out = new PrintWriter(bout);

            progress.init("Receiving message " + index + " of "+numberOfMails +" (POP3)",
messageSizes[index - 1],
                          Config.numberOfUpdates);

            sizeInBytes = 0;
            result = serverRead();
            int print = 1;
            // Make sure we only stop receiving when
            // we read a '.' AND the full message has been read.
            while(!(result.equals(".")
                    && sizeInBytes >= messageSizes[index - 1]))
            {
                out.write(result);
                out.write("\n");
                sizeInBytes += result.length() + 2;
                progress.update(result.length() + 2);
                try
                {
                    result = serverRead();
                }
                catch(Exception e)
                {
                    Debugger.debug(getClass(), 3, "Read timed out...");
                }
                //Debugger.debug(getClass(), 3, "Size: "+(sizeInBytes - 2) +" >=
"+messageSizes[index - 1]);
            }
            out.close();
            bout.close();
            oFile.close();

            control.incomingMailQueue.addPacket(new MimePacket(tempFile,
```

```
                                                        index
                                                        == numberOfMails));

            index++;
        }

        //Deleting the mails.
        index = 1;
        while(index <= numberOfMails)
        {
            serverWrite("DELE " + index);
            result = serverRead();
            if(!parseArgument(result, 0).equals("+OK"))
                Debugger.debug(getClass(), 1, "Could not delete mail " + index);
            index++;
        }

        if(numberOfMails == 0)
        {
            Debugger.debug(getClass(), 2,
                           "No mails to process telling control to continue.");
            control.incomingMailProcessingCompleted();
        }

        serverWrite("QUIT");
        quitMessage = serverRead();

        return quitMessage;
    }

    /**
     * Handles all the commands related the the retrieval of mail.
     */
    private void handleCommands() throws Exception
    {

        //Reusable Variables.
        String inputString;
        String command;
        String argument;

        //This just runs until a SystemException is thrown, which
        //signals us to disconnect.
        while(true)
        {
            inputString = clientRead();

            command = parseCommand(inputString);
            argument = parseArgument(inputString);

            //Debugger.debug(getClass(),3,"Command: " + command + " " + argument);
            //Identify the command and call the appropriate helper method.
            if(command.equals(COMMAND_QUIT))
            {
                handleQuit();
            }
            else if(command.equals(COMMAND_STAT))
            {
                handleStat();
            }
            else if(command.equals(COMMAND_LIST))
            {
                handleList(argument);
            }
            else if(command.equals(COMMAND_RETR))
            {
                handleRetr(argument);
            }
            else if(command.equals(COMMAND_DELE))
            {
                handleDele(argument);
```

```
            }
            else if(command.equals(COMMAND_NOOP))
            {
                clientWrite("+OK");
            }
            else if(command.equals(COMMAND_RSET))
            {
                handleRset();
            }
            else if(command.equals(COMMAND_TOP))
            {
                handleTop(argument);
            }
            else if(command.equals(COMMAND_UIDL))
            {
                handleUidl(argument);
            }
            else
            {
                clientWrite(MESSAGE_INVALID_COMMAND + command);
            }
        }
    }

    /**
     * Checks to verify that the command is not a quit command.  If it is,
     * the current state is finalized (all messaged marked as deleted are
     * actually deleted) and closes the connection.
     */
    private void handleQuit()
    {
        Debugger.debug(getClass(), 2, "User has QUIT the session.");

        //Delete the messages marked as deleted from disk

        if(deleteMessages)
        {
            int numMessage = messages.length;
            Debugger.debug(getClass(), 3, "Testing for delete..");

            for(int index = 0; index < numMessage; index++)
            {
                Debugger.debug(getClass(), 3, "delete " + index + "?");
                if(messages[index].isDeleted())
                {
                    Debugger.debug(getClass(), 3, "Deleting..." +
messages[index].getMessageLocation());
                    messages[index].getMessageLocation().delete();
                }
            }
        }

        clientWrite(quitMessage);
        Debugger.debug(getClass(), 1, "Mail retrieved successfully.");
        throw new RuntimeException();
    }

    /**
      * Handles the 'stat' command, which returns the total number of message
      * and the total size of those message.
      */
     private void handleStat()
     {
         clientWrite("+OK " + getNumberOfMessages() + " " + getSizeOfAllMessages());
     }

    /**
     * Handles the 'list' command, which returns the total number of messages and
     * size along with a list of the individual message sizes.
     */
    private void handleList(String argument)
```

```
{

    if(argument.equals(""))
    {
        long numMessages = getNumberOfMessages();
        long sizeMessage = getSizeOfAllMessages();

        clientWrite("+OK " + numMessages + " messages (" + sizeMessage
                    + " octets)");

        for(int index = 0; index < numMessages; index++)
        {
            clientWrite((index + 1) + " "
                        + getMessage(index + 1).getMessageLocation().length());
        }
        clientWrite(".");
    }
    else
    {
        int messageNumber = 0;

        try
        {
            messageNumber = Integer.parseInt(argument);
        }
        catch(NumberFormatException nfe)
        {
            clientWrite(MESSAGE_NOT_A_NUMBER);
            return;
        }

        long numMessages = getNumberOfMessages();

        if(messageNumber > numMessages
           || getMessage(messageNumber).isDeleted())
        {
            clientWrite(MESSAGE_NO_SUCH_MESSAGE);
            return;
        }

        clientWrite("+OK " + messageNumber + " "
                    + getMessage(messageNumber).getMessageLocation().length());
    }
}

/**
 * Sends the specified email message to the client.
 */
private void handleRetr(String argument)
{

    int messageNumber = 0;

    try
    {
        messageNumber = Integer.parseInt(argument);
    }
    catch(NumberFormatException nfe)
    {
        clientWrite(MESSAGE_NOT_A_NUMBER);
        return;
    }

    long numMessages = getNumberOfMessages();

    Debugger.debug(getClass(), 2,
                   "Message " + messageNumber + " of " + numMessages
                   + " was deleted?: "
                   + getMessage(messageNumber).isDeleted());

    if(messageNumber > numMessages
```

```
       || getMessage(messageNumber).isDeleted())
    {
        clientWrite(MESSAGE_NO_SUCH_MESSAGE);
        return;
    }

    clientWrite(MESSAGE_OK);

    BufferedReader fileIn = null;
    try
    {
        //Open an reader to read the file.
        fileIn = new BufferedReader(new FileReader(getMessage(
            messageNumber).getMessageLocation()));

        //Write the file to the client.
        String currentLine = fileIn.readLine();
        while(currentLine != null)
        {
            clientWrite(currentLine);
            currentLine = fileIn.readLine();
        }
        clientWrite(".");
    }
    catch(FileNotFoundException fnfe)
    {
        Debugger.debug(getClass(), 2,
                       "Requested message could not be found on disk.",
                       fnfe);
    }
    catch(IOException ioe)
    {
        Debugger.debug(getClass(), 2, "Error retrieving message.", ioe);
        clientWrite("-ERR Error retrieving message");
    }
    finally
    {
        //Make sure the input stream gets closed.
        try
        {
            if(fileIn != null)
            {
                fileIn.close();
            }
        }
        catch(IOException ioe)
        {
            //Nothing to do...
        }
    }
}

/**
 * Marks the specified message for deletion.  The message will only
 * be deleted if the user later enters the QUIT command, as per the
 * spec.
 */
private void handleDele(String argument)
{

    int messageNumber = 0;

    try
    {
        messageNumber = Integer.parseInt(argument);
    }
    catch(NumberFormatException nfe)
    {
        clientWrite(MESSAGE_NOT_A_NUMBER);
        return;
    }
```

```
    long numMessages = getNumberOfMessages();

    if(messageNumber > numMessages)
    {
        clientWrite(MESSAGE_NO_SUCH_MESSAGE);
    }
    else if(getMessage(messageNumber).isDeleted())
    {
        clientWrite(MESSAGE_ALREADY_DELETED);
    }
    else
    {
        getMessage(messageNumber).setDeleted(true);
        clientWrite(MESSAGE_OK);
    }
}

/**
 * Unmarks all deleted messages.
 */
private void handleRset()
{

    Message[] messages = getMessages();
    int numMessage = messages.length;

    for(int index = 0; index < numMessage; index++)
    {
        messages[index].setDeleted(false);
    }

    clientWrite(MESSAGE_OK);
}

/**
 * Returns the header and first x lines for the
 * specified message.
 */
private void handleTop(String argument)
{

    int messageNumber = 0;
    int numLines = 0;

    int spaceIndex = argument.indexOf(" ");
    if(spaceIndex == -1)
    {
        clientWrite(MESSAGE_TOO_FEW_ARGUMENTS);
        return;
    }
    String arg1 = argument.substring(0, spaceIndex).trim();
    String arg2 = argument.substring(spaceIndex + 1).trim();

    try
    {
        messageNumber = Integer.parseInt(arg1);
        numLines = Integer.parseInt(arg2);
    }
    catch(NumberFormatException nfe)
    {
        clientWrite(MESSAGE_NOT_A_NUMBER);
        return;
    }

    long numMessages = getNumberOfMessages();

    Debugger.debug(getClass(), 2,
                   "Message " + messageNumber + " of " + numMessages
                   + " was deleted?: "
                   + getMessage(messageNumber).isDeleted());
```

```
    if(messageNumber > numMessages
       || getMessage(messageNumber).isDeleted())
    {
        clientWrite(MESSAGE_NO_SUCH_MESSAGE);
        return;
    }

    clientWrite(MESSAGE_OK);

    BufferedReader fileIn = null;
    try
    {
        //Open an reader to read the file.
        fileIn = new BufferedReader(new FileReader(getMessage(
            messageNumber).getMessageLocation()));

        //Write the Message Header.
        String currentLine = fileIn.readLine();
        while(currentLine != null && !currentLine.equals(""))
        {
            clientWrite(currentLine);
            currentLine = fileIn.readLine();
        }

        //Write an empty line to seperate header from body.
        clientWrite(currentLine);
        currentLine = fileIn.readLine();

        //Write the requested number of lines from the body of the
        //message, or until the entire message has been written.
        int index = 0;
        while(index < numLines && currentLine != null)
        {
            clientWrite(currentLine);
            currentLine = fileIn.readLine();
            index++;
        }

        clientWrite(".");
    }
    catch(FileNotFoundException fnfe)
    {
        Debugger.debug(getClass(), 2,
                       "Requested message for user could not be found on disk.",
                       fnfe);
    }
    catch(IOException ioe)
    {
        Debugger.debug(getClass(), 2, "Error retrieving message.", ioe);
        clientWrite("-ERR Error retrieving message");
    }
    finally
    {
        //Make sure the input stream gets closed.
        try
        {
            if(fileIn != null)
            {
                fileIn.close();
            }
        }
        catch(IOException ioe)
        {
        }
    }
}

/**
 * Returns the unique id of the specified message, or all the unique
 * ids of the non-deleted messages.
```

```
 */
private void handleUidl(String argument)
{

    //Return all messages unique ids
    if(argument == null || argument.length() == 0)
    {

        long numMessages = getNumberOfMessages();
        Message message;

        clientWrite(MESSAGE_OK);

        //Write out each non-deleted message id.
        for(int index = 0; index < numMessages; index++)
        {
            message = getMessage(index + 1);
            if(!message.isDeleted())
            {
                clientWrite((index + 1) + " " + message.getUniqueId());
            }
        }

        clientWrite(".");
    }
    //Ouput a single messages unique id.
    else
    {

        int messageNumber = 0;

        try
        {
            messageNumber = Integer.parseInt(argument);
        }
        catch(NumberFormatException nfe)
        {
            clientWrite(MESSAGE_NOT_A_NUMBER);
            return;
        }

        long numMessages = getNumberOfMessages();

        if(messageNumber > numMessages
           || getMessage(messageNumber).isDeleted())
        {
            clientWrite(MESSAGE_NO_SUCH_MESSAGE);
            return;
        }

        clientWrite(MESSAGE_OK + " " + messageNumber + " "
                    + getMessage(messageNumber).getUniqueId());
    }
}

private String parseCommand(String inputString)
{

    int index = inputString.indexOf(" ");

    if(index == -1)
    {
        String command = inputString.toUpperCase();
        return command;
    }
    else
    {
        String command = inputString.substring(0, index).toUpperCase();
        return command;
    }
}
```

```
private String parseArgument(String inputString)
{

    int index = inputString.indexOf(" ");

    if(index == -1)
    {
        return "";
    }
    else
    {
        return inputString.substring(index + 1).trim();
    }
}

public Message[] getMessages()
{

    File directory = new File(Config.incomingMailDirectory);

    String[] fileNames = directory.list();

    int numMessage = 0;
    if(fileNames != null)
        numMessage = fileNames.length;

    messages = new Message[numMessage];
    Message currentMessage;

    for(int index = 0; index < numMessage; index++)
    {
        currentMessage = new Message();
        currentMessage.setMessageLocation(new File(directory,
                                                   fileNames[index]));
        messages[index] = currentMessage;
    }

    return messages;
}

/**
 * Gets the specified message.  Message numbers are 1 based.
 * This method counts on the calling method to verify that the
 * messageNumber actually exists.
 */
private Message getMessage(int messageNumber)
{
    return messages[messageNumber - 1];
}

/**
 * Gets the total number of messages currently stored for this user.
 */
public long getNumberOfMessages()
{
    return messages.length;
}

/**
 * Gets the total size of the messages currently stored for this user.
 */
public long getSizeOfAllMessages()
{
    Message[] message = getMessages();

    long totalSize = 0;

    for(int index = 0; index < message.length; index++)
    {
        totalSize += message[index].getMessageLocation().length();
    }
```

```
        return totalSize;
    }

    /**
     * Gets the user's directory as a file.  This method also verifies
     * that that directory exists.
     */
    public File getUserDirectory()
    {
        return new File(Config.incomingMailDirectory);
    }

    /**
     * This method removes any cached message information this user may have stored
     */
    private void reset()
    {

        messages = null;
    }

    private class Message
    {
        private File messageLocation;
        private boolean deleted = false;

        public File getMessageLocation()
        {
            return messageLocation;
        }

        public void setMessageLocation(File messageLocation)
        {
            this.messageLocation = messageLocation;
        }


        public long getMessageSize()
        {
            return messageLocation.length();
        }

        public boolean isDeleted()
        {
            return deleted;
        }

        public void setDeleted(boolean deleted)
        {
            this.deleted = deleted;
        }

        public String getUniqueId()
        {
            String location = messageLocation.getAbsolutePath();

            int end = location.lastIndexOf(".msg");

            return location.substring(0, end);
        }
    }
}
```

10.1.6.f SMTPProxy

```
package SpamCash.Client.Proxy;


import java.io.*;
```

```
import SpamCash.Utilities.Debugger;
import SpamCash.Client.Control;
import SpamCash.Client.MailHandler.OutgoingMailProcessor;
import SpamCash.Utilities.Cryptotools;
import SpamCash.Utilities.*;
import SpamCash.Client.MailHandler.CoinHeader;
import javax.mail.internet.MimeMessage;

public class SMTPProxy extends AbstractNonBlockingProxy
{
    private Class className;
    private String lastCommand = "", partialMessage = "";
    private int buffers;
    private File tempFile;
    private boolean readingMail,processingDone;//, savingCoin;
    private OutgoingMailProcessor outgoingMailProcessor;

    private FileWriter oFile;
    private BufferedWriter bout;
    private PrintWriter out;

    public SMTPProxy(Control control, int port) throws Exception
    {
        super(control, port, 1);
        className = getClass();
        buffers = 0;
        readingMail = false;
       // savingCoin = false;
        outgoingMailProcessor = new OutgoingMailProcessor(control);
        Debugger.debug(className, 1, "Proxy is ready on port " + port);
    }

    protected String[] testServerCommand(String command)
    {
        if(processingDone && parseArgument(command, 0).equalsIgnoreCase("250"))
        {
            Debugger.debug(getClass(), 1, "Mail Sent succesfully.");
            processingDone = false;
        }

        return new String[]
            {command};
    }

    protected Object testClientCommand(String command) throws Exception
    {
        if( parseArgument(lastCommand, 0).equalsIgnoreCase("DATA"))
        {
            try
            {

                if(!readingMail)
                {
                    tempFile = new
File(FileHandler.findNewFile(Config.outgoingMailDirectory));

                    oFile = new FileWriter(tempFile);
                    bout = new BufferedWriter(oFile);
                    out = new PrintWriter(bout);
                    readingMail = true;
                }

                if( -1 == command.indexOf("\r\n.\r\n"))
                {
                    buffers++;
                    Debugger.debug(className, 3, "Message used " + buffers + "
buffers.");
                    out.write(command);
                    //Return "" so that nothing is sent to the server.
                    return "";
```

```
                }

                buffers = 0;
                //command = partialMessage + command;
                out.write(command);
                out.close();
                bout.close();
                oFile.close();

                Debugger.debug(className, 1, "Sending mail...");
                lastCommand = "";

                readingMail = false;

                //Process mail
                processingDone = true;
                String filename = Config.outgoingMailDirectory + File.separator +
Config.processedMailFilename;
                MimeMessage processedMail =
outgoingMailProcessor.processMail(MimeTools.readMimeFile(tempFile.

getAbsolutePath()));
                MimeTools.writeMimeFile(processedMail, filename);
                return new File(filename);
            }
            catch(Exception ex)
            {
                Debugger.debug(getClass(), 1, "Could not send mail. Initialization of
file failed.", ex);
                //return command;
                throw ex;
            }
        }
        lastCommand = command;
        return command;
    }

    protected String getServerAddress()
    {
        return Config.configuration.outgoingMailserver;
    }

    public static void main(String[] s)
    {
        //Debugger.debug(getClass(),3,"Bytes: ");
    }
}
```

10.1.7 Safe

10.1.7.a Safe

```
package SpamCash.Client.Safe;


import java.util.Vector;

import SpamCash.Currency.Coin;
import SpamCash.Utilities.Debugger;
import javax.mail.Address;
import java.util.Collections;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.*;

class Safe implements java.io.Serializable
{
    private Vector ACoins;
    private Vector CCoins;
```

```
    private CoinReceivers coinReceivers;

    protected Safe()
    {
        ACoins = new Vector();
        CCoins = new Vector();
        coinReceivers = new CoinReceivers();

        //Test data...
        /*System.out.println("In the beginning:\n" + coinReceivers);
        try
        {
            coinReceivers.sentCoin(1, new InternetAddress("lerkenfeld@gmail.com"));
            coinReceivers.sentCoin(2, new InternetAddress("lerkenfeld@gmail.com"));
            coinReceivers.sentCoin(3, new InternetAddress("lerkenfeld@gmail.com"));
            coinReceivers.sentCoin(4, new InternetAddress("krabo@gmail.com"));
            coinReceivers.sentCoin(5, new InternetAddress("krabo@gmail.com"));
            coinReceivers.sentCoin(6, new InternetAddress("5503tv@abenghaven.dk"));
            coinReceivers.sentCoin(7, new InternetAddress("krabo@ishoejby.dk"));
            coinReceivers.sentCoin(8, new InternetAddress("cdj@imm.dtu.dk"));
            coinReceivers.sentCoin(9, new InternetAddress("cdj@imm.dtu.dk"));
            coinReceivers.sentCoin(10, new InternetAddress("cdj@imm.dtu.dk"));
            coinReceivers.sentCoin(11, new InternetAddress("cdj@imm.dtu.dk"));
            coinReceivers.sentCoin(12, new InternetAddress("cdj@imm.dtu.dk"));
            coinReceivers.sentCoin(13, new InternetAddress("cdj@imm.dtu.dk"));
            coinReceivers.sentCoin(14, new InternetAddress("cdj@imm.dtu.dk"));
            coinReceivers.sentCoin(15, new InternetAddress("cdj@imm.dtu.dk"));
            coinReceivers.sentCoin(16, new InternetAddress("cdj@imm.dtu.dk"));
            coinReceivers.sentCoin(17, new InternetAddress("cdj@imm.dtu.dk"));
            coinReceivers.sentCoin(18, new InternetAddress("santa@greenland.net"));
            coinReceivers.sentCoin(19, new InternetAddress("santa@greenland.net"));
            coinReceivers.sentCoin(20, new InternetAddress("santa@greenland.net"));
            coinReceivers.sentCoin(21, new InternetAddress("santa@greenland.net"));
            coinReceivers.sentCoin(22, new InternetAddress("santa@greenland.net"));
            coinReceivers.sentCoin(23, new InternetAddress("bob@krypto.com"));
            coinReceivers.sentCoin(24, new InternetAddress("alice@krypto.com"));

            coinReceivers.deletedCoin(10);
            coinReceivers.deletedCoin(11);
            coinReceivers.deletedCoin(12);
            coinReceivers.deletedCoin(13);
            coinReceivers.deletedCoin(14);
            coinReceivers.deletedCoin(15);
            coinReceivers.deletedCoin(16);
            coinReceivers.deletedCoin(17);
            coinReceivers.deletedCoin(18);
            coinReceivers.deletedCoin(19);
            coinReceivers.deletedCoin(20);
            coinReceivers.deletedCoin(21);
            coinReceivers.deletedCoin(22);
            coinReceivers.deletedCoin(4);

        }
        catch(AddressException ex)
        {
        }
*/
    }


    protected void addCoin(Coin a)
    {
        if(a.getType() == Coin.A_TYPE)
            ACoins.add(a);
        else if(a.getType() == Coin.B_TYPE)
            Debugger.debug(getClass(),1,"FATAL. Trying to add a B-coin to the safe.");
        else if(a.getType() == Coin.C_TYPE)
        {
            Debugger.debug(getClass(),1,"FATAL. Trying to add a C-coin to the safe.");
        }
        else
```

```
            Debugger.debug(getClass(),1,"FATAL. Unknown cointype in addCoin(Coin).");
    }

    protected int getNumberOfACoins() throws Exception
    {
        //Maby some of the CCoins are expired, and can be turned into A-coins
        updateCCoins();
        return ACoins.size();
    }

    protected Coin[] getACoins(int n) throws Exception
    {
        updateCCoins();
        if(n > ACoins.size())
            throw new Exception("Could not get "+n+" A-coins.");

        Coin[] coins = new Coin[n];

        for(int i=coins.length-1;i >= 0;i--)
        {
            coins[i] = (Coin)ACoins.remove(i);
        }
        return coins;
    }

    protected Coin[] getBCoins(int n, Address[] receivers) throws Exception
    {
        updateCCoins();
        if(n > ACoins.size())
            throw new Exception("Could not get "+n+" B-coins.");

        Coin[] coins = new Coin[n];
        Coin[] tempCCoins = new Coin[n];

        for(int i=coins.length-1;i >= 0;i--)
        {
            coins[i] = (Coin)ACoins.remove(i);
            coins[i].setType(Coin.B_TYPE);
            tempCCoins[i] = new Coin(coins[i]);
            tempCCoins[i].setType(Coin.C_TYPE);
            CCoins.add(tempCCoins[i]);
            coinReceivers.sentCoin(tempCCoins[i].getSerialNumber(),receivers[i]);
        }
        return coins;
    }

    private void updateCCoins() throws Exception
    {
        Debugger.debug(getClass(),1,"Updating coins...");
        int count = 0;
        for(int i=CCoins.size()-1;i >= 0;i--)
        {
            if(((Coin)CCoins.elementAt(i)).transferTimeexpired())
            {
                Coin c = ((Coin) CCoins.remove(i));
                coinReceivers.expiredCoin(c.getSerialNumber());
                c.setType(Coin.A_TYPE);
                ACoins.add(c);
                count++;
            }
        }
        Debugger.debug(getClass(),1,"Update complete: "+count+" C-coins where turned into
A-coins.");
    }

    private double getValue(Vector v)
    {
        double value = 0.0;
        for(int i=0;i<v.size();i++)
        {
            value += ((Coin)v.elementAt(i)).getValue();
```

```
    }
    return value;
}

protected boolean deleteCCoin(long serialNumber)
{
    for(int i=0;i < CCoins.size();i++)
    {
        if(((Coin)CCoins.elementAt(i)).getType() == Coin.C_TYPE &&
           ((Coin)CCoins.elementAt(i)).getSerialNumber() == serialNumber)
        {
            CCoins.remove(i);
            coinReceivers.deletedCoin(serialNumber);
            return true;
        }
    }
    return false;
}

public String toString()
{
   try
    {
        updateCCoins();
    }
    catch(Exception ex)
    {
    }
    String s = "\n";

    s += "Usable coins: "+ACoins.size()+" (worth "+getValue(ACoins)+" bucks!)";

    if(Debugger.debugging)
    {
        s += "\nSerialNumbers:";

            for(int i = 0; i < ACoins.size(); i++)
        {
            s += ((Coin) ACoins.elementAt(i)).getSerialNumber();
            if(i < ACoins.size() - 1)
                s += ", ";
        }
    }

    s += "\n\nWaiting coins: "+CCoins.size()+" (worth "+getValue(CCoins)+" bucks!)";

    if(Debugger.debugging)
    {
        s += "\nSerialNumbers: ";

        for(int i = 0; i < CCoins.size(); i++)
        {
            s += ((Coin) CCoins.elementAt(i)).getSerialNumber();
            if(i < CCoins.size() - 1)
                s += ", ";
        }
    }

    s += coinReceivers;

    return s;
}

private static class CoinReceivers implements java.io.Serializable
{
    private Vector receivers;

    public CoinReceivers()
    {
        receivers = new Vector();
    }
```

```
        public void sentCoin(long serial,Address receiver)
        {
            CoinReceiver r = new CoinReceiver(serial,receiver);
            if( ! receivers.contains(r))
                receivers.add(r);
            else
                ((CoinReceiver)receivers.elementAt(receivers.indexOf(r))).add(serial);
        }

        public void deletedCoin(long serial)
        {
            find(serial).addKept();
        }

        public void expiredCoin(long serial)
        {
            find(serial).substract();
        }

        private CoinReceiver find(long serial)
        {
            for(int i=0;i<receivers.size();i++)
                if(((CoinReceiver)receivers.elementAt(i)).serials.contains(new
Long(serial)))
                     return (CoinReceiver)receivers.elementAt(i);
            return null;
        }

        public String toString()
        {
            String s = "\n\nThe following shows the number of your e-coins the receivers
has received and kept.";
            s += "\n\nKept , Received , Address\n\n";
            Collections.sort(receivers);
            for(int i=0;i<receivers.size();i++)
                s += receivers.elementAt(i)+"\n";
            return s;
        }

        private class CoinReceiver implements Comparable,java.io.Serializable
        {
            public Vector serials;
            public Address receiver;
            public int count,keptCount;

            public CoinReceiver(long serial,Address receiver)
            {
                this.serials = new Vector();
                this.serials.add(new Long(serial));
                this.receiver = receiver;
                this.count = 1;
                this.keptCount = 0;
            }

            public void add(long serial)
            {
                serials.add(new Long(serial));
                count++;
            }

            public void addKept()
            {
                keptCount++;
            }

            public void substract()
            {
                count--;
            }
```

```
            public int compareTo(Object o)
            {
                return (this.keptCount > ((CoinReceiver)o).keptCount) ? -1 : 1;
            }

            public boolean equals(Object o)
            {
                return this.receiver.equals(((CoinReceiver)o).receiver);
            }

            public String toString()
            {
                return keptCount+" , "+count+" , "+ this.receiver.toString();
            }
        }
    }

    public static void main(String[] s)
    {
        try
        {
            CoinReceivers coinReceivers = new CoinReceivers();
            System.out.println("In the beginning:\n"+coinReceivers);
            coinReceivers.sentCoin(1, new InternetAddress("bo@boesen.dk"));
            coinReceivers.sentCoin(2, new InternetAddress("anders@something.dk"));
            coinReceivers.sentCoin(3, new InternetAddress("niels@something.dk"));
            System.out.println("Sent to three guys:\n"+coinReceivers);
            coinReceivers.sentCoin(4, new InternetAddress("niels@something.dk"));
            System.out.println("Sent to niels again:\n"+coinReceivers);
            coinReceivers.sentCoin(5, new InternetAddress("niels@something.dk"));
            System.out.println("Sent to niels again:\n"+coinReceivers);
            coinReceivers.expiredCoin(1);
            coinReceivers.expiredCoin(4);
            System.out.println("Bo and one of niels coins expired:\n"+coinReceivers);
            coinReceivers.sentCoin(6, new InternetAddress("niels@something.dk"));
            System.out.println("Sent to niels again:\n"+coinReceivers);
            coinReceivers.deletedCoin(5);
            System.out.println("Niels kept 1 coin:\n"+coinReceivers);
            coinReceivers.deletedCoin(6);
            System.out.println("Niels kept 1 coin:\n"+coinReceivers);
        }
        catch(AddressException ex)
        {
            System.out.println("Error: "+ex);
            ex.printStackTrace();
        }
    }
}
```

10.1.7.b SafeHandler

```
package SpamCash.Client.Safe;


import java.io.*;
import java.util.Vector;

import SpamCash.Currency.Coin;
import SpamCash.Utilities.Debugger;
import SpamCash.Utilities.*;
import java.security.Key;
import javax.crypto.SecretKey;
import java.util.Enumeration;
import java.security.cert.Certificate;
import java.security.interfaces.RSAPrivateKey;
import SpamCash.Client.Control;
import javax.mail.Address;
import java.util.Hashtable;
```

```
public class SafeHandler implements java.io.Serializable
{
    private boolean isOpen = false;
    private String keyFilePath, password;

    private Safe safe;
    private Vector pendingMails;
    protected static CoinHistory coinHistory;
    private RSAPrivateKey privatekey;

    public void addPrivateKey(RSAPrivateKey key) throws Exception
    {
        this.privatekey = key;
        saveToFile();
    }

    public RSAPrivateKey getPrivateKey() throws Exception
    {
        if(!isOpen())
            throw new Exception("The safe is not open.");
        if(privatekey == null)
            throw new Exception("The privatekey is not present!");
        return this.privatekey;
    }

    public Vector getPendingMails() throws Exception
    {
        if(!isOpen())
            throw new Exception("The safe is not open.");

        for(int i = pendingMails.size() - 1; i >= 0; i--)
        {
            if(((MailInformation) pendingMails.elementAt(i)).c.transferTimeexpired())
                pendingMails.remove(i);
        }

        return pendingMails;
    }

    public void addPendingMail(MailInformation m) throws Exception
    {
        pendingMails.add(m);
        saveToFile();
    }

    protected CoinHistory getHistory()
    {
        return coinHistory;
    }

    public boolean addCoins(Coin[] coins) throws Exception
    {
        //Add the coins to the correct arrays and save the information on disk.
        for(int i = 0; i < coins.length; i++)
        {
            safe.addCoin(coins[i]);
            coinHistory.earnedCoin(coins[i]);
        }
        saveToFile();
        return true;
    }

    public boolean checkEarnedCoins(Coin c, Certificate cert) throws Exception
    {
        return coinHistory.checkEarnedCoins(c, cert);
    }

    public boolean checkInboxCoins(long sn)
    {
        for(int i = 0; i < pendingMails.size(); i++)
        {
```

```
            Debugger.debug(getClass(), 3,
                           "Pendingmailserial " + ((MailInformation)
pendingMails.elementAt(i)).c.getSerialNumber()
                           + " =? " + sn);
            if(((MailInformation) pendingMails.elementAt(i)).c.getSerialNumber() == sn)
                return true;
        }
        return false;
    }

    public void transferCoin(MailInformation m, Certificate cert, RSAPrivateKey key)
throws Exception
    {
        if(pendingMails.indexOf(m) == -1)
            throw new Exception("Could not find " + m);

        // We can now sign the contents of the list (type 2)
        m.c.getTransactionlist().signList(cert, key);

        m.c.setType(Coin.A_TYPE);
        coinHistory.earnedCoin(m.c);
        safe.addCoin(m.c);
        pendingMails.remove(m);
        saveToFile();
    }

    public boolean deleteCoinWithSerial(long serial)
    {
        return safe.deleteCCoin(serial);
    }

    public Coin getCoinWithSerial(long serial)
    {
        return coinHistory.getCoinWithSerial(serial);
    }

    /**
     * Retrieves 'numberOfCoins' coins from the safe.
     * @param numberOfCoins int
     * @return Coin[]
     */
    public Coin[] getCoins(int numberOfCoins) throws Exception
    {
        if(!isOpen())
            throw new Exception("The safe is not open. Cannot retrieve " + numberOfCoins
+ " coins.");

        if(numberOfCoins > safe.getNumberOfACoins())
            throw new Exception("There is only " + safe.getNumberOfACoins()
                                + " coins available in the safe. Could not retrieve " +
numberOfCoins + " coins.");

        Coin[] c = safe.getACoins(numberOfCoins);
        saveToFile();
        return c;
    }

    public Coin[] getCoinsToSend(int numberOfCoins, Address[] receivers) throws Exception
    {
        if(!isOpen())
            throw new Exception("The safe is not open. Cannot retrieve " + numberOfCoins
+ " coins.");

        if(numberOfCoins > safe.getNumberOfACoins())
            throw new Exception("There is only " + safe.getNumberOfACoins()
                                + " coins available in the safe. Could not retrieve " +
numberOfCoins + " coins.");

        Coin[] c = safe.getBCoins(numberOfCoins, receivers);
        saveToFile();
        return c;
```

```
    }

    private Coin[] mergeCoinArrays(Coin[] a, Coin[] b)
    {
        Coin[] c = new Coin[a.length + b.length];
        int i = 0;
        for(; i < a.length; i++)
        {
            c[i] = a[i];
        }
        for(; i < a.length + b.length; i++)
        {
            c[i] = b[i];
        }
        return c;
    }

    private byte[] mergeByteArrays(byte[] a, byte[] b)
    {
        byte[] c = new byte[a.length + b.length];
        int i = 0;
        for(; i < a.length; i++)
        {
            c[i] = a[i];
        }
        for(; i < a.length + b.length; i++)
        {
            c[i] = b[i];
        }
        return c;
    }

    public boolean isOpen()
    {
        return isOpen;
    }

    public void saveToFile() throws Exception
    {
        Debugger.debug(getClass(), 1,
                       "Saving Safe to " + Config.safeFilePath + " using password: " +
password
                       + " and key-file-path: "
                       + keyFilePath);
        if(!isOpen())
            throw new Exception("The safe is not open. Cannot save to disk");

        String error = "";
        try
        {
            SecretKey key = (SecretKey) FileHandler.readObject(new File(keyFilePath));

            Debugger.debug(getClass(), 3, "Encrypting the privatekey.");
            byte[] passEncryptedKey = Cryptotools.encryptWithPassword(privatekey,
password);
            byte[] encryptedKey = Cryptotools.encryptAES(passEncryptedKey, key);

            Debugger.debug(getClass(), 3, "Encrypting the safe.");
            byte[] passEncryptedSafe = Cryptotools.encryptWithPassword(safe, password);
            byte[] encryptedSafe = Cryptotools.encryptAES(passEncryptedSafe, key);

            Debugger.debug(getClass(), 3, "Encrypting the pending mails.");
            byte[] passEncryptedMails = Cryptotools.encryptWithPassword(pendingMails,
password);
            byte[] encryptedMails = Cryptotools.encryptAES(passEncryptedMails, key);

            Debugger.debug(getClass(), 3, "Encrypting the coin history.");
            byte[] passEncryptedCoinHistory =
Cryptotools.encryptWithPassword(coinHistory, password);
            byte[] encryptedCoinHistory =
Cryptotools.encryptAES(passEncryptedCoinHistory, key);
```

```
            Debugger.debug(getClass(), 3, "Saving to file...");

            if(Config.safeInOneFile)
            {
                ByteArrayOutputStream outStream = new ByteArrayOutputStream();
                ObjectOutputStream out = new ObjectOutputStream(outStream);
                out.writeObject(encryptedKey);
                out.writeObject(encryptedSafe);
                out.writeObject(encryptedMails);
                out.writeObject(encryptedCoinHistory);

                byte[] result = outStream.toByteArray();
                outStream.close();
                out.close();

                FileHandler.writeBytes(result, new File(Config.safeFilePath));
            }
            else
            {
                FileHandler.writeBytes(encryptedKey, new
File(Config.privateKeyFilePath));
                FileHandler.writeBytes(encryptedSafe, new File(Config.safeFilePath));
                FileHandler.writeBytes(encryptedMails, new
File(Config.pendingMailsFilePath));
                FileHandler.writeBytes(encryptedCoinHistory, new
File(Config.coinHistoryFilePath));
            }
            Debugger.debug(getClass(), 1, "Safe stored succesfully in " +
Config.safeFilePath);
            return;
        }
        catch(FileNotFoundException fnf)
        {

            error = "The safe could not be saved. The keyfile " + keyFilePath
                + " was not found. Please specify the correct path.";
            Debugger.debug(getClass(), 1, error);
            throw new FileNotFoundException(error);
        }
        catch(Exception ioe)
        {
            error = "The safe could not be saved. The keyfile " + keyFilePath
                + " is corrupt. Please specify the correct path.";
            Debugger.debug(getClass(), 1, error, ioe);
            throw new Exception(error);
        }
    }

    public void openSafe(String keyFilePath, String password) throws Exception
    {
        //Make sure we dont overwrite any information
        if(safe != null || pendingMails != null || coinHistory != null)
            return;

        String error = "";
        try
        {
            Debugger.debug(getClass(), 1,
                           "Loading " + Config.safeFilePath + " using password: " +
password + " and key-file-path: "
                           + keyFilePath);
            byte[] safeBytes, mailBytes, coinBytes, keyBytes;
            try
            {
                if(Config.safeInOneFile)
                {
                    byte[] bytes = FileHandler.readBytes(new File(Config.safeFilePath));
                    ByteArrayInputStream stream = new ByteArrayInputStream(bytes);
                    ObjectInputStream in = new ObjectInputStream(stream);
```

```
                    keyBytes = (byte[]) in.readObject();
                    safeBytes = (byte[]) in.readObject();
                    mailBytes = (byte[]) in.readObject();
                    coinBytes = (byte[]) in.readObject();

                    stream.close();
                    in.close();
                }
                else
                {
                    keyBytes = FileHandler.readBytes(new
File(Config.privateKeyFilePath));
                    safeBytes = FileHandler.readBytes(new File(Config.safeFilePath));
                    mailBytes = FileHandler.readBytes(new
File(Config.pendingMailsFilePath));
                    coinBytes = FileHandler.readBytes(new
File(Config.coinHistoryFilePath));
                }
            }
            catch(FileNotFoundException ex)
            {
                Debugger.debug(getClass(), 1, "The safefiles could not be found. Creating
new safe file.");

                //Create a new Safe
                Control.generateKeyFile(Config.defaultSafeKeyFilePath);
                safe = new Safe();
                pendingMails = new Vector();
                coinHistory = new CoinHistory();
                this.keyFilePath = keyFilePath;
                this.password = password;
                isOpen = true;
                saveToFile();
                return;
            }

            SecretKey key = (SecretKey) FileHandler.readObject(new File(keyFilePath));

            //Restoring Private key
            byte[] decrypted = Cryptotools.decryptAES(keyBytes, key);
            this.privatekey = (RSAPrivateKey) Cryptotools.decryptWithPassword(decrypted,
password);

            //Restoring Safe
            decrypted = Cryptotools.decryptAES(safeBytes, key);
            safe = (Safe) Cryptotools.decryptWithPassword(decrypted, password);

            //Restoring pending mails
            decrypted = Cryptotools.decryptAES(mailBytes, key);
            pendingMails = (Vector) Cryptotools.decryptWithPassword(decrypted, password);

            //Restoring coinHistory
            decrypted = Cryptotools.decryptAES(coinBytes, key);
            coinHistory = (CoinHistory) Cryptotools.decryptWithPassword(decrypted,
password);

            //Save the safeinformation
            this.keyFilePath = keyFilePath;
            this.password = password;
            isOpen = true;

            Debugger.debug(getClass(), 1, "Safe loaded succesfully.");
            return;
        }
        catch(FileNotFoundException fnf)
        {
            error = "The safe could not be opened. The keyfile " + keyFilePath
                + " was not found. Please specify the correct path.";
            Debugger.debug(getClass(), 1, error, fnf);
        }
        catch(Exception ioe)
```

```
        {
            error = "The safe could not be opened. An Input/Output error has occured.";
            Debugger.debug(getClass(), 1, error, ioe);
        }
        throw new Exception(error);
    }

    public String toString()
    {
        String s = safe.toString();

        /*      try
              {
                  s += "COIN size:\n";
                  s += safe.getACoins(1)[0].printSizes();
              }
              catch(Exception ex)
              {
                  Debugger.debug(getClass(),3,"Error:",ex);
              }
         */
        if(Debugger.debugging)
            s += "\n\nUsed serialNumbers:\n" + coinHistory + " \n\nPending mails:\n" +
toString(pendingMails);
        return s;
    }

    private String toString(Vector v)
    {
        String s = "";
        for(int i = 0; i < v.size(); i++)
        {
            s += pendingMails.elementAt(i).toString() + "\n";
        }
        return s;

    }

    public class CoinHistory implements java.io.Serializable
    {
        private Hashtable usedCoins;

        public CoinHistory()
        {
            usedCoins = new Hashtable();
        }

        public void earnedCoin(Coin c)
        {
            if(!usedCoins.containsKey(new Long(c.getSerialNumber())))
                usedCoins.put(new Long(c.getSerialNumber()), new EarnedCoin(c));
            else
            {
                EarnedCoin ec = (EarnedCoin) usedCoins.get(new
Long(c.getSerialNumber()));
                ec.coin = c;
                ec.earnedTimes++;
            }
        }

        /**
         * Returns true if the coin is not in the list
         * Returns true if the coin is in the list, and
         * the number of appearances in the transactionlist = timesEarned
         * @param c Coin
         * @return boolean
         */
        public boolean checkEarnedCoins(Coin c, Certificate cert) throws Exception
        {
            if(!usedCoins.containsKey(new Long(c.getSerialNumber())))
                return true;
```

```
            EarnedCoin ec = (EarnedCoin) usedCoins.get(new Long(c.getSerialNumber()));
            Debugger.debug(getClass(), 1, "EarnedCoin found: " + ec);
            Debugger.debug(getClass(), 1, "NumnberofAppearances: " +
c.getTransactionlist().numberOfAppearances(cert));
            if(ec.earnedTimes - 1 < c.getTransactionlist().numberOfAppearances(cert))
                return true;
            Debugger.debug(getClass(), 1, "returning false.");
            return false;
        }

        public Coin getCoinWithSerial(long serial)
        {
            return((EarnedCoin) usedCoins.get(new Long(serial))).coin;
        }

        public String toString()
        {
            String s = "(serialnumber,earned-count) : ";
            Enumeration e = usedCoins.elements();
            int i = 0;
            while(e.hasMoreElements())
            {
                s += (EarnedCoin) e.nextElement();
                if(i < usedCoins.size() - 1)
                    s += ", ";
                i++;
            }
            return s;
        }

        private class EarnedCoin implements java.io.Serializable
        {
            public int earnedTimes;
            public Coin coin;

            public EarnedCoin(Coin coin)
            {
                this.coin = coin;
                earnedTimes = 1;
            }

            public boolean equals(EarnedCoin ec)
            {
                return earnedTimes == ec.earnedTimes && this.coin.equals(ec.coin);
            }

            public String toString()
            {
                return "(" + coin.getSerialNumber() + "," + earnedTimes + ")";
            }
        }
    }


}
```

10.2 Currency

10.2.1 BlindedCoin

```
package SpamCash.Currency;


import org.logi.crypto.sign.BlindFingerprint;
import java.security.interfaces.*;
import org.logi.crypto.keys.RSABlindingFactor;
import org.logi.crypto.sign.Signature;
```

```
import SpamCash.Utilities.Cryptotools;
import org.logi.crypto.sign.BlindSignature;

public class BlindedCoin implements java.io.Serializable
{
    private byte[] bf1,bf2;
    private String hashFunc,blindFunc;

    private String signatureHashFunc;
    private byte[] blindSignature1,blindSignature2;

    public BlindedCoin(BlindFingerprint bf1,BlindFingerprint bf2)
    {
        this.bf1 = bf1.getBytes();
        this.bf2 = bf2.getBytes();
        this.hashFunc = bf1.getHashFunc();
        this.blindFunc = bf1.getBlindFunc();
    }

    protected Signature getSignature1(RSAPublicKey pub,RSAPrivateKey
priv,RSABlindingFactor factor) throws Exception
    {
        return
Cryptotools.unblind(pub,Cryptotools.blindSign(priv,getFingerprint1()),factor);
    }

    protected Signature getSignature2(RSAPublicKey pub,RSAPrivateKey
priv,RSABlindingFactor factor) throws Exception
    {
        return
Cryptotools.unblind(pub,Cryptotools.blindSign(priv,getFingerprint2()),factor);
    }

    public Signature getAndUnblindSignature1(RSAPublicKey pub, RSABlindingFactor factor)
throws Exception
    {
        return Cryptotools.unblind(pub,new
BlindSignature(hashFunc,blindFunc,blindSignature1),factor);
    }

    public Signature getAndUnblindSignature2(RSAPublicKey pub, RSABlindingFactor factor)
throws Exception
    {
        return Cryptotools.unblind(pub,new
BlindSignature(hashFunc,blindFunc,blindSignature2),factor);
    }

    public void blindSign(RSAPrivateKey privateKey) throws Exception
    {
        Signature b1 = Cryptotools.blindSign(privateKey, getFingerprint1());
        Signature b2 = Cryptotools.blindSign(privateKey, getFingerprint2());

        this.blindSignature1 = b1.getBytes();
        this.blindSignature2 = b2.getBytes();
        this.signatureHashFunc = b1.getHashFunc();
    }

    private BlindFingerprint getFingerprint1()
    {
        return new BlindFingerprint(hashFunc,blindFunc,bf1);
    }

    private BlindFingerprint getFingerprint2()
    {
        return new BlindFingerprint(hashFunc,blindFunc,bf2);
    }
}
```

10.2.2 Coin

```
package SpamCash.Currency;

import java.io.Serializable;
import java.util.GregorianCalendar;
import org.logi.crypto.sign.Signature;
import SpamCash.Utilities.Cryptotools;
import java.security.interfaces.*;
import org.logi.crypto.sign.Fingerprint;
import java.security.cert.Certificate;
import org.logi.crypto.keys.RSABlindingFactor;
import org.logi.crypto.sign.BlindFingerprint;
import SpamCash.Utilities.*;
import SpamCash.Utilities.Debugger;
import java.io.*;
import java.util.Date;
import SpamCash.Client.Control;
import java.util.StringTokenizer;
import java.security.cert.X509Certificate;
import java.security.cert.*;

public class Coin implements Serializable
{
    public final static int A_TYPE = 1, B_TYPE = 2, C_TYPE = 3;
    private int type;

    //These are signed by the CES
    private double value;
    private long serialNumber;
    private long expirationTime;
    private int maxNumberOfUsages;
    private byte[] signature;
    private String signHash;

    private long transferExpirationTime;
    public Transactionlist transactionList;

    public Coin(double value, long serialNumber, long expirationTime, int
maxNumberOfUsages,
                Signature signature,
                Transactionlist transactionList)
    {
        this.type = A_TYPE;
        this.value = value;
        this.serialNumber = serialNumber;
        this.expirationTime = expirationTime;
        this.maxNumberOfUsages = maxNumberOfUsages;

        this.transferExpirationTime = 0;
        this.transactionList = transactionList;
        if(signature != null)
        {
            this.signature = signature.getBytes();
            this.signHash = signature.getHashFunc();
        }
    }

    public Coin(Coin a)
    {
        this.type = a.type;
        this.value = a.value;
        this.serialNumber = a.serialNumber;
        this.expirationTime = a.expirationTime;
        this.maxNumberOfUsages = a.maxNumberOfUsages;
        this.signature = new byte[a.signature.length];
        for(int i = 0; i < a.signature.length; i++)
            this.signature[i] = a.signature[i];

        this.signHash = new String(a.signHash);
```

```
        this.transferExpirationTime = a.transferExpirationTime;
        if (a.transactionList instanceof UnencryptedTransactionlist)
            this.transactionList = new
UnencryptedTransactionlist((UnencryptedTransactionlist)a.transactionList);
        else
            this.transactionList = new
EncryptedTransactionlist(((EncryptedTransactionlist)a.transactionList));
    }

    public Coin(byte[] data) throws Exception
    {
        ByteArrayInputStream bis = new ByteArrayInputStream(data);
        ObjectInputStream ois = new ObjectInputStream(bis);

        this.type = ((Integer)ois.readObject()).intValue();
        this.value = ((Double)ois.readObject()).doubleValue();
        this.serialNumber = ((Long)ois.readObject()).longValue();
        this.expirationTime = ((Long)ois.readObject()).longValue();
        this.transferExpirationTime = ((Long)ois.readObject()).longValue();
        this.maxNumberOfUsages = ((Integer)ois.readObject()).intValue();
        Object input = ois.readObject();
        if (input instanceof EncryptedTransactionlist)
            this.transactionList = (EncryptedTransactionlist) input;
        else
            this.transactionList = (UnencryptedTransactionlist) input;
        this.signature = (byte[])ois.readObject();
        this.signHash = ((String)ois.readObject());
    }

    public byte[] getCompressed() throws IOException
    {
        ByteArrayOutputStream outStream = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(outStream);
        out.writeObject(new Integer(type));
        out.writeObject(new Double(value));
        out.writeObject(new Long(serialNumber));
        out.writeObject(new Long(expirationTime));
        out.writeObject(new Long(transferExpirationTime));
        out.writeObject(new Integer(this.maxNumberOfUsages));
        out.writeObject(this.transactionList);
        out.writeObject(this.signature);
        out.writeObject(this.signHash);

        byte[] result = outStream.toByteArray();
        outStream.close();
        out.close();
        return result;
    }

    private void setCExpirationTime()
    {
        //Expirationtime is calculated
        transferExpirationTime = System.currentTimeMillis();
        transferExpirationTime += Config.CCoinExpirationTimeInMilliseconds  +
Config.CCoinDelayTimeInMilliseconds;
        //Debugger.debug(getClass(),1,"C_expirationtime set to:
"+C_expirationTime.getTime());
    }

    private void setBExpirationTime()
    {
        transferExpirationTime = System.currentTimeMillis();
        transferExpirationTime += Config.CCoinExpirationTimeInMilliseconds;
    }

    public boolean transferTimeexpired()
    {
        return System.currentTimeMillis() > transferExpirationTime;
    }

    public void insertValues(double value, int maxUsages)
```

```
    {
        this.value = value;
        this.maxNumberOfUsages = maxUsages;
    }

    public Fingerprint getHash1() throws Exception
    {
        return Cryptotools.merge(new Integer(this.maxNumberOfUsages),
                                 Cryptotools.merge(new Long(this.serialNumber),
Cryptotools.SHA1(new Double(value))));
    }

    public static Fingerprint getHash2(long serialNumber, Certificate certificate) throws
Exception
    {
        String info = ((X509Certificate)certificate).getSubjectDN().getName();
        StringTokenizer st = new StringTokenizer(info,"=,");
        st.nextToken();
        String email = st.nextToken();
        return getHash2(serialNumber, email);
    }

    public static Fingerprint getHash2(long serialNumber, String email) throws Exception
    {
        return Cryptotools.merge(email, Cryptotools.SHA1(new Long(serialNumber)));
    }


    /**
     * Creates the two signatures and compares these to the ones in 'blindedCoin'
     */
    public boolean signAndVerify(RSAPublicKey pub, RSAPrivateKey priv, Certificate cert,
BlindedCoin blindedCoin,
                                 RSABlindingFactor factor) throws Exception
    {
        Signature s1 = blindedCoin.getSignature1(pub, priv, factor);
        Signature s2 = blindedCoin.getSignature2(pub, priv, factor);

        return Cryptotools.verify(pub, s1, getHash1()) && Cryptotools.verify(pub, s2,
getHash2(serialNumber,cert));
    }

    public BlindedCoin getBlindedCoin(RSAPublicKey pub, RSABlindingFactor bf, Certificate
certificate) throws Exception
    {
        BlindFingerprint bf1 = Cryptotools.blind(pub, bf, getHash1());
        BlindFingerprint bf2 = Cryptotools.blind(pub, bf,
getHash2(serialNumber,certificate));
        return new BlindedCoin(bf1, bf2);
    }

    public void setTransactionlist(Transactionlist list)
    {
        this.transactionList = list;
    }

    /**
     * Checks the following:
     *  - The CES-signature
     *  - The Tranactionlists signatures
     * @return boolean
     */
    public boolean verify(Certificate RSCert, Certificate CESCert) throws Exception
    {
        return transactionList.verify(RSCert, CESCert,this.serialNumber);
    }

    /**
     * Checks the following:
     *  - The CES-signature
     * @return boolean
```

```
     */
    public boolean verifyCES(Certificate CESCert) throws Exception
    {

        return Cryptotools.verify((RSAPublicKey) CESCert.getPublicKey(), new
Signature(signHash, signature), getHash1());
    }

    public boolean initialVerify(Certificate RSCert, Certificate CESCert,Certificate
cert) throws Exception
    {
        return verifyCES(CESCert) &&
transactionList.initialVerify(RSCert,CESCert,getHash2(serialNumber,cert));
    }

    public void setCESSignature(Signature s)
    {
        signature = s.getBytes();
        signHash = s.getHashFunc();
    }

    public boolean equals(Coin a)
    {
        boolean test = value == a.value && serialNumber == a.serialNumber &&
            expirationTime == a.expirationTime &&
            transferExpirationTime == a.transferExpirationTime &&
            maxNumberOfUsages == a.maxNumberOfUsages;
        return test && Cryptotools.equals(signature, a.signature) &&
transactionList.equals(a.transactionList);
    }

    public Transactionlist getTransactionlist()
    {
        return transactionList;
    }

    public double getValue()
    {
        return value;
    }

    public int getType()
    {
        return type;
    }

    public int getMaxNumberOfUsages()
    {
        return maxNumberOfUsages;
    }

    public void setType(int type)
    {
        if(type == C_TYPE)
            setCExpirationTime();
        else if(type == B_TYPE)
            setBExpirationTime();
        else if(type == A_TYPE)
            transferExpirationTime = 0;
        this.type = type;
    }

    public long getSerialNumber()
    {
        return serialNumber;
    }

    public void decryptTransactionlist(RSAPrivateKey key) throws Exception
    {
        if(this.transactionList instanceof EncryptedTransactionlist)
        {
```

```
            this.transactionList = new UnencryptedTransactionlist
(((EncryptedTransactionlist)this.transactionList),key);
            return;
        }
        throw new Exception("Cannot decrypt. The list is not encrypted.");
    }

    public void encryptTransactionlist(RSAPublicKey key) throws Exception
    {
        if (this.transactionList instanceof UnencryptedTransactionlist)
        {
            this.transactionList = new EncryptedTransactionlist
((UnencryptedTransactionlist)this.transactionList, key);
            return;
        }
        throw new Exception("Transactionlist is already encrypted.");
    }

    public String printSizes() throws Exception
    {
        String s="";
        int numbers = (4+8+8+4);
        int extime = 8;
        int sign = signature.length+signHash.length();
        int cextime  = 8;

        s += "    Numbers:              "+numbers+" bytes\n";
        s += "    ExpirationTime:       "+extime+" bytes\n";
        s += "    Signature:            "+sign+" bytes\n";
        s += "    Coin expirationTime:  "+cextime+" bytes\n";
        s += "    Transactionlist:  "+transactionList.printSize();
        s += "    TransactionList total "+transactionList.getSize()+" bytes\n";
        s += "    Total:
"+(numbers+extime+sign+cextime+transactionList.getSize())+" bytes\n";

        return s;

    }

    public int size() throws Exception
    {
        int numbers = (4+8+8+4);
        int extime = 8;
        int sign = signature.length + signHash.length();
        int cextime = 8;
        int tlist = transactionList.getSize();

        return (numbers + extime + sign + cextime + tlist);
    }

    public String toString()
    {
        return "Coin:\n\t value = " + value + "\n\t SerialNumber:" + serialNumber + "\n\t
ExpirationTime = "+ new Date(expirationTime) + "\n\t transferExpirationTime = "+ new
Date(transferExpirationTime) + "\n\t maxUsages = " + maxNumberOfUsages + "\n" +new
Signature(signHash,signature)+"\n" + transactionList;
    }

    public static void main(String[] sa)
    {
        long expTime = System.currentTimeMillis();
        System.out.println("Current time: long="+expTime+" Date="+new Date(expTime));
        expTime += Config.expirationTimeInMilliseconds;
        System.out.println("+30days time: long="+expTime+" Date="+new Date(expTime));
        System.out.println("0-time      : long="+0+" Date="+new Date(0));
        System.out.println("Config time : long="+Config.expirationTimeInMilliseconds+"
Date="+new Date(Config.expirationTimeInMilliseconds));
        System.out.println("number time : long="+(100*24*60*60*1000)+" Date="+new
Date(100*(24*60*60*1000)));

        Coin a = new Coin(5.0, 1241324, expTime , 10, null, null);
```

```
        System.out.println("Coin:"+ new Date(a.expirationTime));

        //Testing encryption...
            try
            {
                Certificate cert = FileHandler.importCertificate(new
File(Config.BLSCertificateName));
                RSAPublicKey publicKey = (RSAPublicKey) cert.getPublicKey();
                RSAPrivateKey privateKey = (RSAPrivateKey)FileHandler.readObject(new
File(Config.BLSPrivateKeyName));

                //Just a random coin...
                Coin orig = new Coin(5.0, 1241324, expTime , 10,
Cryptotools.sign(privateKey,a), new UnencryptedTransactionlist(new
TransactionlistElement(Cryptotools.sign(privateKey,a),cert)));

                System.out.println("Original coin: "+orig);
                orig.encryptTransactionlist(publicKey);
                System.out.println("Encrypted coin: "+orig);
                orig.decryptTransactionlist(privateKey);
                System.out.println("Restored coin: "+orig);
            }
            catch(Exception ex)
            {
                System.out.println("Error: "+ex);
                ex.printStackTrace();
            }
    }
}
```

10.2.3 EncryptedTransactionlist

```
package SpamCash.Currency;


import java.security.interfaces.RSAPublicKey;
import javax.crypto.SecretKey;
import SpamCash.Utilities.Cryptotools;
import java.security.cert.Certificate;
import org.logi.crypto.sign.Fingerprint;
import java.security.interfaces.RSAPrivateKey;
import SpamCash.NetworkPackets.CryptoPacket;
import SpamCash.Utilities.FileHandler;

public class EncryptedTransactionlist implements Transactionlist,java.io.Serializable
{
    public byte[] encryptedTransactionlist;
    public CryptoPacket encryptedAESKey;

    public EncryptedTransactionlist(EncryptedTransactionlist list)
    {
        for (int i=0;i<list.encryptedTransactionlist.length;i++)
            this.encryptedTransactionlist[i] = list.encryptedTransactionlist[i];


        for (int i=0;i<list.encryptedAESKey.data.length;i++)
            this.encryptedAESKey.data[i] = list.encryptedAESKey.data[i];
    }

    public EncryptedTransactionlist(UnencryptedTransactionlist list, RSAPublicKey pubkey)
throws Exception
    {
        SecretKey AESKey = Cryptotools.generateAESKey();
        encryptedTransactionlist =
Cryptotools.encryptAES(FileHandler.writeObject(list.list),AESKey);
        encryptedAESKey = Cryptotools.encryptRSA(AESKey,pubkey);
    }

 public void addElement(TransactionlistElement elem)
```

```
 {
     return;
 }

 public boolean verify(Certificate RSCert, Certificate CESCert, long serialNumber) throws
Exception
 {
     throw new Exception("error, transactionlist is encrypted.");
 }

 public boolean initialVerify(Certificate RSCert, Certificate CESCert,Fingerprint hash)
throws Exception
 {
     throw new Exception("error, transactionlist is encrypted.");
 }

 public Fingerprint hashList() throws Exception
 {
     throw new Exception("error, transactionlist is encrypted.");
 }

 private Fingerprint hashList(int size) throws Exception
 {
     throw new Exception("error, transactionlist is encrypted.");
 }

 public void signList(Certificate cert,RSAPrivateKey key) throws Exception
 {
     throw new Exception("error, transactionlist is encrypted.");
 }

 public int numberOfAppearances(Certificate cert) throws Exception
 {
      throw new Exception("error, transactionlist is encrypted.");
  }

 public boolean equals(Transactionlist list)
 {
     if (list instanceof EncryptedTransactionlist)
         return
Cryptotools.equals(this.encryptedTransactionlist,((EncryptedTransactionlist)list).encrypt
edTransactionlist);
     return false;
 }

 public TransactionlistElement elementAt(int index) throws Exception
 {
    throw new Exception("error, transactionlist is encrypted.");
 }

 public int length()
 {
     return 0;
 }

 public int getSize()
 {
     return 0;
 }

 public String printSize()
 {
   return "0";
 }

 public String toString()
 {
     return Cryptotools.byteToHexString(this.encryptedTransactionlist);
 }
```

```
}
```

10.2.4 Transactionlist

```
package SpamCash.Currency;


import java.util.Vector;

import java.security.cert.Certificate;
import org.logi.crypto.sign.Fingerprint;
import java.security.interfaces.RSAPrivateKey;
import SpamCash.Currency.TransactionlistElement;

public interface Transactionlist
{
    public void addElement(TransactionlistElement elem);
    public boolean verify(Certificate RCCert, Certificate CESCert, long serialNumber)
throws Exception;
    public boolean initialVerify(Certificate RSCert, Certificate CESCert,Fingerprint
hash) throws Exception;
    public Fingerprint hashList() throws Exception;
    public void signList(Certificate cert, RSAPrivateKey key) throws Exception;
    public int numberOfAppearances(Certificate cert) throws Exception;
    public boolean equals(Transactionlist list);
    public TransactionlistElement elementAt(int index) throws Exception;
    public int length();
    public int getSize();
    public String printSize();
    public String toString();
}
```

10.2.5 TransactionlistElement

```
package SpamCash.Currency;


import java.security.cert.Certificate;
import org.logi.crypto.sign.Fingerprint;
import org.logi.crypto.sign.Signature;
import SpamCash.Utilities.Cryptotools;
import java.io.ObjectInputStream;
import java.io.InputStream;
import java.io.ByteArrayInputStream;
import SpamCash.Utilities.Debugger;
import java.security.interfaces.RSAPublicKey;
import java.security.cert.CertificateEncodingException;
import SpamCash.Utilities.FileHandler;
import java.security.cert.X509Certificate;

public class TransactionlistElement implements java.io.Serializable
{
    //Signature
    private byte[] signature;
    private static String signHashFunc;

    //Certificate
    private Certificate cert;

    public TransactionlistElement(Signature signature, Certificate cert)
    {
        this.signature = signature.getBytes();
        this.signHashFunc = signature.getHashFunc();

        if(cert != null)
            this.cert = cert;
    }
```

```
    public TransactionlistElement(TransactionlistElement elem) throws Exception
    {
        this.signature = copy(elem.signature);

        if(elem.cert != null)
            this.cert =
(Certificate)FileHandler.readObject(FileHandler.writeObject(elem.cert));
    }

    public Certificate getCertificate()
    {
        return cert;
    }

    private byte[] copy(byte[] data)
    {
        byte[] result = new byte[data.length];
        for(int i = 0; i < data.length; i++)
        {
            result[i] = data[i];
        }

        return result;
    }

    public Signature getSignature()
    {
        return new Signature(signHashFunc, signature);
    }

    protected boolean verify(Certificate RSCert, Certificate cert, Fingerprint hashed)
throws Exception
    {
        try
        {
            cert.verify(RSCert.getPublicKey());
        }
        catch(Exception e)
        {
            Debugger.debug(getClass(), 1, "Could not verify certificate, no valid
RSsignature found",e);
            return false;
        }

        return Cryptotools.verify((RSAPublicKey) cert.getPublicKey(), getSignature(),
hashed);
    }

    public boolean equals(TransactionlistElement e)
    {
        if(Cryptotools.equals(signature, e.signature))
        {
            try
            {
                if( ! cert.equals(e.cert))
                    return false;
            }
            catch(NullPointerException ex)
            {
                if( ! (cert == null && e.cert == null))
                    return false;
            }
            return true;
        }
        return false; //signature mismatch
    }

    public int getSize()
    {
        try
        {
```

```
            //System.out.println("signature: "+signature.length);
            //System.out.println("cert size: "+cert.getEncoded().length);
            //System.out.println("cert: "+cert);
            return cert == null ? signature.length : signature.length +
cert.getEncoded().length;
        }
        catch(CertificateEncodingException ex)
        {
        }
        return -1;
    }

    public String toString()
    {
        String s = "";
        if(cert != null)
            s += "Certificate identification: " + ((X509Certificate)
cert).getSubjectDN().getName() + "\t";
        else
            s += "Certificate identification: none \t";

        if(signature != null)
            s += "Signature: " + /*Cryptotools.byteToHexString(signature)*/signature[0] +
"\n";
        else
            s += "Signature: none\n\n";

        return s;
    }
}
```

10.2.6 UnencryptedTransactionlist

```
package SpamCash.Currency;


import java.util.Vector;

import SpamCash.Utilities.Cryptotools;
import SpamCash.Utilities.Debugger;
import java.security.cert.Certificate;
import java.security.interfaces.RSAPublicKey;
import org.logi.crypto.sign.Fingerprint;
import java.util.Enumeration;
import java.security.interfaces.RSAPrivateKey;
import org.logi.crypto.sign.Signature;
import SpamCash.Utilities.FileHandler;
import SpamCash.Client.MailHandler.MailFunctions;
import java.security.cert.X509Certificate;
import java.util.StringTokenizer;
import java.security.Security;
import javax.crypto.SecretKey;
import SpamCash.Currency.EncryptedTransactionlist;

public class UnencryptedTransactionlist implements Transactionlist,java.io.Serializable
{
    public Vector list;

    public UnencryptedTransactionlist(TransactionlistElement elem)
    {
        list = new Vector();
        addElement(elem);
    }

    public UnencryptedTransactionlist(UnencryptedTransactionlist tlist)
    {
        this.list = new Vector();
        for(int i = 0; i < tlist.list.size(); i++)
        {
```

```
            try
            {
                this.list.addElement(new TransactionlistElement(tlist.elementAt(i)));
            }
            catch(Exception ex)
            {
                Debugger.debug(getClass(), 1, "Error copyconstructing Transactionlist");
            }
        }
    }

    public UnencryptedTransactionlist(EncryptedTransactionlist list, RSAPrivateKey
privkey) throws Exception
    {
        SecretKey AESKey = Cryptotools.decryptRSA(list.encryptedAESKey, privkey);
        this.list = (Vector)
FileHandler.readObject(Cryptotools.decryptAES(list.encryptedTransactionlist, AESKey));
    }

    public void addElement(TransactionlistElement elem)
    {
        list.addElement(elem);
    }

    public boolean verify(Certificate RSCert, Certificate CESCert, long serialNumber)
throws Exception
    {
        Debugger.debug(getClass(), 3, "Verifying list: \n" + this.toString());

        //Verify CES-certificate
        CESCert.verify(RSCert.getPublicKey());

        //The number of elements must be even
        if(list.size() % 2 == 1)
        {
            Debugger.debug(getClass(), 1, "Number of elements in the list is odd.
Returning false.");
            return false;
        }

        TransactionlistElement elem,oldElem;
        Fingerprint hashedlist;

        for(int i = 0; i < list.size(); i++)
        {
            elem = (TransactionlistElement)this.list.elementAt(i);

            //if first element check for CES signature.
            if(i==0)
                oldElem = new TransactionlistElement(elem.getSignature(),CESCert);
            else
                oldElem = (TransactionlistElement)this.list.elementAt(i - 1);

            if(i % 2 == 1) //element of type 2
            {
                hashedlist = hashList(i);
                Debugger.debug(getClass(), 3, "Hash value at index " + i + " : " +
hashedlist.toString());
                if(!elem.verify(RSCert, elem.getCertificate(), hashedlist))
                {
                    Debugger.debug(getClass(), 1,
                                   "Could not verify transactionlist-element at index
(list signature): " + i);
                    return false;
                }
            }
            else //element of type 1
            {
                if(!elem.verify(RSCert, oldElem.getCertificate(),
                                Coin.getHash2(serialNumber,
```

```
                                                ((TransactionlistElement) list.elementAt(i
+ 1)).getCertificate())))
                {
                    Debugger.debug(getClass(), 1,
                                   "Could not verify transactionlist-element at index
(receiver signature): " + i);
                    return false;
                }
            }

        }
        return true;
    }

    public boolean initialVerify(Certificate RSCert, Certificate CESCert, Fingerprint
hash) throws Exception
    {
        TransactionlistElement elem = (TransactionlistElement) list.elementAt(0);

        if(!Cryptotools.verify((RSAPublicKey) CESCert.getPublicKey(),
elem.getSignature(), hash))
        {
            Debugger.debug(getClass(), 1, "Could not verify CES signature on first
element in list");
            return false;
        }
        return true;
    }

    public Fingerprint hashList() throws Exception
    {
        return hashList(list.size());
    }

    private Fingerprint hashList(int size) throws Exception
    {
        Fingerprint hash;
        try
        {
            hash = Cryptotools.SHA1((TransactionlistElement) list.elementAt(0));
            for(int i = 1; i < size; i++)
                hash = Cryptotools.merge((TransactionlistElement) list.elementAt(i),
hash);
            return hash;
        }
        catch(Exception ex)
        {
            throw new Exception("Error occured while trying to hash (part of) the
transactionlist");
        }
    }

    public void signList(Certificate cert, RSAPrivateKey key) throws Exception
    {
        Fingerprint listHash = hashList();
        Debugger.debug(getClass(), 3, "Hash value               : " + listHash);
        Signature listSignature = Cryptotools.sign(key, listHash);
        list.addElement(new TransactionlistElement(listSignature, cert));
    }

    public int numberOfAppearances(Certificate cert) throws Exception
    {
        TransactionlistElement elem;
        int totalFound = 0;

        //Look for cert in the transactionlist, but only in the elements with an odd
index (type 2)
        for(int i = 1; i < list.size(); i = i + 2)
        {
            elem = (TransactionlistElement) list.elementAt(i);
```

```
            if(elem.getCertificate().equals(cert))
                totalFound++;
        }

        return totalFound;
    }

    public boolean equals(Transactionlist a)
    {
        if(list.size() != ((UnencryptedTransactionlist)a).list.size())
            return false;

        for(int i = 0; i < list.size(); i++)
        {

            if(!((TransactionlistElement)
list.elementAt(i)).equals((TransactionlistElement)
((UnencryptedTransactionlist)a).list.elementAt(i)))
                return false;
        }
        return true;
    }

    public TransactionlistElement elementAt(int index) throws Exception
    {
        return(TransactionlistElement) list.elementAt(index);
    }

    public int length()
    {
        return list.size();
    }

    public int getSize()
    {
        int size = 0;
        for(int i = 0; i < this.list.size(); i++)
            size += ((TransactionlistElement) list.elementAt(i)).getSize();
        return size;
    }

    public String printSize()
    {
        String s = "\n";
        for(int i = 0; i < this.list.size(); i++)
            s += "       Element " + i + " size " + ((TransactionlistElement)
list.elementAt(i)).getSize() + "\n";
        return s;
    }

    public String toString()
    {
        String s = "\tTransactionlist:\n ";
        for(int i = 0; i < list.size(); i++)
        {
            s += "\tElement " + i + ": " + ((TransactionlistElement) list.elementAt(i));
        }
        return s;
    }
}
```

10.3 NetworkPackets

10.3.1 ActivationCompletedPacket

```
package SpamCash.NetworkPackets;
```

```
import java.security.cert.Certificate;

public class ActivationCompletedPacket extends NetPacket
{
    public Certificate certificate;
    public ActivationCompletedPacket(Certificate certificate)
    {
        this.certificate = certificate;
    }

    public String toString()
    {
        return("" + getClass()).substring(6) +
certificate.toString().substring(0,30)+"...";
    }

}
```

10.3.2 ActivationPacket

```
package SpamCash.NetworkPackets;


import org.logi.crypto.sign.Signature;
import SpamCash.Utilities.FileHandler;

public class ActivationPacket extends NetPacket
{
    public long activationNumber;
    private byte[] signature;
    private String hashFunction;


    public ActivationPacket(long activationNumber, Signature signature) throws Exception
    {
        this.activationNumber = activationNumber;
        this.hashFunction = signature.getHashFunc();
        this.signature = signature.getBytes();
    }

    public Signature getSignature()
    {
        return new Signature(hashFunction, signature);
    }

    public String toString()
    {
        return("" + getClass()).substring(6) + " Signature: "
            + (new Signature(hashFunction, signature)+"").substring(0,20)+"...";
    }
}
```

10.3.3 AmountDepositedPacket

```
package SpamCash.NetworkPackets;


public class AmountDepositedPacket extends NetPacket
{
    public double amount;

    public AmountDepositedPacket(double amount)
    {
        this.amount = amount;
    }
```

```
    public String toString()
    {
        return (""+getClass()).substring(6) + ": amount = "+amount;
    }
}
```

10.3.4 BlacklistPacket

```
package SpamCash.NetworkPackets;


import SpamCash.Servers.BLS.Blacklist;


public class BlacklistPacket extends NetPacket
{
    public Blacklist blacklist;

    public BlacklistPacket(Blacklist bl)
    {
        blacklist=bl;
    }

    public String toString()
    {
        return (""+getClass()).substring(6) +blacklist;
    }
}
```

10.3.5 CoinSignaturePacket

```
package SpamCash.NetworkPackets;


import SpamCash.Currency.BlindedCoin;


public class CoinSignaturePacket extends NetPacket
{
    public BlindedCoin[] blindedcoins;

    public CoinSignaturePacket(BlindedCoin[] blindedcoins)
    {
        this.blindedcoins = blindedcoins;
    }

    public String toString()
    {
        String s ="";
        for(int i=0;i<blindedcoins.length;i++)
            s += ", "+blindedcoins[i];

      return (""+getClass()).substring(6) + " Signatures: "+ s;
    }

}
```

10.3.6 CryptoPacket

```
package SpamCash.NetworkPackets;


import java.net.InetAddress;
import SpamCash.Utilities.Debugger;
```

```
public class CryptoPacket extends NetPacket implements java.io.Serializable
{
    public byte[] data;
    public InetAddress senderAddress;

    public CryptoPacket(byte[] data)
    {
        this.data = data;
        try
        {
            senderAddress = InetAddress.getLocalHost();
        }
        catch(Exception e)
        {
            Debugger.debug(getClass(),3,"Cryptopacket error:",e);
        }

    }

    public String toString()
    {
        StringBuffer strbuf = new StringBuffer(data.length * 2);
        int i;
        for(i = 0; i < data.length; i++)
        {
            if(((int) data[i] & 0xff) < 0x10)
                strbuf.append("0");
            strbuf.append(Long.toString((int) data[i] & 0xff, 16));
        }
        return (""+getClass()).substring(6) + ": "+strbuf.toString();
    }

}
```

10.3.7 CurrencyBlindingFactorPacket

```
package SpamCash.NetworkPackets;


import java.security.cert.Certificate;
import org.logi.crypto.keys.RSABlindingFactor;
import SpamCash.Currency.Coin;
import java.math.BigInteger;

public class CurrencyBlindingFactorPacket extends NetPacket
{
    private BigInteger[][] bigIntegers;
    public Coin[][] coins;
    public Certificate certificate;

    public CurrencyBlindingFactorPacket(RSABlindingFactor[][] blindingFactors,Coin[][]
coins,Certificate certificate)
    {
        bigIntegers = new BigInteger[blindingFactors.length][blindingFactors[0].length];

        for(int i=0;i<blindingFactors.length;i++)
            for(int j=0;j<blindingFactors[0].length;j++)
            {
                if(blindingFactors[i][j] != null)
                    bigIntegers[i][j] = blindingFactors[i][j].getFactor();
            }

        this.coins = coins;
        this.certificate = certificate;
    }

    public RSABlindingFactor[][] getRSABlindingFactors()
    {
```

```
        RSABlindingFactor[][] rsabf = new
RSABlindingFactor[bigIntegers.length][bigIntegers[0].length];

        for(int i=0;i<bigIntegers.length;i++)
            for(int j=0;j<bigIntegers[0].length;j++)
            {
                if(bigIntegers[i][j] != null)
                    rsabf[i][j] = new RSABlindingFactor(bigIntegers[i][j]);
            }

        return rsabf;
    }

    public RSABlindingFactor getRSABlindingFactor(int c,int r)
    {
        return new RSABlindingFactor(bigIntegers[c][r]);
    }

    public String toString()
    {
        String bf = "BlindingFactors: ",c=" Coins: ";

        boolean simple = true;

        RSABlindingFactor[][] rsabf = getRSABlindingFactors();

        for(int i=0;i<rsabf.length;i++)
        {
            for(int j = 0; j < rsabf[0].length; j++)
            {
                if(!simple)
                {
                    bf += " " + rsabf[i][j];
                    c += "\n   " + coins[i][j];
                }
                else
                {
                    bf += " " + (rsabf[i][j] == null ? "0" : "1");
                    c += " " + (coins[i][j] == null ? "0" : "1");
                }
            }
        }

        return("" + getClass()) + " Certificate: "+(certificate == null ? "null" :
""+certificate.getClass()) + "\n"+ bf + "\n"+ c;
    }
}
```

10.3.8 CurrencyExchangeRequestPacket

```
package SpamCash.NetworkPackets;



import SpamCash.Currency.*;
import org.logi.crypto.sign.BlindFingerprint;

public class CurrencyExchangeRequestPacket extends NetPacket
{
    private BlindedCoin[][] coins;

    public CurrencyExchangeRequestPacket(BlindedCoin[][] coins)
    {
        this.coins = coins;
    }

    public BlindedCoin[][] getCoins()
    {
        return coins;
```

```
    }

    public String toString()
    {
        String bf = "BlindingFactors: ";
        BlindedCoin[][] coins = getCoins();
        for(int i = 0; i < coins.length; i++)
        {
            for(int j = 0; j < coins[0].length; j++)
            {
                bf += " " + coins[i][j];
            }
        }

       return (""+getClass()).substring(6)+ bf;
    }
}
```

10.3.9 DepositRequestPacket

```
package SpamCash.NetworkPackets;


import SpamCash.Currency.Coin;

public class DepositRequestPacket extends NetPacket
{
    public Coin[] coins;

    public DepositRequestPacket(Coin[] coins)
    {
        this.coins = coins;
    }

    public String toString()
    {
        String s = "Coins: \n";
        for(int i = 0; i < coins.length; i++)
            s += "\n    " + coins[i];

        return("" + getClass()).substring(6);// + s;

    }

}
```

10.3.10 GetBlacklistPacket

```
package SpamCash.NetworkPackets;


public class GetBlacklistPacket extends NetPacket
{
    public int splitIndex;

    public GetBlacklistPacket(int splitIndex)
    {
        this.splitIndex = splitIndex;
    }

    public String toString()
    {
        return("" + getClass()).substring(6)+" splitindex = "+splitIndex;
    }
}
```

10.3.11 LogonPacket

```
package SpamCash.NetworkPackets;

public class LogonPacket extends NetPacket
{
    public String username;
    public String password;

    public LogonPacket(String username, String password)
    {
        this.username = username;
        this.password = password;
    }

    public boolean equals(LogonPacket p)
    {
        return this.username.equals(p.username) && this.password.equals(p.password);
    }

    public String toString()
    {
        return (""+getClass()).substring(6)+": username: "+username+" password:
"+password;
    }
}
```

10.3.12 NetPacket

```
package SpamCash.NetworkPackets;

import java.net.InetAddress;

public class NetPacket implements java.io.Serializable
{
        public NetPacket()
        {
        }

      public String toString()
      {
        return (""+getClass()).substring(6);
      }
}
```

10.3.13 RegistrationInfoPacket

```
package SpamCash.NetworkPackets;


import java.security.cert.X509Certificate;

public class RegistrationInfoPacket extends NetPacket
{
    public long activationNumber;

    public RegistrationInfoPacket(long activationNumber)
    {
        this.activationNumber = activationNumber;
    }

    public RegistrationInfoPacket(RegistrationInfoPacket p)
    {
        this.activationNumber = p.activationNumber;
    }
```

```
    public String toString()
    {
      return (""+getClass()).substring(6) + "ActivationNumber = "+activationNumber;
    }
}
```

10.3.14 RegistrationRequestPacket

```
package SpamCash.NetworkPackets;


public class RegistrationRequestPacket extends NetPacket
{
    public byte[] certificateSigningRequest;

    public RegistrationRequestPacket(byte[] certificateSigningRequest)
    {
        this.certificateSigningRequest = certificateSigningRequest;
    }

    public String toString()
    {
        return (""+getClass()).substring(6) + " requestarray =
"+certificateSigningRequest;
    }
}
```

10.3.15 ReportCoinsPacket

```
package SpamCash.NetworkPackets;

import SpamCash.Currency.Coin;

public class ReportCoinsPacket extends NetPacket
{
    public Coin a,b;

  public ReportCoinsPacket(Coin a,Coin b)
  {
      this.a = a;
      this.b = b;
  }
       public String toString()
    {
        return (""+getClass()).substring(6) + " Coin a: "+a+" Coin b: "+b;
    }
}
```

10.3.16 RequestBlindingFactorsPacket

```
package SpamCash.NetworkPackets;


public class RequestBlindingFactorsPacket extends NetPacket
{
    public int[] index;

    public RequestBlindingFactorsPacket(int[] index)
    {
        this.index = index;
    }

    public String toString()
    {
        String s = " blindingFactorIndexes = [";
```

```
        for(int i=0;i<index.length;i++)
            s += " "+index[i];
        return (""+getClass()).substring(6) +s+" ]";
    }

}
```

10.3.17 StatusPacket

```
package SpamCash.NetworkPackets;


public class StatusPacket extends NetPacket
{
    public boolean ok;
    public String message;
    public double amount;
    public long nonce;

    public StatusPacket(boolean ok,String message)
    {
        this.ok = ok;
        this.message = message;
    }

    public StatusPacket(boolean ok,double amount)
    {
        this.ok = ok;
        this.amount = amount;
    }

    public StatusPacket(boolean ok,String message,long nonce)
    {
        this.message = message;
        this.ok = ok;
        this.nonce = nonce;
    }

    public String toString()
    {
        return("" + getClass()).substring(6) + ": ok:" + ok + " message: " + message;
    }

}
```

10.3.18 VerificationPacket

```
package SpamCash.NetworkPackets;


import SpamCash.Currency.Coin;
import java.security.cert.Certificate;

public class VerificationPacket extends NetPacket
{
    public Coin coin;
    public Certificate certificate;
    public long nonce;

    public VerificationPacket(Coin coin, Certificate cert, long nonce)
    {
        this.coin = coin;
        certificate = cert;
        this.nonce = nonce;
    }

    public String toString()
    {
```

```
        return("nonce: "+nonce);
    }
}
```

10.4 Servers

10.4.1 AbstractServer

```
package SpamCash.Servers;


import SpamCash.Servers.ReceiveThread;
import java.security.Key;
import SpamCash.Utilities.*;
import java.security.interfaces.*;
import java.io.File;
import SpamCash.Utilities.Debugger;

public abstract class AbstractServer
{
    private static ReceiveThread rces;
    public static RSAPublicKey publicKey;
    public static RSAPrivateKey privateKey;

    public AbstractServer(String certificateName, String privateKeyName, int port) throws
Exception
    {
        //Public and private keys
        this.publicKey = (RSAPublicKey) FileHandler.importCertificate(new
File(certificateName)).getPublicKey();
        this.privateKey = (RSAPrivateKey) FileHandler.readObject(new
File(privateKeyName));

        ShutdownHook shutdownHook = new ShutdownHook();
        Runtime.getRuntime().addShutdownHook(shutdownHook);

        //Testing the keys
        String s = "The asymmetric keys are a pair, and passed the cryptographic test.";
        s = (String)
FileHandler.readObject(Cryptotools.decryptRSA(Cryptotools.encryptRSA(FileHandler.writeObj
ect(s),

publicKey), privateKey));
        Debugger.debug(getClass(), 2, s);

        //instantiate and start the ReceiverThread which will handle all incoming
connections to the CurrencyExchangeServer
        rces = new ReceiveThread(this, port, privateKey);
        rces.start();

        Debugger.debug(getClass(), 1, "Server ready to receive on port: " + port);
    }

    public abstract void shutdown() throws Exception;

    class ShutdownHook extends Thread
    {
        public void run()
        {
            try
            {
                shutdown();
            }
            catch(Exception e)
            {
                Debugger.debug(getClass(), 1, "Could not save configuration.");
            }

            Debugger.debug(getClass(), 1, "Shutting down");
        }
```

```
    }
}
```

10.4.2 CommunicationThread

```
package SpamCash.Servers;


import java.net.*;
import SpamCash.Utilities.Debugger;
import SpamCash.Utilities.session;
import java.security.Key;

public class CommunicationThread extends Thread
{
    protected session clientSession;
    protected boolean loggedIn;

    public CommunicationThread(Socket sock, Key key, boolean useEncryption)
    {
        try
        {
            this.clientSession = new session(sock, true, key, useEncryption);
        }
        catch(Exception ex)
        {
            Debugger.debug(getClass(), 1, ex.getMessage(),ex);
        }
    }
}
```

10.4.3 ReceiveThread

```
package SpamCash.Servers;


import java.net.*;
import java.io.*;

import SpamCash.Servers.CES.*;
import SpamCash.Servers.BLS.*;
import SpamCash.Servers.RS.*;
import SpamCash.Servers.VS.*;
import SpamCash.Utilities.Debugger;
import java.security.Key;

public class ReceiveThread extends Thread
{
    private Socket sock;
    private ServerSocket sersock;
    private int PORT;
    private Key key;
    private AbstractServer server;

    public ReceiveThread(AbstractServer server,int port, Key key)
    {

        try
        {
            this.server = server;
            this.PORT = port;
            this.key = key;
            sersock = new ServerSocket();
            sersock.bind(new InetSocketAddress(PORT));
        }
```

```
        catch(IOException ex)
        {
            Debugger.debug(getClass(), 1, "Initialization exception...", ex);
        }
    }

    public void run()
    {

        while(true)
        {
            try
            {
                Debugger.debug(getClass(),3,"Waiting for connection...");
                sock = sersock.accept();

                CommunicationThread rt;
                if(server instanceof CurrencyExchangeServer)
                {
                    rt = new CESCommunicationThread((CurrencyExchangeServer) server,sock,
key);
                }
                else if(server instanceof BlackListServer)
                {
                     rt = new BLSCommunicationThread((BlackListServer) server,sock, key);
                }
                else if(server instanceof VerificationServer)
                {
                     rt = new VSCommunicationThread((VerificationServer) server,sock,
key);
                }
                else //if(server instanceof RegistrationServer)
                {
                     rt = new RSCommunicationThread((RegistrationServer) server,sock,
key);
                }

                Debugger.debug(getClass(),3,"Connection established");
                rt.start();
            }
            catch(Exception e)
            {
                Debugger.debug(getClass(),3,"Error creating CommunicationThread ",e);
            }
        }

    }


}
```

10.4.4 BLS

10.4.4.a Blacklist

```
package SpamCash.Servers.BLS;


import java.util.Vector;
import java.util.Enumeration;
import java.io.Serializable;
import java.security.cert.Certificate;
import java.security.cert.X509Certificate;
import SpamCash.Utilities.Config;
import java.util.Date;
import SpamCash.Utilities.Debugger;

public class Blacklist implements Serializable
```

```
{
    //Performance variables
    private final int initialVectorSize = 10;
    private final int incrementVectorSize = 10;

    //When was the list updated

    private long lastUpdate;

    public Vector elements;

    public Blacklist(Vector list)
    {
        elements = list;
        updatedList();
    }

    public Blacklist()
    {
        elements = new Vector(initialVectorSize,incrementVectorSize);
    }

    public Blacklist(int initialSize)
    {
        elements = new Vector(initialSize,incrementVectorSize);
    }

    public void merge(Blacklist b)
    {
        Enumeration e = b.elements.elements();
        while(e.hasMoreElements())
        {
            add((Certificate)e.nextElement());
        }
        updatedList();
    }

    public void add(Certificate cert)
    {
        if(!onBlacklist(cert))
            elements.addElement(cert);
    }

    public void remove(Certificate cert)
    {
        elements.removeElement(cert);
    }

    public boolean onBlacklist(Certificate cert)
    {
        return elements.contains(cert);
    }

    public Blacklist getFromIndex(int index)
    {
        //Blacklist copy = new Blacklist(elements.size()-index);
        //Enumeration e = elements.elements();
        //while(e.hasMoreElements())
        //{
        //    copy.add((Certificate)e.nextElement());
        //}
        //return new Blacklist((Vector)elements.subList(index,elements.size()));

        return this;

    }

    private void updatedList()
    {
        lastUpdate = System.currentTimeMillis();
    }
```

```
    public boolean needUpdate()
    {
        long timeUntilUpdate = lastUpdate + Config.blacklistUpdateInterval -
System.currentTimeMillis();
        if(timeUntilUpdate > 0)
        {
            Date date = new Date(lastUpdate + Config.blacklistUpdateInterval);
            Debugger.debug(getClass(),1,"No need to update the blacklist until
"+date+".");
            return false;
        }
        return true;
    }

    public String toString()
    {
        String s = "Blacklist:\n   ";
        Enumeration e = elements.elements();
        while(e.hasMoreElements())
        {
            s += ((X509Certificate) e.nextElement()).getSubjectDN().getName() + "\n   ";
        }
        return s;
    }
}
```

10.4.4.b BlacklistServer

```
package SpamCash.Servers.BLS;


import SpamCash.NetworkPackets.*;
import SpamCash.Utilities.Config;
import SpamCash.Currency.*;
import SpamCash.Utilities.Debugger;
import SpamCash.Servers.AbstractServer;
import java.security.cert.Certificate;
import SpamCash.Utilities.Cryptotools;
import java.io.*;
import SpamCash.Utilities.FileHandler;
import java.security.Key;
import java.security.interfaces.*;
import org.logi.crypto.sign.Fingerprint;
import java.util.GregorianCalendar;
import java.security.cert.X509Certificate;

public class BlackListServer extends AbstractServer
{
    private RSAPublicKey RSPublicKey, CESPublicKey;
    private Certificate certificate;
    private Certificate CESCert, RSCert;
    private static Blacklist blacklist;

    private double time,count=0;
    private boolean firsttime = true;

    public BlackListServer() throws Exception
    {
        super(Config.BLSCertificateName, Config.BLSPrivateKeyName, Config.BLSPort);



        //Load the blacklist
        try
        {
            blacklist = (Blacklist) FileHandler.readObject(new
File(Config.blacklistFilePath));
        }
```

```
        catch(Exception e)
        {
            //Debugger.debug(getClass(), 1, "Blacklist could not be loaded. Creating
new.");
            blacklist = new Blacklist();
        }

        try
        {
            CESCert = FileHandler.importCertificate(new File(Config.CESCertificateName));
            RSCert = FileHandler.importCertificate(new File(Config.RSCertificateName));

            RSPublicKey = (RSAPublicKey) RSCert.getPublicKey();
            CESPublicKey = (RSAPublicKey) CESCert.getPublicKey();
        }
        catch(Exception ex)
        {
            //Debugger.debug(getClass(), 1, "Initializationexception...", ex);
        }

    }

    public void printStatus()
    {
        //Debugger.debug(getClass(), 1, "Status:\n" + blacklist.toString());
    }

    public static void main(String[] args)
    {
        try
        {
            //Debugger.setApplicationName("BlacklistServer");
            new BlackListServer();
        }
        catch(FileNotFoundException ex)
        {
            //Debugger.debug(BlackListServer.class, 1, "Initializationexception...file
not found " + ex.getMessage(),  ex);
        }
        catch(Exception ex)
        {
            //Debugger.debug(BlackListServer.class, 1, "Initializationexception...", ex);
        }
    }

    public synchronized BlacklistPacket getBlacklist(GetBlacklistPacket getPacket)
    {
        count++;
        return new BlacklistPacket(blacklist.getFromIndex(getPacket.splitIndex));
    }

    public synchronized void reportCoins(ReportCoinsPacket pack) throws Exception
    {
        if(firsttime)
        {
            System.out.println("Starting measurements...");
            time = System.currentTimeMillis();
            count = 0;
            firsttime = false;
        }

        Coin a = pack.a;
        Coin b = pack.b;

        //Debugger.debug(getClass(), 1, "\n\nThe following coins have been reported. List
size :"+this.blacklist.elements.size());
        //Debugger.debug(getClass(), 1, a.toString());
        //Debugger.debug(getClass(), 1, b.toString());

        //Debugger.debug(getClass(), 1, "Testing if the coins have the same
serialnumber");
```

```
        if(a.getSerialNumber() != b.getSerialNumber())
        {
            //Debugger.debug(getClass(), 1, "They do NOT! Please stop wasting my time.");
            return;
        }

        //Debugger.debug(getClass(), 1, "SerialNumbers match, now testing if coins are
identical");

        if(a.equals(b))
        {
            //Debugger.debug(getClass(), 1, "Coins are identical. Someone probably tried
to deposit the same coin twice. No blacklisting necessary since no emails were sent with
copied coin");
            return;
        }

        //Debugger.debug(getClass(), 1, "Coins are NOT identical");

        //Debugger.debug(getClass(), 1, "Decrypting transactionlists...");

        a.decryptTransactionlist(this.privateKey);
        b.decryptTransactionlist(this.privateKey);

        //Debugger.debug(getClass(), 1, "Decrypted transactionlists succesfully.");

        Transactionlist listA = a.getTransactionlist();
        Transactionlist listB = b.getTransactionlist();

        //Debugger.debug(getClass(), 1, "Verifying the list-integrity signatures.....");

        if(!(verifyIntegrity(listA,a.getSerialNumber()) &&
verifyIntegrity(listB,b.getSerialNumber())))
        {
            //Debugger.debug(getClass(), 1, "Verifying the list-integrity signatures
failed.");
            return;
        }
        //Debugger.debug(getClass(), 1, "Verifying the list-integrity signatures
successful.");

        //Debugger.debug(getClass(), 1, "Searching the transactionLists.....");

        int i = 1;

        try
        {

            while(true)
            {
                //If the elements are different
                //We should blacklist someone
                if(!listA.elementAt(i).equals(listB.elementAt(i)))
                {
                    //Debugger.debug(getClass(), 1, "Lists differ at position " + i);

                    //Did the CESServer make two coins with same serialnumber?
                    if(i == 1)
                    {
                        //Debugger.debug(getClass(), 1,"CES made two coins with the same
serialnumber ! (That is VERY unlikely)");
                        //Debugger.debug(getClass(), 1, "Therefore noone should be
blacklistet.");
                        return;
                    }
                    else
                    {
                        //Add the previous certificate to the blacklist
                        addToBlacklist(listA.elementAt(i - 2).getCertificate(), i - 2);
                        return;
```

```
                    }
                }
                //If we have reached the end of a list
                //We should blacklist someone
                if(listA.length() - 1 == i || listB.length() - 1 == i)
                {
                    //Debugger.debug(getClass(), 1, "No more elements in one of the
lists.");
                    addToBlacklist(listA.elementAt(i).getCertificate(), i);
                    return;
                }
                i = i + 2;
            }
        }
    catch(Exception e)
    {
        //Debugger.debug(getClass(),1,"Error while blacklisting, noone is
blacklisted",e);
    }
    }

    private boolean verifyIntegrity(Transactionlist list,long serial)
    {
        try
        {
            return list.verify(RSCert, CESCert,serial);
        }
        catch(Exception ex)
        {
            //Debugger.debug(getClass(), 1, "SHA1-hash could not be calculated.");
            return false;
        }

    }

    private synchronized void addToBlacklist(Certificate cert, int position)
    {
        blacklist.add(cert);
        try
        {
            FileHandler.writeObject(blacklist, new File(Config.blacklistFilePath));
        }
        catch(Exception ex)
        {
            //Debugger.debug(getClass(), 1, "Blacklist could not be written to disk.");
        }
        //Debugger.debug(getClass(), 1, "Certificate at position " + position + " has
been blacklisted. Contents: "+((X509Certificate) cert).getSubjectDN().getName());
    }

    public void shutdown() throws Exception
    {
        System.out.println("Rate:" +count/((System.currentTimeMillis()-time)/1000)+" pr.
sek");
    }
}
```

10.4.4.c BLSCommunicationThread

```
package SpamCash.Servers.BLS;


import java.net.Socket;
import java.security.Key;
import SpamCash.Utilities.Debugger;
import SpamCash.NetworkPackets.*;
import SpamCash.Servers.CommunicationThread;

public class BLSCommunicationThread extends CommunicationThread
```

```
{
    private BlackListServer BLS;
    private NetPacket np;

    public BLSCommunicationThread(BlackListServer server, Socket sock, Key key)
    {
        super(sock, key, false);
        this.BLS = server;
    }

    public void run()
    {
        try
        {
            Debugger.debug(getClass(), 3, "Run method initiated");
            np = clientSession.receive();
            //Debugger.debug(getClass(), 1, "Received packet: " + np + "\n From client "
+ clientSession);

            if(np instanceof GetBlacklistPacket)
            {
                clientSession.send((NetPacket) BLS.getBlacklist((GetBlacklistPacket)
np));
            }
            else if(np instanceof ReportCoinsPacket)
            {
                BLS.reportCoins((ReportCoinsPacket) np);
            }
            else
            {
                Debugger.debug(getClass(), 1, "Unknown Packet-type received: " + np);
                throw new Exception("Uknown Packet-type received: " + np);
            }
            BLS.printStatus();
        }
        catch(Exception ex2)
        {
            Debugger.debug(getClass(), 1, "Closing connection.", ex2);
            try
            {
                clientSession.close();
            }
            catch(Exception ex1)
            {}
            return;
        }
    }
}
```

10.4.5 CES

10.4.5.a Accounts

```
package SpamCash.Servers.CES;


import java.util.Hashtable;
import java.util.Enumeration;

public class Accounts
{
    private Hashtable accounts = new Hashtable();

    protected Accounts(BankAccount acc)
    {
        accounts.put(acc.username, acc);
```

```
}

protected void addAccount(BankAccount acc)
{
    accounts.put(acc.username, acc);
}

protected boolean removeAccount(BankAccount acc)
{
    return(accounts.remove(acc.username) != null);
}

protected double totalbalance()
{
    double total = 0;
    int i = 0;
    for(Enumeration e = accounts.keys(); e.hasMoreElements(); )
    {
        total += ((BankAccount) e.nextElement()).balance;
    }

    return total;
}

protected BankAccount getAccount(String username)
{
    return(BankAccount) accounts.get(username);
}

public String toString()
{
    String s = "BankAccounts:\n";
    for(Enumeration e = accounts.elements(); e.hasMoreElements(); )
    {
        s += "    " + ((BankAccount) e.nextElement()).toString() + "\n";
    }
    return s;
}

public static void main(String[] s)
{
    try
    {
        //Partial Functional test
        Accounts accounts = new Accounts(new BankAccount("Bent", "Bent01", 100));

        System.out.println(accounts.toString());

        BankAccount acc = accounts.getAccount("Bent");

        acc.withdraw(9.0);
        System.out.println("Withrawed 9.0 from Bents account");
        System.out.println(accounts.toString());

        acc.deposit(20.0);
        System.out.println("Deposit 20.0 to Bents account");
        System.out.println(accounts.toString());

        acc.withdrawNoteUpFront(5.0);
        System.out.println("withdrawNoteUpFront 5.0  from Bents account");
        System.out.println(accounts.toString());

        acc.withdrawFromUpFront(5.0);
        System.out.println("withdrawFromUpFront 5.0 from Bents account");
        System.out.println(accounts.toString());

        acc.withdrawNoteUpFront(12.0);
        System.out.println("withdrawNoteUpFront 12.0  from Bents account");
        System.out.println(accounts.toString());

        acc.withdrawFromUpFront(20.0);
```

```
            System.out.println("withdrawFromUpFront 20.0 from Bents account");
            System.out.println(accounts.toString());

            /*

                    PASTE IN THE CES !!!!!!!!!!!!!!!!!!!!!!!!!!

            try
            {
                acc.withdrawFromUpFront(amount);
            }
            catch(Exception ex)
            {
                String error = "The client could pay for the coins. Terminating! (..not
the client...just the connection). "+ex.getMessage();
                Debugger.debug(getClass(), 1,error);
                throw new Exception(error);
            }
*/
        }
        catch(Exception ex)
        {
        }

    }
}
```

10.4.5.b BankAccount

```
package SpamCash.Servers.CES;

import java.security.cert.Certificate;
import java.util.Vector;

public class BankAccount
{
    public String username;
    public String password;
    public double balance;
    public double paidUpFront;
    public Certificate cert;
    public Vector activationNumbers;

    public BankAccount(String username, String password, double balance)
    {
        this.username = username;
        this.password = password;
        this.balance = balance;
        activationNumbers = new Vector();
    }

    public void deposit(double amount) throws Exception
    {
        if(amount >= 0)
            balance += amount;
        else
            throw new Exception("BankAccount: Cannot deposit a negative amount!
("+amount+")");
    }


    public void withdraw(double amt) throws Exception
    {
        if(amt <= 0 || balance < amt)
            throw new Exception("Cannot withdraw "+amt+" since there is only "+balance+"
available.");
        balance -= amt;
    }
```

```
    public void withdrawNoteUpFront(double amt) throws Exception
    {
        if(amt <= 0 || balance < amt)
            throw new Exception("Cannot withdraw "+amt+" since there is only "+balance+"
available.");
        balance -= amt;
        paidUpFront += amt;
    }

    public void withdrawFromUpFront(double amt) throws Exception
    {
        if(amt <= 0 || balance + paidUpFront < amt)
            throw new Exception("Cannot withdraw "+amt+" since there is only
"+(balance+paidUpFront)+" available.");
        balance -= withdrawWithUpFront(amt);
    }

    /**
     * Withdraw amt from the account if possible.
     * If not the remaining amount that needs to be withdrawed is returned.
     * @param amt double Amount to be withdrawn
     * @return double Amount that was not withdrawed
     */
    private double withdrawWithUpFront(double amt)
    {
        if(amt <= paidUpFront)
        {
            paidUpFront -= amt;
            return 0.0;
        }
        else
        {
            double notWithdrawed = amt - paidUpFront;
            paidUpFront = 0.0;
            return notWithdrawed;
        }
    }

    public boolean equals(BankAccount acc)
    {
        return(this.username.equals(acc.username) && this.password.equals(acc.password));
    }

    public String toString()
    {
        return "BankAccount: name = "+username+" \tpass = "+password+" \tbalance =
"+balance+"\t("+paidUpFront+" paid upfront)";
    }
}
```

10.4.5.c CESCommunicationThread

```
package SpamCash.Servers.CES;


import java.net.Socket;
import java.security.Key;
import SpamCash.Utilities.Debugger;
import SpamCash.NetworkPackets.*;
import SpamCash.Servers.CommunicationThread;
import SpamCash.Currency.Coin;
import java.util.Random;

public class CESCommunicationThread extends CommunicationThread
{
    private NetPacket np;
    protected CurrencyExchangeServer CES;
    private String username;
```

```
    public CESCommunicationThread(CurrencyExchangeServer server, Socket sock, Key key)
    {
        super(sock, key, true);
        this.CES = server;
    }

    public void run()
    {
        Debugger.debug(getClass(), 3, "New Thread started...");
        while(true)
        {
            try
            {
                np = clientSession.receive();
                //Debugger.debug(getClass(), 1, "Received packet: " + np + "\n From
client " + clientSession);

                if(np instanceof LogonPacket)
                {
                    StatusPacket lrp = CES.logon((LogonPacket) np);
                    loggedIn = lrp.ok;
                    if(loggedIn)
                        username = ((LogonPacket) np).username;

                    clientSession.send(lrp);
                }
                else if(loggedIn && np instanceof DepositRequestPacket)
                {
                    clientSession.send(CES.depositRequest(username,
(DepositRequestPacket) np));
                }
                else if(loggedIn && np instanceof CurrencyExchangeRequestPacket)
                {
                    clientSession.send(CES.exchangeRequest(username,
(CurrencyExchangeRequestPacket) np));
                }
                else if(loggedIn && np instanceof CurrencyBlindingFactorPacket)
                {
                    clientSession.send(CES.blindingFactor(username,
(CurrencyBlindingFactorPacket) np));
                }
                else if(loggedIn && np instanceof RegistrationInfoPacket)
                {
                    try
                    {
                        clientSession.send(CES.activationRequest(username,
(RegistrationInfoPacket) np));
                    }
                    catch(Exception e)
                    {
                        Debugger.debug(getClass(),3,"Error: "+e.getMessage(),e);
                        clientSession.send(new StatusPacket(false,"The registration
information could not be sent to the registration server."));
                    }
                }
                else
                {
                    Debugger.debug(getClass(), 1, "Unknown Packet-type received: " + np);
                    Debugger.debug(getClass(), 1, "Closing connection.");
                    clientSession.close();
                }
                CES.printStatus();
            }
            catch(Exception ex)
            {
                Debugger.debug(getClass(), 1, "Exception: ",ex);
                try
                {
                    clientSession.close();
                }
                catch(Exception ex1)
```

```
                {
                }
                return;
            }

        }
    }
}
```

10.4.5.d CESConfiguration

```
package SpamCash.Servers.CES;


import java.util.Vector;

public class CESConfiguration implements java.io.Serializable
{
    Vector usedCoinSerials, usedCoins, coinSets;
    Accounts accounts;

    public CESConfiguration(Vector usedCoinSerials, Vector usedCoins, Vector coinSets,
Accounts accounts)
    {
        this.usedCoinSerials = usedCoinSerials;
        this.usedCoins = usedCoins;
        this.coinSets = coinSets;
        this.accounts = accounts;
    }

}
```

10.4.5.e CoinSet

```
package SpamCash.Servers.CES;


import SpamCash.Currency.*;
import org.logi.crypto.sign.BlindFingerprint;

public class CoinSet
{
    public String username;
    public BlindedCoin[][] blindedCoins;
    public int[] blindingFactorIndexes;

    public CoinSet(String username, BlindedCoin[][] blindedCoins,int[]
blindingFactorIndexes)
    {
        this.username = username;
        this.blindedCoins = blindedCoins;
        this.blindingFactorIndexes = blindingFactorIndexes;
    }
}
```

10.4.5.f CurrencyExchangeServer

```
package SpamCash.Servers.CES;


import SpamCash.Utilities.*;
import SpamCash.NetworkPackets.*;
import SpamCash.Currency.*;
import SpamCash.Servers.*;
import SpamCash.Utilities.Cryptotools;
```

```
import SpamCash.Utilities.Debugger;
import SpamCash.Servers.AbstractServer;

import java.util.Vector;
import java.security.cert.*;
import java.util.Random;
import java.net.Socket;
import java.net.InetAddress;
import java.io.*;
import java.security.interfaces.*;
import org.logi.crypto.keys.RSABlindingFactor;

public class CurrencyExchangeServer extends AbstractServer
{
    private RSAPublicKey RSPublicKey, BLSPublicKey;
    private Certificate certificate, BLSCert;

    private Vector usedCoinSerials, usedCoins;
    private static Accounts accounts;

    private File file;
    private CESConfiguration config;

    private Vector coinSets;

    public CurrencyExchangeServer() throws Exception
    {
        super(Config.CESCertificateName, Config.CESPrivateKeyName, Config.CESPort);
        org.logi.crypto.Crypto.initRandom();
        file = new File(Config.CESConfigFilePath);

        if(!file.exists())
        {
            Debugger.debug(getClass(), 1, "Creating server data.");
            usedCoinSerials = new Vector();
            usedCoins = new Vector();
            coinSets = new Vector();
            accounts = new Accounts(new BankAccount("Bent", "Bent01", 100));
            accounts.addAccount(new BankAccount("Santa", "Claus", 40000));
            accounts.addAccount(new BankAccount("user1", "user1", 40000));
            accounts.addAccount(new BankAccount("user2", "user2", 40000));
            accounts.addAccount(new BankAccount("Anders", "And", 3));
            accounts.addAccount(new BankAccount("test", "test", 12000));
        }
        else
        {
            Debugger.debug(getClass(), 1, "Reading server data from file.");
            config = (CESConfiguration) FileHandler.readObject(file);
            usedCoinSerials = config.usedCoinSerials;
            usedCoins = config.usedCoins;
            coinSets = config.coinSets;
            accounts = config.accounts;
        }

        RSPublicKey = (RSAPublicKey) FileHandler.importCertificate(new
File(Config.RSCertificateName)).getPublicKey();
        BLSCert = FileHandler.importCertificate(new File(Config.BLSCertificateName));
        BLSPublicKey = (RSAPublicKey) BLSCert.getPublicKey();

        certificate = FileHandler.importCertificate(new File(Config.CESCertificateName));
    }

    protected void printStatus()
    {
        Debugger.debug(getClass(), 1, "Status:\n" + accounts.toString());
    }

    public static void main(String[] args)
    {
        try
        {
```

```
            Debugger.setApplicationName("CurrencyExchangeServer");
            new CurrencyExchangeServer();
        }
        catch(FileNotFoundException ex)
        {
            Debugger.debug(CurrencyExchangeServer.class, 1,
"Initializationexception...file not found " + ex.getMessage(),
                           ex);

        }
        catch(Exception ex)
        {
            Debugger.debug(CurrencyExchangeServer.class, 1, "Initializationexception...",
ex);
        }
    }

    public synchronized StatusPacket logon(LogonPacket lp)
    {
        Debugger.debug(getClass(), 1, "Logon...username: " + lp.username + "  password:"
+ lp.password);
        BankAccount acc = (BankAccount) accounts.getAccount(lp.username);
        if(acc == null || !acc.password.equals(lp.password))
            return new StatusPacket(false,
                                    "The username and/or password supplied for the
Currency Exchange Server was incorrect. Please try again.");
        return new StatusPacket(true, "Successful login");

    }

    public synchronized AmountDepositedPacket depositRequest(String username,
DepositRequestPacket pack) throws
        Exception
    {
        Debugger.debug(getClass(), 1, "Deposit...User: " + username);

        BankAccount acc = accounts.getAccount(username);

        double amount = 0.0;
        Long sn;
        Random rand = new Random();
        long nonce;
        session VSSession = createVSSession();
        StatusPacket statusPack;

        for(int i = 0; i < pack.coins.length; i++)
        {
            sn = new Long(pack.coins[i].getSerialNumber());
            if(!usedCoinSerials.contains(sn))
            {
                //Verify coins using the Verificationserver
                Debugger.debug(getClass(), 1, "Coin with a valid serial received. Sending
to VS and verifying...");
                nonce = rand.nextLong();
                VSSession.send(new VerificationPacket(pack.coins[i], acc.cert, nonce));
                do
                {
                    statusPack = (StatusPacket) VSSession.receive();
                }
                while(nonce != statusPack.nonce);
                if(statusPack.ok)
                {
                    Debugger.debug(getClass(), 1, "VS verifyed successfully...");
                    Debugger.debug(getClass(), 1, "Verifying CES-signature...");
                    pack.coins[i].verifyCES(certificate);
                    amount += pack.coins[i].getValue();
                    Debugger.debug(getClass(), 1,
                                   "Verification of CES-signature succesful. Current
amount validated: " + amount);
                    usedCoins.add(pack.coins[i]);
                    usedCoinSerials.add(sn);
```

```
                }
            }
            else
                reportCoins(pack.coins[i], (Coin)
usedCoins.elementAt(usedCoinSerials.indexOf(sn)));
        }

        Debugger.debug(getClass(), 1,
                       "Coin verification complete. Depositing " + amount + " to account
with username: "
                       + acc.username);
        acc.deposit(amount);
        Debugger.debug(getClass(), 1, "DepositRequest completed.");
        return new AmountDepositedPacket(amount);
    }

    private session createBLSSession() throws Exception
    {
        Debugger.debug(getClass(), 3, "Creating new session with the BLS: " +
Config.BLSAddress + ":" + Config.BLSPort);
        Socket socket = new Socket(InetAddress.getByName(Config.BLSAddress),
Config.BLSPort);
        return new session(socket, false, BLSPublicKey, false);
    }

    private session createVSSession() throws Exception
    {
        Debugger.debug(getClass(), 3, "Creating new session with the VS: " +
Config.VSAddress + ":" + Config.VSPort);
        Socket socket = new Socket(InetAddress.getByName(Config.VSAddress),
Config.VSPort);
        return new session(socket, false, BLSPublicKey, true);
    }

    public synchronized void reportCoins(Coin a, Coin b)
    {
        Debugger.debug(getClass(), 1, "Reporting coins....");
        try
        {
            session serverSession = createBLSSession();
            serverSession.send(new ReportCoinsPacket(a, b));
            Debugger.debug(getClass(), 1, "Reported coins succesfully.");
        }
        catch(Exception ex1)
        {
            Debugger.debug(getClass(), 1, "Error reporting coin.");
        }
    }

    public synchronized RequestBlindingFactorsPacket exchangeRequest(String username,
CurrencyExchangeRequestPacket cp)
    {
        Debugger.debug(getClass(), 1, "Exchange request (withdraw)...User: " + username);
        BlindedCoin[][] coins = ((CurrencyExchangeRequestPacket) cp).getCoins();
        int[] index = new int[coins.length];
        Random x = new Random();
        Debugger.debug(getClass(), 1, "Calculating random index in each coinset:");
        String indexes = "";
        for(int i = 0; i < index.length; i++)
        {
            index[i] = x.nextInt(coins[i].length); //nextInt(n) returns random number in
the interval [0;n-1]
            indexes += " " + index[i];
        }
        Debugger.debug(getClass(), 1, "Indexes chosen: [" + indexes + " ]");

        //Save the information to check later.
        coinSets.add(new CoinSet(username, coins, index));

        Debugger.debug(getClass(), 1, "ExchangeRequest completed for user: " + username);
        return new RequestBlindingFactorsPacket(index);
```

```
    }

    public synchronized StatusPacket activationRequest(String username,
RegistrationInfoPacket pack) throws
        Exception
    {
        Debugger.debug(getClass(), 1, "Activationrequest... User: " + username);

        BankAccount acc = accounts.getAccount(username);

        Debugger.debug(getClass(), 1, "Found account.");
        Debugger.debug(getClass(), 1, "Testing if activationNumber has already been
used");
        if(acc.activationNumbers.contains(new Long(pack.activationNumber)))
            return new StatusPacket(true, "activationNumber accepted.");

        acc.activationNumbers.addElement(new Long(pack.activationNumber));
        Debugger.debug(getClass(), 1, "activationNumber has NOT been used before.");
        session RSsession = createRSSession();
        RSsession.send(new ActivationPacket(pack.activationNumber,
                                            Cryptotools.sign(super.privateKey, new
Long(pack.activationNumber))));

        // CurrencyServer now tries to retrieve the users certificate from the
registration server

        while(true)
        {
            RSsession.send(new RegistrationInfoPacket(pack.activationNumber));
            NetPacket packet = RSsession.receive();

            if(packet instanceof ActivationCompletedPacket)
            {
                acc.cert = ((ActivationCompletedPacket) packet).certificate;
                Debugger.debug(getClass(), 1, "Certificate succesfully retrieved from
RS.");
                break;
            }
            else if(packet instanceof StatusPacket)
            {
                Debugger.debug(getClass(), 1, "Error retrieving certificate from RS.");
                return new StatusPacket(false, "Error generating certificate.");
            }
            else
                throw new Exception("Error in getting certificate from RS. Server
responded unexpectedly.");

        }

        Debugger.debug(getClass(), 1,
                       "Withdrawing the registrationFee " + Config.registrationFee + " +
" + Config.registrationCoinValue
                       + ".");
        acc.withdrawNoteUpFront(Config.registrationFee + Config.registrationCoinValue);
        Debugger.debug(getClass(), 1, "ActivationRequest completed for user: " +
username);

        return new StatusPacket(true, Config.registrationCoinValue);
    }

    private session createRSSession() throws Exception
    {
        Debugger.debug(getClass(), 3, "Creating new session with the RS: " +
Config.RSAddress + ":" + Config.RSPort);
        Socket socket = new Socket(InetAddress.getByName(Config.RSAddress),
Config.RSPort);
        return new session(socket, false, RSPublicKey, true);
    }

    public synchronized CoinSignaturePacket blindingFactor(String username,
CurrencyBlindingFactorPacket cp) throws
```

```
        Exception
    {
        Debugger.debug(getClass(), 1, "BlindFactors received... User: " + username);
        CoinSet coinset = null;
        boolean found = false;
        int i = 0;
        while(i < coinSets.size())
        {
            coinset = ((CoinSet) coinSets.elementAt(i));
            if(coinset.username.equals(username))
            {
                coinSets.remove(i);
                found = true;
                break;
            }
            i++;
        }
        //Not found?
        if(!found)
        {
            String error =
                "Username unknown. Probably a BlindingFactorPacket was sent before the
corresponding CurrencyExchangePacket.";
            Debugger.debug(getClass(), 1, error);
            throw new Exception(error);
        }

        //Validate certificate
        try
        {
            Debugger.debug(getClass(), 1, "Verifying the certificate...");
            cp.certificate.verify(RSPublicKey);
        }
        catch(Exception e)
        {
            String error = "The certificate supplied is invalid. Verification of the RS
signature failed.";
            Debugger.debug(getClass(), 1, error);
            throw new Exception(error);
        }
        Debugger.debug(getClass(), 1, "Certificate succesfully verified.");

        //Calculate the amount of the coins on the way
        boolean amountAddedForSet = false;
        double amount = 0.0;

        Debugger.debug(getClass(), 1, "Unblinding and verifying the coins....");
        //Check that all coins are the same (except the blinded one in each set).
        for(int c = 0; c < coinset.blindedCoins.length; c++)
        {
            amountAddedForSet = false;
            //This loop goes through the coins in each set
            //Since the sets MUST be the same size, the loop runs to coins[0].lenght and
not coins[c].length
            for(int r = 0; r < coinset.blindedCoins[0].length; r++)
            {
                //Do we have a blindingfactor?
                if(r != coinset.blindingFactorIndexes[c])
                {
                    Debugger.debug(getClass(), 1,
                                   "Verifying coin " + (1 + c *
coinset.blindedCoins[0].length + r) + " of "
                                   + (coinset.blindedCoins.length *
coinset.blindedCoins[0].length) + "...");
                    //Check the contents of the coin
                    compareCoins(coinset.blindedCoins[c][r], cp.getRSABlindingFactor(c,
r), cp.coins[c][r],
                                 cp.certificate);

                    //Calculate the amount of all the coins
                    if(!amountAddedForSet)
```

```
                    {
                        amount += cp.coins[c][r].getValue();
                        amountAddedForSet = true;
                    }

                }
                else
                {
                    Debugger.debug(getClass(), 1,
                                   "No blindingFactor received for coin " + (1 + c *
coinset.blindedCoins[0].length + r)
                                   + " of " + (coinset.blindedCoins.length *
coinset.blindedCoins[0].length)
                                   + " as expected.");
                }
            }
        }

        //Lets withdraw the money. Did the client pay the price earlier?

        BankAccount acc = accounts.getAccount(username);

        // This condition is just for testing purposes
        // If there only one coin in each set
        // They are all blinded, and we dont know the amount to withdraw.
        // Of course this is not the case when not testing.
        if(Config.numberOfCoinsPerCoinset != 1)
        {
            try
            {
                acc.withdrawFromUpFront(amount);
            }
            catch(Exception ex)
            {
                String error =
                    "The client could not pay for the coins. Terminating! (..not the
client...just the connection). "
                    + ex.getMessage();
                Debugger.debug(getClass(), 1, error);
                throw new Exception(error);
            }
        }
        Debugger.debug(getClass(), 1, "Withdrawed " + amount + " from " + username + "'s
account.");

        Debugger.debug(getClass(), 1, "Everything seems to be ok.");
        Debugger.debug(getClass(), 1, "Lets sign the blinded coins and return them to the
client.");
        BlindedCoin[] signedCoins = new BlindedCoin[coinset.blindedCoins.length];
        for(int c = 0; c < coinset.blindedCoins.length; c++)
        {

coinset.blindedCoins[c][coinset.blindingFactorIndexes[c]].blindSign(super.privateKey);
            signedCoins[c] = coinset.blindedCoins[c][coinset.blindingFactorIndexes[c]];
        }

        Debugger.debug(getClass(), 1, "Checked and signed the coins blindly. Returning
signed coins.");
        return new CoinSignaturePacket(signedCoins);
    }

    private boolean compareCoins(BlindedCoin blindedCoin, RSABlindingFactor bf, Coin
coin, Certificate cert) throws
        Exception
    {
        /*Debugger.debug(getClass(), 2,
         "Comparing coin with blindingFactor=" + bf + " certificate=" +
cert.toString().substring(0, 30));
                Debugger.debug(getClass(), 2, "Blinded coin: " + blindedCoin);
                Debugger.debug(getClass(), 2, "Coin: " + coin);
         */
```

```
        try
        {
            coin.insertValues(Config.coinValue, Config.maxNumberOfUsages);
            if(!coin.signAndVerify(super.publicKey, super.privateKey, cert, blindedCoin,
bf))
                throw new Exception(
                    "The unblinded coins are not the same. Maby someone is trying to
cheat the currencyServer.");

            return true;
        }
        catch(Exception e)
        {
            throw new Exception(
                "The unblinded coins could not be compared. Maby someone is trying to
cheat the currencyServer.");
        }
    }

    public void shutdown() throws Exception
    {
        config.accounts = accounts;
        config.coinSets = coinSets;
        config.usedCoins = usedCoins;
        config.usedCoinSerials = usedCoinSerials;

        FileHandler.writeObject(config, file);
        Debugger.debug(getClass(), 1, "Saved configuration successfully");

    }


}
```

10.4.6 RS

10.4.6.a PendingAccount

```
package SpamCash.Servers.RS;


public class PendingAccount
{
    public long activationNumber;
    public byte[] certificateSigningRequest;

    public PendingAccount(long activationNumber,byte[] certificateSigningRequest)
    {
        this.activationNumber = activationNumber;
        this.certificateSigningRequest = certificateSigningRequest;
    }

    public String toString()
    {
      return "PendingAccount:\nActivationNumber:
"+activationNumber+"\ncertificateSigningRequest: "+certificateSigningRequest+"\n";
    }
}
```

10.4.6.b PendingAccounts

```
package SpamCash.Servers.RS;


import java.util.Vector;

public class PendingAccounts
{
```

```
    Vector accounts;

    public PendingAccounts()
    {
        accounts = new Vector();
    }

    public void add(PendingAccount a)
    {
        accounts.add(a);
    }

    public PendingAccount get(long activationNumber) throws Exception
    {
        PendingAccount acc;
        for(int i=0;i<accounts.size();i++)
        {
            acc = (PendingAccount)accounts.elementAt(i);
            if(acc.activationNumber == activationNumber)
            {
               return (PendingAccount) accounts.remove(i);
            }
        }
        throw new Exception("Could not find PendingAccount with activationNumber =
"+activationNumber);
    }


    public void remove(PendingAccount p)
    {
       accounts.remove(p);
       }

    public String toString()
    {
      String s="PendingAccounts:\n";
      for (int i=0;this.accounts.size()<i;i++)
      {
        s += "Element "+(i+1)+": "+((PendingAccount)
accounts.elementAt(i)).toString()+"\n";
      }
      return s;
    }


}
```

10.4.6.c RegistrationServer

```
package SpamCash.Servers.RS;


import SpamCash.NetworkPackets.*;
import SpamCash.Utilities.Config;
import SpamCash.Utilities.Debugger;
import SpamCash.Servers.AbstractServer;
import SpamCash.Utilities.FileHandler;
import SpamCash.Utilities.Cryptotools;

import java.io.FileInputStream;
import java.security.cert.CertificateFactory;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.io.FileNotFoundException;
import java.util.Random;
import java.security.interfaces.*;
import java.io.*;

public class RegistrationServer extends AbstractServer
{
    private RSAPublicKey CESPublicKey;
```

```
    public PendingAccounts pendingAccounts;

    public RegistrationServer() throws Exception
    {
        super(Config.RSCertificateName, Config.RSPrivateKeyName, Config.RSPort);

        pendingAccounts = new PendingAccounts();

        CESPublicKey = (RSAPublicKey) FileHandler.importCertificate(new
File(Config.CESCertificateName)).getPublicKey();

    }


    public synchronized void printStatus()
    {
        Debugger.debug(getClass(), 1, "Status:\n" + pendingAccounts.toString());
    }

    public static void main(String[] args)
    {
        try
        {
            Debugger.setApplicationName("RegistrationServer");
            new RegistrationServer();
        }
        catch(FileNotFoundException ex)
        {
            Debugger.debug(RegistrationServer.class, 1, "Initializationexception...file
not found ",
                           ex);
        }
        catch(Exception ex)
        {
            Debugger.debug(RegistrationServer.class, 1, "Initializationexception...",
ex);
        }
    }

    public synchronized RegistrationInfoPacket
registrationRequest(RegistrationRequestPacket pack)
    {
        Debugger.debug(getClass(), 1, "RegistrationRequest-method called");
        Random rand = new Random();
        Debugger.debug(getClass(), 1,
            "Generating ActivationNumber, and adding it to PendingAccounts together with
the certificateSigningRequest");
        long activationNumber = rand.nextLong();

        pendingAccounts.add(new PendingAccount(activationNumber,
pack.certificateSigningRequest));

        return new RegistrationInfoPacket(activationNumber + 1);
    }

    public synchronized void activate(ActivationPacket pack)
    {
        try
        {
            Debugger.debug(getClass(), 1, "Verifying CES-signature of activationNumber: "
+ pack.activationNumber);
            //Verify the currency servers signature...
            if(!Cryptotools.verify(CESPublicKey, pack.getSignature(),
                                   new Long(pack.activationNumber)))
            {
                Debugger.debug(getClass(), 1,
                               "Could not verify signature of activationNumber = " +
pack.activationNumber
                               + ". Number still not aproved.");
                return;
```

```
            }
            //pendingAccounts.aprove(pack.activationNumber);
            Debugger.debug(getClass(), 1, "Verifued signature of activationNumber = " +
pack.activationNumber + ".");
            Debugger.debug(getClass(), 1, "Generating certificate.");
            PendingAccount p = pendingAccounts.get(pack.activationNumber);
            createCertificate(pack.activationNumber, p.certificateSigningRequest);
            pendingAccounts.remove(p);
            Debugger.debug(getClass(), 1,
                           "Certificate for activationNumber: " + pack.activationNumber +
" Created successfully.");
            return;
        }
        catch(Exception e)
        {
            Debugger.debug(getClass(), 1, "Exception: ", e);
            Debugger.debug(getClass(), 1,
                           "Error generating certificate.");
            return;
        }
    }

    public synchronized NetPacket activationRequest(RegistrationInfoPacket pack)
    {
        try
        {
            Debugger.debug(getClass(), 1, "Attempting to retrieve certificate");
            File generatedCert = new File(Config.OpenSSLpath + pack.activationNumber +
".crt");

            Certificate certificate = FileHandler.importCertificate(generatedCert);
            Debugger.debug(getClass(), 1, "Certificate found, returning it in an
ActivationCompletedPacket");
            return new ActivationCompletedPacket(certificate);
        }
        catch(Exception ex)
        {
            Debugger.debug(getClass(), 1, "Exception: ", ex);
            return new StatusPacket(false, "Error retrieving certificate " +
ex.getMessage());
        }
    }

    public synchronized void createCertificate(long activationNumber, byte[]
certificateSigningRequest) throws
        Exception
    {
        //String filesep = System.getProperty("file.separator");
        Debugger.debug(getClass(), 1, "Creating cerfiticate from received
certificaterequest");
        Debugger.debug(getClass(), 1, "Writing certificaterequest to file");
        File certRequestFile = new File(Config.OpenSSLpath + "certificaterequest.csr");
        FileOutputStream fos = new FileOutputStream(certRequestFile);
        fos.write(certificateSigningRequest);
        fos.close();
        Runtime rt = Runtime.getRuntime();
        String command = "openssl ca -in " + Config.OpenSSLpath + "certificaterequest.csr
-out " + Config.OpenSSLpath
            + activationNumber + ".crt -notext -batch -key spammail";
        Debugger.debug(getClass(), 1, "Executing openssl command: " + command);
        Process process = rt.exec(command);
        File generatedCert = new File(Config.OpenSSLpath + activationNumber + ".crt");
        Debugger.debug(getClass(), 1, "Waiting for certificatefile to be created and
proceed when it is.");

        int checks = 0;
        while(!generatedCert.exists() || generatedCert.length() == 0 ||
!generatedCert.canRead())
        {
            if(checks >= 30)
            {
```

```
                InputStream in = process.getErrorStream();
                BufferedReader d = new BufferedReader(new InputStreamReader(in));

                String err = "Error generating certificate. ";
                while(d.ready())
                {
                    err += d.readLine() + "\n";
                }
                generatedCert.delete();
                certRequestFile.delete();
                Debugger.debug(getClass(), 1, err);
                throw new Exception(err);
            }

            Thread.sleep(300);
            Debugger.debug(Cryptotools.class, 3, "Waiting for certificate to be
generated");
            checks++;
        }

        Debugger.debug(getClass(), 3, "Certificate generated");
        Certificate cert = FileHandler.importCertificate(generatedCert);
        Debugger.debug(getClass(), 1, "Certificate imported successfully from file");
        //generatedCert.delete();
        certRequestFile.delete();
        return;
    }

    public void shutdown() throws Exception
    {

    }


}
```

10.4.6.d RSCommunicationThread

```
package SpamCash.Servers.RS;


import SpamCash.NetworkPackets.*;
import SpamCash.Utilities.Config;
import SpamCash.Utilities.Debugger;
import SpamCash.Servers.AbstractServer;
import SpamCash.Utilities.FileHandler;
import SpamCash.Utilities.Cryptotools;

import java.io.FileInputStream;
import java.security.cert.CertificateFactory;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.io.FileNotFoundException;
import java.util.Random;
import java.security.interfaces.*;
import java.io.*;

public class RegistrationServer extends AbstractServer
{
    private RSAPublicKey CESPublicKey;

    public PendingAccounts pendingAccounts;

    public RegistrationServer() throws Exception
    {
        super(Config.RSCertificateName, Config.RSPrivateKeyName, Config.RSPort);

        pendingAccounts = new PendingAccounts();
```

```
        CESPublicKey = (RSAPublicKey) FileHandler.importCertificate(new
File(Config.CESCertificateName)).getPublicKey();

    }


    public synchronized void printStatus()
    {
        Debugger.debug(getClass(), 1, "Status:\n" + pendingAccounts.toString());
    }

    public static void main(String[] args)
    {
        try
        {
            Debugger.setApplicationName("RegistrationServer");
            new RegistrationServer();
        }
        catch(FileNotFoundException ex)
        {
            Debugger.debug(RegistrationServer.class, 1, "Initializationexception...file
not found ",
                           ex);
        }
        catch(Exception ex)
        {
            Debugger.debug(RegistrationServer.class, 1, "Initializationexception...",
ex);
        }
    }

    public synchronized RegistrationInfoPacket
registrationRequest(RegistrationRequestPacket pack)
    {
        Debugger.debug(getClass(), 1, "RegistrationRequest-method called");
        Random rand = new Random();
        Debugger.debug(getClass(), 1,
            "Generating ActivationNumber, and adding it to PendingAccounts together with
the certificateSigningRequest");
        long activationNumber = rand.nextLong();

        pendingAccounts.add(new PendingAccount(activationNumber,
pack.certificateSigningRequest));

        return new RegistrationInfoPacket(activationNumber + 1);
    }

    public synchronized void activate(ActivationPacket pack)
    {
        try
        {
            Debugger.debug(getClass(), 1, "Verifying CES-signature of activationNumber: "
+ pack.activationNumber);
            //Verify the currency servers signature...
            if(!Cryptotools.verify(CESPublicKey, pack.getSignature(),
                                   new Long(pack.activationNumber)))
            {
                Debugger.debug(getClass(), 1,
                               "Could not verify signature of activationNumber = " +
pack.activationNumber
                               + ". Number still not aproved.");
                return;
            }
            //pendingAccounts.aprove(pack.activationNumber);
            Debugger.debug(getClass(), 1, "Verifued signature of activationNumber = " +
pack.activationNumber + ".");
            Debugger.debug(getClass(), 1, "Generating certificate.");
            PendingAccount p = pendingAccounts.get(pack.activationNumber);
            createCertificate(pack.activationNumber, p.certificateSigningRequest);
            pendingAccounts.remove(p);
            Debugger.debug(getClass(), 1,
```

```
                        "Certificate for activationNumber: " + pack.activationNumber +
" Created successfully.");
            return;
        }
        catch(Exception e)
        {
            Debugger.debug(getClass(), 1, "Exception: ", e);
            Debugger.debug(getClass(), 1,
                        "Error generating certificate.");
            return;
        }
    }

    public synchronized NetPacket activationRequest(RegistrationInfoPacket pack)
    {
        try
        {
            Debugger.debug(getClass(), 1, "Attempting to retrieve certificate");
            File generatedCert = new File(Config.OpenSSLpath + pack.activationNumber +
".crt");

            Certificate certificate = FileHandler.importCertificate(generatedCert);
            Debugger.debug(getClass(), 1, "Certificate found, returning it in an
ActivationCompletedPacket");
            return new ActivationCompletedPacket(certificate);
        }
        catch(Exception ex)
        {
            Debugger.debug(getClass(), 1, "Exception: ", ex);
            return new StatusPacket(false, "Error retrieving certificate " +
ex.getMessage());
        }
    }

    public synchronized void createCertificate(long activationNumber, byte[]
certificateSigningRequest) throws
        Exception
    {
        //String filesep = System.getProperty("file.separator");
        Debugger.debug(getClass(), 1, "Creating cerfiticate from received
certificaterequest");
        Debugger.debug(getClass(), 1, "Writing certificaterequest to file");
        File certRequestFile = new File(Config.OpenSSLpath + "certificaterequest.csr");
        FileOutputStream fos = new FileOutputStream(certRequestFile);
        fos.write(certificateSigningRequest);
        fos.close();
        Runtime rt = Runtime.getRuntime();
        String command = "openssl ca -in " + Config.OpenSSLpath + "certificaterequest.csr
-out " + Config.OpenSSLpath
            + activationNumber + ".crt -notext -batch -key spammail";
        Debugger.debug(getClass(), 1, "Executing openssl command: " + command);
        Process process = rt.exec(command);
        File generatedCert = new File(Config.OpenSSLpath + activationNumber + ".crt");
        Debugger.debug(getClass(), 1, "Waiting for certificatefile to be created and
proceed when it is.");

        int checks = 0;
        while(!generatedCert.exists() || generatedCert.length() == 0 ||
!generatedCert.canRead())
        {
            if(checks >= 30)
            {
                InputStream in = process.getErrorStream();
                BufferedReader d = new BufferedReader(new InputStreamReader(in));

                String err = "Error generating certificate. ";
                while(d.ready())
                {
                    err += d.readLine() + "\n";
                }
                generatedCert.delete();
```

```
                certRequestFile.delete();
                Debugger.debug(getClass(), 1, err);
                throw new Exception(err);
            }

            Thread.sleep(300);
            Debugger.debug(Cryptotools.class, 3, "Waiting for certificate to be
generated");
            checks++;
        }

        Debugger.debug(getClass(), 3, "Certificate generated");
        Certificate cert = FileHandler.importCertificate(generatedCert);
        Debugger.debug(getClass(), 1, "Certificate imported successfully from file");
        //generatedCert.delete();
        certRequestFile.delete();
        return;
    }

    public void shutdown() throws Exception
    {

    }


}
```

10.4.7 VS

10.4.7.a VerificationServer

```
package SpamCash.Servers.VS;


import SpamCash.NetworkPackets.*;
import SpamCash.Utilities.Config;
import SpamCash.Utilities.Debugger;
import java.io.FileInputStream;
import java.security.cert.CertificateFactory;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.io.FileNotFoundException;
import java.util.Random;
import SpamCash.Utilities.Cryptotools;
import java.security.interfaces.*;
import java.io.*;
import SpamCash.Utilities.FileHandler;
import SpamCash.Servers.AbstractServer;
import SpamCash.NetworkPackets.StatusPacket;
import SpamCash.Currency.Coin;
import SpamCash.Utilities.FileHandler;
import SpamCash.Currency.TransactionlistElement;
import SpamCash.Currency.UnencryptedTransactionlist;

public class VerificationServer extends AbstractServer
{
    private Certificate CESCertificate, RSCertificate;

    public VerificationServer() throws Exception
    {
        super(Config.BLSCertificateName, Config.BLSPrivateKeyName, Config.VSPort);

        CESCertificate = FileHandler.importCertificate(new
File(Config.CESCertificateName));
        RSCertificate = FileHandler.importCertificate(new
File(Config.RSCertificateName));
```

```
    }

    public static void main(String[] args)
    {
        try
        {
            Debugger.setApplicationName("VerificationServer");
            new VerificationServer();
        }
        catch(FileNotFoundException ex)
        {
            Debugger.debug(VerificationServer.class, 1, "Initializationexception...file
not found ",
                           ex);
        }
        catch(Exception ex)
        {
            Debugger.debug(VerificationServer.class, 1, "Initializationexception...",
ex);
        }
    }

    public synchronized StatusPacket VerificationRequest(VerificationPacket pack)
    {
        long time = System.currentTimeMillis();
        Debugger.debug(getClass(), 1, "Verifying coin(s)...");

        try
        {
            pack.coin.decryptTransactionlist(this.privateKey);

            //MaxUsages exceeded...?
            if(pack.coin.getMaxNumberOfUsages() <=
(pack.coin.getTransactionlist().length() - 2) / 2)
            {
                Debugger.debug(getClass(), 1,
                               "MaxNumberOfUsages exceeded. " +
pack.coin.getMaxNumberOfUsages() + " <= "
                               + (pack.coin.getTransactionlist().length() - 2) / 2);

                return new StatusPacket(false, "MaxNumberOfUsages exceeded.",
pack.nonce);
            }

            //Verify transactionlist
            if(!pack.coin.verify(RSCertificate, CESCertificate))
            {
                Debugger.debug(getClass(), 1, "Could not verify transactionlist.");
                return new StatusPacket(false, "Could not verify transactionlist.",
pack.nonce);
            }

            //Rightful owner?
            if(!pack.certificate.equals(((TransactionlistElement)
((UnencryptedTransactionlist) pack.coin.

transactionList).list.lastElement()).
                                        getCertificate()))
            {
                Debugger.debug(getClass(), 1, "Not rightful owner of coin.");
                return new StatusPacket(false, "Not rightful owner of coin.",
pack.nonce);
            }
            Debugger.debug(getClass(), 1, "Verified coin succesfully in
"+(System.currentTimeMillis()-time)+" milliseconds.");
            return new StatusPacket(true, "Everything OK.", pack.nonce);
        }

        catch(Exception e)
        {
            Debugger.debug(getClass(), 1, "An error occured during verification", e);
```

```
            return new StatusPacket(false, "An error occured during verification",
pack.nonce);
        }
    }

    public void shutdown() throws Exception
    {

    }


}
```

10.4.7.b VSCommunicationThread

```
package SpamCash.Servers.VS;

import java.net.Socket;
import java.security.Key;
import SpamCash.Utilities.Debugger;
import SpamCash.NetworkPackets.*;
import SpamCash.Servers.CommunicationThread;

public class VSCommunicationThread extends CommunicationThread
{
    private NetPacket np;
    protected VerificationServer VS;

    public VSCommunicationThread(VerificationServer server, Socket sock, Key key)
    {
        super(sock, key, true);
        this.VS = server;
    }

    public void run()
    {
        Debugger.debug(getClass(), 3, "New Thread started...");
        while(true)
        {

            try
            {
                np = clientSession.receive();
                //Debugger.debug(getClass(), 1, "Received packet " + np + "\n From client
" + clientSession);

                if(np instanceof VerificationPacket)
                {
                    clientSession.send(VS.VerificationRequest((VerificationPacket) np));
                }
                else
                {
                    Debugger.debug(getClass(), 1, "Unknown Packet-type received: " + np);
                    Debugger.debug(getClass(), 1, "Closing connection.");
                    try
                    {
                        Debugger.debug(getClass(), 1, "Closing connection.");
                        clientSession.close();
                    }
                    catch(Exception ex1)
                    {
                    }
                }
            }
            catch(Exception ex)
            {
                Debugger.debug(getClass(), 3, "Error ", ex);
                try
                {
```

```
                    Debugger.debug(getClass(), 1, "Closing connection.");
                    clientSession.close();
                }
                catch(Exception ex1)
                {
                }
                return;
            }
        }
    }
}
```

10.5 Utilities

10.5.1 Base64

```
/**
 * Encodes and decodes to and from Base64 notation.
 *
 * <p>
 * Change Log:
 * </p>
 * <ul>
 *  <li>v2.1 - Cleaned up javadoc comments and unused variables and methods. Added
 *   some convenience methods for reading and writing to and from files.</li>
 *  <li>v2.0.2 - Now specifies UTF-8 encoding in places where the code fails on systems
 *   with other encodings (like EBCDIC).</li>
 *  <li>v2.0.1 - Fixed an error when decoding a single byte, that is, when the
 *   encoded data was a single byte.</li>
 *  <li>v2.0 - I got rid of methods that used booleans to set options.
 *   Now everything is more consolidated and cleaner. The code now detects
 *   when data that's being decoded is gzip-compressed and will decompress it
 *   automatically. Generally things are cleaner. You'll probably have to
 *   change some method calls that you were making to support the new
 *   options format (<tt>int</tt>s that you "OR" together).</li>
 *  <li>v1.5.1 - Fixed bug when decompressing and decoding to a
 *   byte[] using <tt>decode( String s, boolean gzipCompressed )</tt>.
 *   Added the ability to "suspend" encoding in the Output Stream so
 *   you can turn on and off the encoding if you need to embed base64
 *   data in an otherwise "normal" stream (like an XML file).</li>
 *  <li>v1.5 - Output stream pases on flush() command but doesn't do anything itself.
 *      This helps when using GZIP streams.
 *      Added the ability to GZip-compress objects before encoding them.</li>
 *  <li>v1.4 - Added helper methods to read/write files.</li>
 *  <li>v1.3.6 - Fixed OutputStream.flush() so that 'position' is reset.</li>
 *  <li>v1.3.5 - Added flag to turn on and off line breaks. Fixed bug in input stream
 *      where last buffer being read, if not completely full, was not returned.</li>
 *  <li>v1.3.4 - Fixed when "improperly padded stream" error was thrown at the wrong
time.</li>
 *  <li>v1.3.3 - Fixed I/O streams which were totally messed up.</li>
 * </ul>
 *
 * <p>
 * I am placing this code in the Public Domain. Do with it as you will.
 * This software comes with no guarantees or warranties but with
 * plenty of well-wishing instead!
 * Please visit <a href="http://iharder.net/base64">http://iharder.net/base64</a>
 * periodically to check for updates or to contribute improvements.
 * </p>
 *
 * @author Robert Harder
 * @author rob@iharder.net
 * @version 2.1
 */

package SpamCash.Utilities;
```

```
public class Base64
{

/* ********  P U B L I C   F I E L D S  ******** */


    /** No options specified. Value is zero. */
    public final static int NO_OPTIONS = 0;

    /** Specify encoding. */
    public final static int ENCODE = 1;


    /** Specify decoding. */
    public final static int DECODE = 0;


    /** Specify that data should be gzip-compressed. */
    public final static int GZIP = 2;


    /** Don't break lines when encoding (violates strict Base64 specification) */
    public final static int DONT_BREAK_LINES = 8;


/* ********  P R I V A T E   F I E L D S  ******** */


    /** Maximum line length (76) of Base64 output. */
    private final static int MAX_LINE_LENGTH = 76;


    /** The equals sign (=) as a byte. */
    private final static byte EQUALS_SIGN = (byte)'=';


    /** The new line character (\n) as a byte. */
    private final static byte NEW_LINE = (byte)'\n';


    /** Preferred encoding. */
    private final static String PREFERRED_ENCODING = "UTF-8";


    /** The 64 valid Base64 values. */
    private final static byte[] ALPHABET;
    private final static byte[] _NATIVE_ALPHABET = /* May be something funny like EBCDIC
*/
    {
        (byte)'A', (byte)'B', (byte)'C', (byte)'D', (byte)'E', (byte)'F', (byte)'G',
        (byte)'H', (byte)'I', (byte)'J', (byte)'K', (byte)'L', (byte)'M', (byte)'N',
        (byte)'O', (byte)'P', (byte)'Q', (byte)'R', (byte)'S', (byte)'T', (byte)'U',
        (byte)'V', (byte)'W', (byte)'X', (byte)'Y', (byte)'Z',
        (byte)'a', (byte)'b', (byte)'c', (byte)'d', (byte)'e', (byte)'f', (byte)'g',
        (byte)'h', (byte)'i', (byte)'j', (byte)'k', (byte)'l', (byte)'m', (byte)'n',
        (byte)'o', (byte)'p', (byte)'q', (byte)'r', (byte)'s', (byte)'t', (byte)'u',
        (byte)'v', (byte)'w', (byte)'x', (byte)'y', (byte)'z',
        (byte)'0', (byte)'1', (byte)'2', (byte)'3', (byte)'4', (byte)'5',
        (byte)'6', (byte)'7', (byte)'8', (byte)'9', (byte)'+', (byte)'/'
    };

    /** Determine which ALPHABET to use. */
    static
    {
        byte[] __bytes;
        try
        {
            __bytes =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/".getBytes(
PREFERRED_ENCODING );
```

```
        }   // end try
        catch (java.io.UnsupportedEncodingException use)
        {
            __bytes = _NATIVE_ALPHABET; // Fall back to native encoding
        }   // end catch
        ALPHABET = __bytes;
    }   // end static


    /**
     * Translates a Base64 value to either its 6-bit reconstruction value
     * or a negative number indicating some other meaning.
     **/
    private final static byte[] DECODABET =
    {
        -9,-9,-9,-9,-9,-9,-9,-9,-9,                 // Decimal  0 -  8
        -5,-5,                                      // Whitespace: Tab and Linefeed
        -9,-9,                                      // Decimal 11 - 12
        -5,                                         // Whitespace: Carriage Return
        -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,     // Decimal 14 - 26
        -9,-9,-9,-9,-9,                             // Decimal 27 - 31
        -5,                                         // Whitespace: Space
        -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,              // Decimal 33 - 42
        62,                                         // Plus sign at decimal 43
        -9,-9,-9,                                   // Decimal 44 - 46
        63,                                         // Slash at decimal 47
        52,53,54,55,56,57,58,59,60,61,              // Numbers zero through nine
        -9,-9,-9,                                   // Decimal 58 - 60
        -1,                                         // Equals sign at decimal 61
        -9,-9,-9,                                      // Decimal 62 - 64
        0,1,2,3,4,5,6,7,8,9,10,11,12,13,            // Letters 'A' through 'N'
        14,15,16,17,18,19,20,21,22,23,24,25,        // Letters 'O' through 'Z'
        -9,-9,-9,-9,-9,-9,                          // Decimal 91 - 96
        26,27,28,29,30,31,32,33,34,35,36,37,38,     // Letters 'a' through 'm'
        39,40,41,42,43,44,45,46,47,48,49,50,51,     // Letters 'n' through 'z'
        -9,-9,-9,-9                                 // Decimal 123 - 126
        /*,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,     // Decimal 127 - 139
        -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,     // Decimal 140 - 152
        -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,     // Decimal 153 - 165
        -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,     // Decimal 166 - 178
        -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,     // Decimal 179 - 191
        -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,     // Decimal 192 - 204
        -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,     // Decimal 205 - 217
        -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,     // Decimal 218 - 230
        -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,     // Decimal 231 - 243
        -9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9,-9         // Decimal 244 - 255 */
    };

    // I think I end up not using the BAD_ENCODING indicator.
    //private final static byte BAD_ENCODING    = -9; // Indicates error in encoding
    private final static byte WHITE_SPACE_ENC = -5; // Indicates white space in encoding
    private final static byte EQUALS_SIGN_ENC = -1; // Indicates equals sign in encoding


    /** Defeats instantiation. */
    private Base64(){}



/* ********  E N C O D I N G   M E T H O D S  ******** */


    /**
     * Encodes up to the first three bytes of array <var>threeBytes</var>
     * and returns a four-byte array in Base64 notation.
     * The actual number of significant bytes in your array is
     * given by <var>numSigBytes</var>.
     * The array <var>threeBytes</var> needs only be as big as
     * <var>numSigBytes</var>.
     * Code can reuse a byte array by passing a four-byte array as <var>b4</var>.
     *
```

```
     * @param b4 A reusable byte array to reduce array instantiation
     * @param threeBytes the array to convert
     * @param numSigBytes the number of significant bytes in your array
     * @return four byte array in Base64 notation.
     * @since 1.5.1
     */
    private static byte[] encode3to4( byte[] b4, byte[] threeBytes, int numSigBytes )
    {
        encode3to4( threeBytes, 0, numSigBytes, b4, 0 );
        return b4;
    }   // end encode3to4


    /**
     * Encodes up to three bytes of the array <var>source</var>
     * and writes the resulting four Base64 bytes to <var>destination</var>.
     * The source and destination arrays can be manipulated
     * anywhere along their length by specifying
     * <var>srcOffset</var> and <var>destOffset</var>.
     * This method does not check to make sure your arrays
     * are large enough to accomodate <var>srcOffset</var> + 3 for
     * the <var>source</var> array or <var>destOffset</var> + 4 for
     * the <var>destination</var> array.
     * The actual number of significant bytes in your array is
     * given by <var>numSigBytes</var>.
     *
     * @param source the array to convert
     * @param srcOffset the index where conversion begins
     * @param numSigBytes the number of significant bytes in your array
     * @param destination the array to hold the conversion
     * @param destOffset the index where output will be put
     * @return the <var>destination</var> array
     * @since 1.3
     */
    private static byte[] encode3to4(
     byte[] source, int srcOffset, int numSigBytes,
     byte[] destination, int destOffset )
    {
        //           1         2         3
        // 01234567890123456789012345678901 Bit position
        // --------000000001111111122222222 Array position from threeBytes
        // --------|    ||    ||    ||    | Six bit groups to index ALPHABET
        //          >>18  >>12  >> 6  >> 0  Right shift necessary
        //                0x3f  0x3f  0x3f  Additional AND

        // Create buffer with zero-padding if there are only one or two
        // significant bytes passed in the array.
        // We have to shift left 24 in order to flush out the 1's that appear
        // when Java treats a value as negative that is cast from a byte to an int.
        int inBuff =   ( numSigBytes > 0 ? ((source[ srcOffset     ] << 24) >>>  8) : 0 )
                     | ( numSigBytes > 1 ? ((source[ srcOffset + 1 ] << 24) >>> 16) : 0 )
                     | ( numSigBytes > 2 ? ((source[ srcOffset + 2 ] << 24) >>> 24) : 0
);

        switch( numSigBytes )
        {
            case 3:
                destination[ destOffset     ] = ALPHABET[ (inBuff >>> 18)        ];
                destination[ destOffset + 1 ] = ALPHABET[ (inBuff >>> 12) & 0x3f ];
                destination[ destOffset + 2 ] = ALPHABET[ (inBuff >>>  6) & 0x3f ];
                destination[ destOffset + 3 ] = ALPHABET[ (inBuff       ) & 0x3f ];
                return destination;

            case 2:
                destination[ destOffset     ] = ALPHABET[ (inBuff >>> 18)        ];
                destination[ destOffset + 1 ] = ALPHABET[ (inBuff >>> 12) & 0x3f ];
                destination[ destOffset + 2 ] = ALPHABET[ (inBuff >>>  6) & 0x3f ];
                destination[ destOffset + 3 ] = EQUALS_SIGN;
                return destination;

            case 1:
```

```
                destination[ destOffset     ] = ALPHABET[ (inBuff >>> 18)        ];
                destination[ destOffset + 1 ] = ALPHABET[ (inBuff >>> 12) & 0x3f ];
                destination[ destOffset + 2 ] = EQUALS_SIGN;
                destination[ destOffset + 3 ] = EQUALS_SIGN;
                return destination;

            default:
                return destination;
        }   // end switch
    }   // end encode3to4



    /**
     * Serializes an object and returns the Base64-encoded
     * version of that serialized object. If the object
     * cannot be serialized or there is another error,
     * the method will return <tt>null</tt>.
     * The object is not GZip-compressed before being encoded.
     *
     * @param serializableObject The object to encode
     * @return The Base64-encoded object
     * @since 1.4
     */
    public static String encodeObject( java.io.Serializable serializableObject )
    {
        return encodeObject( serializableObject, NO_OPTIONS );
    }   // end encodeObject



    /**
     * Serializes an object and returns the Base64-encoded
     * version of that serialized object. If the object
     * cannot be serialized or there is another error,
     * the method will return <tt>null</tt>.
     * <p>
     * Valid options:<pre>
     *   GZIP: gzip-compresses object before encoding it.
     *   DONT_BREAK_LINES: don't break lines at 76 characters
     *     <i>Note: Technically, this makes your encoding non-compliant.</i>
     * </pre>
     * <p>
     * Example: <code>encodeObject( myObj, Base64.GZIP )</code> or
     * <p>
     * Example: <code>encodeObject( myObj, Base64.GZIP | Base64.DONT_BREAK_LINES )</code>
     *
     * @param serializableObject The object to encode
     * @param options Specified options
     * @return The Base64-encoded object
     * @see Base64#GZIP
     * @see Base64#DONT_BREAK_LINES
     * @since 2.0
     */
    public static String encodeObject( java.io.Serializable serializableObject, int
options )
    {
        // Streams
        java.io.ByteArrayOutputStream  baos  = null;
        java.io.OutputStream           b64os = null;
        java.io.ObjectOutputStream     oos   = null;
        java.util.zip.GZIPOutputStream gzos  = null;

        // Isolate options
        int gzip           = (options & GZIP);
        int dontBreakLines = (options & DONT_BREAK_LINES);

        try
        {
            // ObjectOutputStream -> (GZIP) -> Base64 -> ByteArrayOutputStream
            baos  = new java.io.ByteArrayOutputStream();
```

```
            b64os = new Base64.OutputStream( baos, ENCODE | dontBreakLines );

            // GZip?
            if( gzip == GZIP )
            {
                gzos = new java.util.zip.GZIPOutputStream( b64os );
                oos  = new java.io.ObjectOutputStream( gzos );
            }   // end if: gzip
            else
                oos   = new java.io.ObjectOutputStream( b64os );

            oos.writeObject( serializableObject );
        }   // end try
        catch( java.io.IOException e )
        {
            e.printStackTrace();
            return null;
        }   // end catch
        finally
        {
            try{ oos.close();   } catch( Exception e ){}
            try{ gzos.close();  } catch( Exception e ){}
            try{ b64os.close(); } catch( Exception e ){}
            try{ baos.close();  } catch( Exception e ){}
        }   // end finally

        // Return value according to relevant encoding.
        try
        {
            return new String( baos.toByteArray(), PREFERRED_ENCODING );
        }   // end try
        catch (java.io.UnsupportedEncodingException uue)
        {
            return new String( baos.toByteArray() );
        }   // end catch

    }   // end encode



    /**
     * Encodes a byte array into Base64 notation.
     * Does not GZip-compress data.
     *
     * @param source The data to convert
     * @since 1.4
     */
    public static String encodeBytes( byte[] source )
    {
        return encodeBytes( source, 0, source.length, NO_OPTIONS );
    }   // end encodeBytes



    /**
     * Encodes a byte array into Base64 notation.
     * <p>
     * Valid options:<pre>
     *   GZIP: gzip-compresses object before encoding it.
     *   DONT_BREAK_LINES: don't break lines at 76 characters
     *     <i>Note: Technically, this makes your encoding non-compliant.</i>
     * </pre>
     * <p>
     * Example: <code>encodeBytes( myData, Base64.GZIP )</code> or
     * <p>
     * Example: <code>encodeBytes( myData, Base64.GZIP | Base64.DONT_BREAK_LINES )</code>
     *
     *
     * @param source The data to convert
     * @param options Specified options
     * @see Base64#GZIP
```

```
 * @see Base64#DONT_BREAK_LINES
 * @since 2.0
 */
public static String encodeBytes( byte[] source, int options )
{
    return encodeBytes( source, 0, source.length, options );
}   // end encodeBytes


/**
 * Encodes a byte array into Base64 notation.
 * Does not GZip-compress data.
 *
 * @param source The data to convert
 * @param off Offset in array where conversion should begin
 * @param len Length of data to convert
 * @since 1.4
 */
public static String encodeBytes( byte[] source, int off, int len )
{
    return encodeBytes( source, off, len, NO_OPTIONS );
}   // end encodeBytes



/**
 * Encodes a byte array into Base64 notation.
 * <p>
 * Valid options:<pre>
 *   GZIP: gzip-compresses object before encoding it.
 *   DONT_BREAK_LINES: don't break lines at 76 characters
 *     <i>Note: Technically, this makes your encoding non-compliant.</i>
 * </pre>
 * <p>
 * Example: <code>encodeBytes( myData, Base64.GZIP )</code> or
 * <p>
 * Example: <code>encodeBytes( myData, Base64.GZIP | Base64.DONT_BREAK_LINES )</code>
 *
 *
 * @param source The data to convert
 * @param off Offset in array where conversion should begin
 * @param len Length of data to convert
 * @param options Specified options
 * @see Base64#GZIP
 * @see Base64#DONT_BREAK_LINES
 * @since 2.0
 */
public static String encodeBytes( byte[] source, int off, int len, int options )
{
    // Isolate options
    int dontBreakLines = ( options & DONT_BREAK_LINES );
    int gzip           = ( options & GZIP   );

    // Compress?
    if( gzip == GZIP )
    {
        java.io.ByteArrayOutputStream  baos  = null;
        java.util.zip.GZIPOutputStream gzos  = null;
        Base64.OutputStream            b64os = null;


        try
        {
            // GZip -> Base64 -> ByteArray
            baos = new java.io.ByteArrayOutputStream();
            b64os = new Base64.OutputStream( baos, ENCODE | dontBreakLines );
            gzos  = new java.util.zip.GZIPOutputStream( b64os );

            gzos.write( source, off, len );
            gzos.close();
        }   // end try
```

```
            catch( java.io.IOException e )
            {
                e.printStackTrace();
                return null;
            }   // end catch
            finally
            {
                try{ gzos.close();  } catch( Exception e ){}
                try{ b64os.close(); } catch( Exception e ){}
                try{ baos.close();  } catch( Exception e ){}
            }   // end finally

            // Return value according to relevant encoding.
            try
            {
                return new String( baos.toByteArray(), PREFERRED_ENCODING );
            }   // end try
            catch (java.io.UnsupportedEncodingException uue)
            {
                return new String( baos.toByteArray() );
            }   // end catch
        }   // end if: compress

        // Else, don't compress. Better not to use streams at all then.
        else
        {
            // Convert option to boolean in way that code likes it.
            boolean breakLines = false;

            int    len43   = len * 4 / 3;
            byte[] outBuff = new byte[   ( len43 )                      // Main 4:3
                                       + ( (len % 3) > 0 ? 4 : 0 )      // Account for
padding
                                       + (breakLines ? ( len43 / MAX_LINE_LENGTH ) : 0)
]; // New lines
            int d = 0;
            int e = 0;
            int len2 = len - 2;
            int lineLength = 0;
            for( ; d < len2; d+=3, e+=4 )
            {
                encode3to4( source, d+off, 3, outBuff, e );

                lineLength += 4;
                if( breakLines && lineLength == MAX_LINE_LENGTH )
                {
                    outBuff[e+4] = NEW_LINE;
                    e++;
                    lineLength = 0;
                }   // end if: end of line
            }   // en dfor: each piece of array

            if( d < len )
            {
                encode3to4( source, d+off, len - d, outBuff, e );
                e += 4;
            }   // end if: some padding needed


            // Return value according to relevant encoding.
            try
            {
                return new String( outBuff, 0, e, PREFERRED_ENCODING );
            }   // end try
            catch (java.io.UnsupportedEncodingException uue)
            {
                return new String( outBuff, 0, e );
            }   // end catch

        }   // end else: don't compress
```

```
    }   // end encodeBytes



/* ********  D E C O D I N G   M E T H O D S  ******** */


    /**
     * Decodes four bytes from array <var>source</var>
     * and writes the resulting bytes (up to three of them)
     * to <var>destination</var>.
     * The source and destination arrays can be manipulated
     * anywhere along their length by specifying
     * <var>srcOffset</var> and <var>destOffset</var>.
     * This method does not check to make sure your arrays
     * are large enough to accomodate <var>srcOffset</var> + 4 for
     * the <var>source</var> array or <var>destOffset</var> + 3 for
     * the <var>destination</var> array.
     * This method returns the actual number of bytes that
     * were converted from the Base64 encoding.
     *
     *
     * @param source the array to convert
     * @param srcOffset the index where conversion begins
     * @param destination the array to hold the conversion
     * @param destOffset the index where output will be put
     * @return the number of decoded bytes converted
     * @since 1.3
     */
    private static int decode4to3( byte[] source, int srcOffset, byte[] destination, int
destOffset )
    {
        // Example: Dk==
        if( source[ srcOffset + 2] == EQUALS_SIGN )
        {
            // Two ways to do the same thing. Don't know which way I like best.
            //int outBuff =   ( ( DECODABET[ source[ srcOffset    ] ] << 24 ) >>>  6 )
            //              | ( ( DECODABET[ source[ srcOffset + 1] ] << 24 ) >>> 12 );
            int outBuff =   ( ( DECODABET[ source[ srcOffset    ] ] & 0xFF ) << 18 )
                          | ( ( DECODABET[ source[ srcOffset + 1] ] & 0xFF ) << 12 );

            destination[ destOffset ] = (byte)( outBuff >>> 16 );
            return 1;
        }

        // Example: DkL=
        else if( source[ srcOffset + 3 ] == EQUALS_SIGN )
        {
            // Two ways to do the same thing. Don't know which way I like best.
            //int outBuff =   ( ( DECODABET[ source[ srcOffset     ] ] << 24 ) >>>  6 )
            //              | ( ( DECODABET[ source[ srcOffset + 1 ] ] << 24 ) >>> 12 )
            //              | ( ( DECODABET[ source[ srcOffset + 2 ] ] << 24 ) >>> 18 );
            int outBuff =   ( ( DECODABET[ source[ srcOffset     ] ] & 0xFF ) << 18 )
                          | ( ( DECODABET[ source[ srcOffset + 1 ] ] & 0xFF ) << 12 )
                          | ( ( DECODABET[ source[ srcOffset + 2 ] ] & 0xFF ) <<  6 );

            destination[ destOffset     ] = (byte)( outBuff >>> 16 );
            destination[ destOffset + 1 ] = (byte)( outBuff >>>  8 );
            return 2;
        }

        // Example: DkLE
        else
        {
            try{
            // Two ways to do the same thing. Don't know which way I like best.
            //int outBuff =   ( ( DECODABET[ source[ srcOffset     ] ] << 24 ) >>>  6 )
            //              | ( ( DECODABET[ source[ srcOffset + 1 ] ] << 24 ) >>> 12 )
            //              | ( ( DECODABET[ source[ srcOffset + 2 ] ] << 24 ) >>> 18 )
```

```
            //              | ( ( DECODABET[ source[ srcOffset + 3 ] ] << 24 ) >>> 24 );
            int outBuff =   ( ( DECODABET[ source[ srcOffset     ] ] & 0xFF ) << 18 )
                          | ( ( DECODABET[ source[ srcOffset + 1 ] ] & 0xFF ) << 12 )
                          | ( ( DECODABET[ source[ srcOffset + 2 ] ] & 0xFF ) <<  6)
                          | ( ( DECODABET[ source[ srcOffset + 3 ] ] & 0xFF )      );


            destination[ destOffset     ] = (byte)( outBuff >> 16 );
            destination[ destOffset + 1 ] = (byte)( outBuff >>  8 );
            destination[ destOffset + 2 ] = (byte)( outBuff       );

            return 3;
            }catch( Exception e){
                System.out.println(""+source[srcOffset]+ ": " + ( DECODABET[ source[
srcOffset     ] ]  ) );
                System.out.println(""+source[srcOffset+1]+  ": " + ( DECODABET[ source[
srcOffset + 1 ] ]  ) );
                System.out.println(""+source[srcOffset+2]+  ": " + ( DECODABET[ source[
srcOffset + 2 ] ]  ) );
                System.out.println(""+source[srcOffset+3]+  ": " + ( DECODABET[ source[
srcOffset + 3 ] ]  ) );
                return -1;
            }   //e nd catch
        }
    }   // end decodeToBytes



    /**
     * Very low-level access to decoding ASCII characters in
     * the form of a byte array. Does not support automatically
     * gunzipping or any other "fancy" features.
     *
     * @param source The Base64 encoded data
     * @param off    The offset of where to begin decoding
     * @param len    The length of characters to decode
     * @return decoded data
     * @since 1.3
     */
    public static byte[] decode( byte[] source, int off, int len )
    {
        int    len34   = len * 3 / 4;
        byte[] outBuff = new byte[ len34 ]; // Upper limit on size of output
        int    outBuffPosn = 0;

        byte[] b4        = new byte[4];
        int    b4Posn    = 0;
        int    i         = 0;
        byte   sbiCrop   = 0;
        byte   sbiDecode = 0;
        for( i = off; i < off+len; i++ )
        {
            sbiCrop = (byte)(source[i] & 0x7f); // Only the low seven bits
            sbiDecode = DECODABET[ sbiCrop ];

            if( sbiDecode >= WHITE_SPACE_ENC ) // White space, Equals sign or better
            {
                if( sbiDecode >= EQUALS_SIGN_ENC )
                {
                    b4[ b4Posn++ ] = sbiCrop;
                    if( b4Posn > 3 )
                    {
                        outBuffPosn += decode4to3( b4, 0, outBuff, outBuffPosn );
                        b4Posn = 0;

                        // If that was the equals sign, break out of 'for' loop
                        if( sbiCrop == EQUALS_SIGN )
                            break;
                    }   // end if: quartet built
```

```
                }   // end if: equals sign or better

            }   // end if: white space, equals sign or better
            else
            {
                System.err.println( "Bad Base64 input character at " + i + ": " +
source[i] + "(decimal)" );
                return null;
            }   // end else:
        }   // each input character

        byte[] out = new byte[ outBuffPosn ];
        System.arraycopy( outBuff, 0, out, 0, outBuffPosn );
        return out;
    }   // end decode



    /**
     * Decodes data from Base64 notation, automatically
     * detecting gzip-compressed data and decompressing it.
     *
     * @param s the string to decode
     * @return the decoded data
     * @since 1.4
     */
    public static byte[] decode( String s )
    {
        byte[] bytes;
        try
        {
            bytes = s.getBytes( PREFERRED_ENCODING );
        }   // end try
        catch( java.io.UnsupportedEncodingException uee )
        {
            bytes = s.getBytes();
        }   // end catch
               //</change>

        // Decode
        bytes = decode( bytes, 0, bytes.length );


        // Check to see if it's gzip-compressed
        // GZIP Magic Two-Byte Number: 0x8b1f (35615)
        if( bytes != null && bytes.length >= 4 )
        {

            int head = ((int)bytes[0] & 0xff) | ((bytes[1] << 8) & 0xff00);
            if( java.util.zip.GZIPInputStream.GZIP_MAGIC == head )
            {
                java.io.ByteArrayInputStream  bais = null;
                java.util.zip.GZIPInputStream gzis = null;
                java.io.ByteArrayOutputStream baos = null;
                byte[] buffer = new byte[2048];
                int    length = 0;

                try
                {
                    baos = new java.io.ByteArrayOutputStream();
                    bais = new java.io.ByteArrayInputStream( bytes );
                    gzis = new java.util.zip.GZIPInputStream( bais );

                    while( ( length = gzis.read( buffer ) ) >= 0 )
                    {
                        baos.write(buffer,0,length);
                    }   // end while: reading input

                    // No error? Get new bytes.
                    bytes = baos.toByteArray();
```

```
            }   // end try
            catch( java.io.IOException e )
            {
                // Just return originally-decoded bytes
            }   // end catch
            finally
            {
                try{ baos.close(); } catch( Exception e ){}
                try{ gzis.close(); } catch( Exception e ){}
                try{ bais.close(); } catch( Exception e ){}
            }   // end finally

        }   // end if: gzipped
    }   // end if: bytes.length >= 2

    return bytes;
}   // end decode



/**
 * Attempts to decode Base64 data and deserialize a Java
 * Object within. Returns <tt>null</tt> if there was an error.
 *
 * @param encodedObject The Base64 data to decode
 * @return The decoded and deserialized object
 * @since 1.5
 */
public static Object decodeToObject( String encodedObject )
{
    // Decode and gunzip if necessary
    byte[] objBytes = decode( encodedObject );

    java.io.ByteArrayInputStream  bais = null;
    java.io.ObjectInputStream     ois  = null;
    Object obj = null;

    try
    {
        bais = new java.io.ByteArrayInputStream( objBytes );
        ois  = new java.io.ObjectInputStream( bais );

        obj = ois.readObject();
    }   // end try
    catch( java.io.IOException e )
    {
        e.printStackTrace();
        obj = null;
    }   // end catch
    catch( java.lang.ClassNotFoundException e )
    {
        e.printStackTrace();
        obj = null;
    }   // end catch
    finally
    {
        try{ bais.close(); } catch( Exception e ){}
        try{ ois.close();  } catch( Exception e ){}
    }   // end finally

    return obj;
}   // end decodeObject



/**
 * Convenience method for encoding data to a file.
 *
 * @param dataToEncode byte array of data to encode in base64 form
```

```
     * @param filename Filename for saving encoded data
     * @return <tt>true</tt> if successful, <tt>false</tt> otherwise
     *
     * @since 2.1
     */
    public static boolean encodeToFile( byte[] dataToEncode, String filename )
    {
        boolean success = false;
        Base64.OutputStream bos = null;
        try
        {
            bos = new Base64.OutputStream(
                      new java.io.FileOutputStream( filename ), Base64.ENCODE );
            bos.write( dataToEncode );
            success = true;
        }   // end try
        catch( java.io.IOException e )
        {

            success = false;
        }   // end catch: IOException
        finally
        {
            try{ bos.close(); } catch( Exception e ){}
        }   // end finally

        return success;
    }   // end encodeToFile


    /**
     * Convenience method for decoding data to a file.
     *
     * @param dataToDecode Base64-encoded data as a string
     * @param filename Filename for saving decoded data
     * @return <tt>true</tt> if successful, <tt>false</tt> otherwise
     *
     * @since 2.1
     */
    public static boolean decodeToFile( String dataToDecode, String filename )
    {
        boolean success = false;
        Base64.OutputStream bos = null;
        try
        {
                bos = new Base64.OutputStream(
                          new java.io.FileOutputStream( filename ), Base64.DECODE );
                bos.write( dataToDecode.getBytes( PREFERRED_ENCODING ) );
                success = true;
        }   // end try
        catch( java.io.IOException e )
        {
            success = false;
        }   // end catch: IOException
        finally
        {
                try{ bos.close(); } catch( Exception e ){}
        }   // end finally

        return success;
    }   // end decodeToFile




    /**
     * Convenience method for reading a base64-encoded
     * file and decoding it.
     *
     * @param filename Filename for reading encoded data
     * @return decoded byte array or null if unsuccessful
```

```
     *
     * @since 2.1
     */
    public static byte[] decodeFromFile( String filename )
    {
        byte[] decodedData = null;
        Base64.InputStream bis = null;
        try
        {
            // Set up some useful variables
            java.io.File file = new java.io.File( filename );
            byte[] buffer = null;
            int length   = 0;
            int numBytes = 0;

            // Check for size of file
            if( file.length() > Integer.MAX_VALUE )
            {
                System.err.println( "File is too big for this convenience method (" +
file.length() + " bytes)." );
                return null;
            }   // end if: file too big for int index
            buffer = new byte[ (int)file.length() ];

            // Open a stream
            bis = new Base64.InputStream(
                      new java.io.BufferedInputStream(
                      new java.io.FileInputStream( file ) ), Base64.DECODE );

            // Read until done
            while( ( numBytes = bis.read( buffer, length, 4096 ) ) >= 0 )
                length += numBytes;

            // Save in a variable to return
            decodedData = new byte[ length ];
            System.arraycopy( buffer, 0, decodedData, 0, length );

        }   // end try
        catch( java.io.IOException e )
        {
            System.err.println( "Error decoding from file " + filename );
        }   // end catch: IOException
        finally
        {
            try{ bis.close(); } catch( Exception e) {}
        }   // end finally

        return decodedData;
    }   // end decodeFromFile



    /**
     * Convenience method for reading a binary file
     * and base64-encoding it.
     *
     * @param filename Filename for reading binary data
     * @return base64-encoded string or null if unsuccessful
     *
     * @since 2.1
     */
    public static String encodeFromFile( String filename )
    {
        String encodedData = null;
        Base64.InputStream bis = null;
        try
        {
            // Set up some useful variables
            java.io.File file = new java.io.File( filename );
            byte[] buffer = new byte[ (int)(file.length() * 1.4) ];
            int length   = 0;
```

```
        int numBytes = 0;

        // Open a stream
        bis = new Base64.InputStream(
                  new java.io.BufferedInputStream(
                  new java.io.FileInputStream( file ) ), Base64.ENCODE );

        // Read until done
        while( ( numBytes = bis.read( buffer, length, 4096 ) ) >= 0 )
            length += numBytes;

        // Save in a variable to return
        encodedData = new String( buffer, 0, length, Base64.PREFERRED_ENCODING );

    }   // end try
    catch( java.io.IOException e )
    {
        System.err.println( "Error encoding from file " + filename );
    }   // end catch: IOException
    finally
    {
        try{ bis.close(); } catch( Exception e) {}
    }   // end finally

    return encodedData;
    }   // end encodeFromFile



/* ********  I N N E R   C L A S S   I N P U T S T R E A M  ******** */



/**
 * A {@link Base64.InputStream} will read data from another
 * <tt>java.io.InputStream</tt>, given in the constructor,
 * and encode/decode to/from Base64 notation on the fly.
 *
 * @see Base64
 * @since 1.3
 */
public static class InputStream extends java.io.FilterInputStream
{
    private boolean encode;         // Encoding or decoding
    private int     position;       // Current position in the buffer
    private byte[]  buffer;         // Small buffer holding converted data
    private int     bufferLength;   // Length of buffer (3 or 4)
    private int     numSigBytes;    // Number of meaningful bytes in the buffer
    private int     lineLength;
    private boolean breakLines;     // Break lines at less than 80 characters


    /**
     * Constructs a {@link Base64.InputStream} in DECODE mode.
     *
     * @param in the <tt>java.io.InputStream</tt> from which to read data.
     * @since 1.3
     */
    public InputStream( java.io.InputStream in )
    {
        this( in, DECODE );
    }   // end constructor


    /**
     * Constructs a {@link Base64.InputStream} in
     * either ENCODE or DECODE mode.
     * <p>
     * Valid options:<pre>
     *   ENCODE or DECODE: Encode or Decode as data is read.
```

```
 *   DONT_BREAK_LINES: don't break lines at 76 characters
 *     (only meaningful when encoding)
 *     <i>Note: Technically, this makes your encoding non-compliant.</i>
 * </pre>
 * <p>
 * Example: <code>new Base64.InputStream( in, Base64.DECODE )</code>
 *
 *
 * @param in the <tt>java.io.InputStream</tt> from which to read data.
 * @param options Specified options
 * @see Base64#ENCODE
 * @see Base64#DECODE
 * @see Base64#DONT_BREAK_LINES
 * @since 2.0
 */
public InputStream( java.io.InputStream in, int options )
{
    super( in );
    this.breakLines   = (options & DONT_BREAK_LINES) != DONT_BREAK_LINES;
    this.encode       = (options & ENCODE) == ENCODE;
    this.bufferLength = encode ? 4 : 3;
    this.buffer   = new byte[ bufferLength ];
    this.position = -1;
    this.lineLength = 0;
}   // end constructor

/**
 * Reads enough of the input stream to convert
 * to/from Base64 and returns the next byte.
 *
 * @return next byte
 * @since 1.3
 */
public int read() throws java.io.IOException
{
    // Do we need to get data?
    if( position < 0 )
    {
        if( encode )
        {
            byte[] b3 = new byte[3];
            int numBinaryBytes = 0;
            for( int i = 0; i < 3; i++ )
            {
                try
                {
                    int b = in.read();

                    // If end of stream, b is -1.
                    if( b >= 0 )
                    {
                        b3[i] = (byte)b;
                        numBinaryBytes++;
                    }   // end if: not end of stream

                }   // end try: read
                catch( java.io.IOException e )
                {
                    // Only a problem if we got no data at all.
                    if( i == 0 )
                        throw e;

                }   // end catch
            }   // end for: each needed input byte

            if( numBinaryBytes > 0 )
            {
                encode3to4( b3, 0, numBinaryBytes, buffer, 0 );
                position = 0;
                numSigBytes = 4;
            }   // end if: got data
```

```
                    else
                    {
                        return -1;
                    }   // end else
                }   // end if: encoding

                // Else decoding
                else
                {
                    byte[] b4 = new byte[4];
                    int i = 0;
                    for( i = 0; i < 4; i++ )
                    {
                        // Read four "meaningful" bytes:
                        int b = 0;
                        do{ b = in.read(); }
                        while( b >= 0 && DECODABET[ b & 0x7f ] <= WHITE_SPACE_ENC );

                        if( b < 0 )
                            break; // Reads a -1 if end of stream

                        b4[i] = (byte)b;
                    }   // end for: each needed input byte

                    if( i == 4 )
                    {
                        numSigBytes = decode4to3( b4, 0, buffer, 0 );
                        position = 0;
                    }   // end if: got four characters
                    else if( i == 0 ){
                        return -1;
                    }   // end else if: also padded correctly
                    else
                    {
                        // Must have broken out from above.
                        throw new java.io.IOException( "Improperly padded Base64 input."
);
                    }   // end

                }   // end else: decode
            }   // end else: get data

            // Got data?
            if( position >= 0 )
            {
                // End of relevant data?
                if( /*!encode &&*/ position >= numSigBytes )
                    return -1;

                if( encode && breakLines && lineLength >= MAX_LINE_LENGTH )
                {
                    lineLength = 0;
                    return '\n';
                }   // end if
                else
                {
                    lineLength++;   // This isn't important when decoding
                                    // but throwing an extra "if" seems
                                    // just as wasteful.

                    int b = buffer[ position++ ];

                    if( position >= bufferLength )
                        position = -1;

                    return b & 0xFF; // This is how you "cast" a byte that's
                                     // intended to be unsigned.
                }   // end else
            }   // end if: position >= 0

            // Else error
```

```
        else
        {
            // When JDK1.4 is more accepted, use an assertion here.
            throw new java.io.IOException( "Error in Base64 code reading stream." );
        }   // end else
    }   // end read


    /**
     * Calls {@link #read()} repeatedly until the end of stream
     * is reached or <var>len</var> bytes are read.
     * Returns number of bytes read into array or -1 if
     * end of stream is encountered.
     *
     * @param dest array to hold values
     * @param off offset for array
     * @param len max number of bytes to read into array
     * @return bytes read into array or -1 if end of stream is encountered.
     * @since 1.3
     */
    public int read( byte[] dest, int off, int len ) throws java.io.IOException
    {
        int i;
        int b;
        for( i = 0; i < len; i++ )
        {
            b = read();

            //if( b < 0 && i == 0 )
            //    return -1;

            if( b >= 0 )
                dest[off + i] = (byte)b;
            else if( i == 0 )
                return -1;
            else
                break; // Out of 'for' loop
        }   // end for: each byte read
        return i;
    }   // end read

}   // end inner class InputStream




/* ********  I N N E R   C L A S S   O U T P U T S T R E A M  ******** */



/**
 * A {@link Base64.OutputStream} will write data to another
 * <tt>java.io.OutputStream</tt>, given in the constructor,
 * and encode/decode to/from Base64 notation on the fly.
 *
 * @see Base64
 * @since 1.3
 */
public static class OutputStream extends java.io.FilterOutputStream
{
    private boolean encode;
    private int     position;
    private byte[]  buffer;
    private int     bufferLength;
    private int     lineLength;
    private boolean breakLines;
    private byte[]  b4; // Scratch used in a few places
    private boolean suspendEncoding;
```

```
/**
 * Constructs a {@link Base64.OutputStream} in ENCODE mode.
 *
 * @param out the <tt>java.io.OutputStream</tt> to which data will be written.
 * @since 1.3
 */
public OutputStream( java.io.OutputStream out )
{
    this( out, ENCODE );
}   // end constructor


/**
 * Constructs a {@link Base64.OutputStream} in
 * either ENCODE or DECODE mode.
 * <p>
 * Valid options:<pre>
 *   ENCODE or DECODE: Encode or Decode as data is read.
 *   DONT_BREAK_LINES: don't break lines at 76 characters
 *     (only meaningful when encoding)
 *     <i>Note: Technically, this makes your encoding non-compliant.</i>
 * </pre>
 * <p>
 * Example: <code>new Base64.OutputStream( out, Base64.ENCODE )</code>
 *
 * @param out the <tt>java.io.OutputStream</tt> to which data will be written.
 * @param options Specified options.
 * @see Base64#ENCODE
 * @see Base64#DECODE
 * @see Base64#DONT_BREAK_LINES
 * @since 1.3
 */
public OutputStream( java.io.OutputStream out, int options )
{
    super( out );
    this.breakLines   = (options & DONT_BREAK_LINES) != DONT_BREAK_LINES;
    this.encode       = (options & ENCODE) == ENCODE;
    this.bufferLength = encode ? 3 : 4;
    this.buffer       = new byte[ bufferLength ];
    this.position     = 0;
    this.lineLength   = 0;
    this.suspendEncoding = false;
    this.b4           = new byte[4];
}   // end constructor


/**
 * Writes the byte to the output stream after
 * converting to/from Base64 notation.
 * When encoding, bytes are buffered three
 * at a time before the output stream actually
 * gets a write() call.
 * When decoding, bytes are buffered four
 * at a time.
 *
 * @param theByte the byte to write
 * @since 1.3
 */
public void write(int theByte) throws java.io.IOException
{
    // Encoding suspended?
    if( suspendEncoding )
    {
        super.out.write( theByte );
        return;
    }   // end if: supsended

    // Encode?
    if( encode )
    {
        buffer[ position++ ] = (byte)theByte;
```

```
        if( position >= bufferLength )  // Enough to encode.
        {
            out.write( encode3to4( b4, buffer, bufferLength ) );

            lineLength += 4;
            if( breakLines && lineLength >= MAX_LINE_LENGTH )
            {
                out.write( NEW_LINE );
                lineLength = 0;
            }   // end if: end of line

            position = 0;
        }   // end if: enough to output
    }   // end if: encoding

    // Else, Decoding
    else
    {
        // Meaningful Base64 character?
        if( DECODABET[ theByte & 0x7f ] > WHITE_SPACE_ENC )
        {
            buffer[ position++ ] = (byte)theByte;
            if( position >= bufferLength )  // Enough to output.
            {
                int len = Base64.decode4to3( buffer, 0, b4, 0 );
                out.write( b4, 0, len );
                //out.write( Base64.decode4to3( buffer ) );
                position = 0;
            }   // end if: enough to output
        }   // end if: meaningful base64 character
        else if( DECODABET[ theByte & 0x7f ] != WHITE_SPACE_ENC )
        {
            throw new java.io.IOException( "Invalid character in Base64 data." );
        }   // end else: not white space either
    }   // end else: decoding
}   // end write



/**
 * Calls {@link #write(int)} repeatedly until <var>len</var>
 * bytes are written.
 *
 * @param theBytes array from which to read bytes
 * @param off offset for array
 * @param len max number of bytes to read into array
 * @since 1.3
 */
public void write( byte[] theBytes, int off, int len ) throws java.io.IOException
{
    // Encoding suspended?
    if( suspendEncoding )
    {
        super.out.write( theBytes, off, len );
        return;
    }   // end if: supsended

    for( int i = 0; i < len; i++ )
    {
        write( theBytes[ off + i ] );
    }   // end for: each byte written

}   // end write



/**
 * Method added by PHIL. [Thanks, PHIL. -Rob]
 * This pads the buffer without closing the stream.
 */
public void flushBase64() throws java.io.IOException
```

```
        {
            if( position > 0 )
            {
                if( encode )
                {
                    out.write( encode3to4( b4, buffer, position ) );
                    position = 0;
                }   // end if: encoding
                else
                {
                    throw new java.io.IOException( "Base64 input not properly padded." );
                }   // end else: decoding
            }   // end if: buffer partially full

        }   // end flush


        /**
         * Flushes and closes (I think, in the superclass) the stream.
         *
         * @since 1.3
         */
        public void close() throws java.io.IOException
        {
            // 1. Ensure that pending characters are written
            flushBase64();

            // 2. Actually close the stream
            // Base class both flushes and closes.
            super.close();

            buffer = null;
            out    = null;
        }   // end close



        /**
         * Suspends encoding of the stream.
         * May be helpful if you need to embed a piece of
         * base640-encoded data in a stream.
         *
         * @since 1.5.1
         */
        public void suspendEncoding() throws java.io.IOException
        {
            flushBase64();
            this.suspendEncoding = true;
        }   // end suspendEncoding


        /**
         * Resumes encoding of the stream.
         * May be helpful if you need to embed a piece of
         * base640-encoded data in a stream.
         *
         * @since 1.5.1
         */
        public void resumeEncoding()
        {
            this.suspendEncoding = false;
        }   // end resumeEncoding



    }   // end inner class OutputStream


}   // end class Base64
```

10.5.2 Config

```
package SpamCash.Utilities;


public class Config
{
    //System names
    public final static String name = "SpamCash";
    public final static String MimeDescriptionOfCoinAttachment = "X-SpamCash-Coin";
    public final static String headerNotificationName = "X-SpamCash-Notification";
    public final static String defaultCESUsername = "user1";
    public final static String defaultCESPassword = "user1";
    public final static String defaultLocalPassword = "password";

    //System properties

    public final static long blacklistUpdateInterval = 1000*60*1; // 1 minute

    //Currency properties
    public final static double coinValue = 0.05;
    public final static double registrationCoinValue = 0.1;//5.6;
    public final static double registrationFee = 50.0;
    public final static int maxNumberOfUsages = 10;
    private final static long day = (long)(24*60*60*1000.0);
    public final static long expirationTimeInMilliseconds = 365*day;
    public final static long CCoinExpirationTimeInMilliseconds = 10*day;
    public final static long CCoinDelayTimeInMilliseconds = 3*day;
    public final static int blindingFactorBits = 2048;
    public final static int numberOfCoinsPerCoinset = 2;

    //Directorys
    public final static String mainDirectory = "data";
    public final static String incomingMailDirectory =
mainDirectory+System.getProperty("file.separator")+"incoming";
    public final static String tempIncomingMailDirectory =
mainDirectory+System.getProperty("file.separator")+"tempIncoming";
    public final static String outgoingMailDirectory =
mainDirectory+System.getProperty("file.separator")+"outgoing";
    public final static String keydir = mainDirectory +
System.getProperty("file.separator") + "Keys";
    public final static String certdir = mainDirectory +
System.getProperty("file.separator") + "Certificates";
    public final static String imagedir = mainDirectory +
System.getProperty("file.separator") + "images";
    public final static String OpenSSLpath = System.getProperty("file.separator") +"RS" +
System.getProperty("file.separator") + "CA"+System.getProperty("file.separator");

    //Filenames
    public final static String processedMailFilename = "ready to go.msg";
    public final static boolean safeInOneFile = true;
    public final static String safeFilePath =
mainDirectory+System.getProperty("file.separator")+"Safe.sc";
    public final static String privateKeyFilePath =
mainDirectory+System.getProperty("file.separator")+"Safe part 1 privatekey.sc";
    public final static String pendingMailsFilePath =
mainDirectory+System.getProperty("file.separator")+"Safe part 2 pending mails.sc";
    public final static String coinHistoryFilePath =
mainDirectory+System.getProperty("file.separator")+"Safe part 3 coin history.sc";
    public final static String defaultSafeKeyFilePath =
mainDirectory+System.getProperty("file.separator")+"Safe-key.key";
    public final static String clientCertificatePath =
mainDirectory+System.getProperty("file.separator")+ "certificate.crt";
    public final static String configurationFilePath =
mainDirectory+System.getProperty("file.separator")+ "Configuration.sc";
    public final static String keystoreFilePath =
mainDirectory+System.getProperty("file.separator")+ "keystore";
```

```
    public final static String blacklistFilePath =
mainDirectory+System.getProperty("file.separator")+ "blacklist.sc";
    public final static String whitelistFilePath =
mainDirectory+System.getProperty("file.separator")+ "whitelist.sc";
    public final static String logFolderPath =
mainDirectory+System.getProperty("file.separator")+"log"+System.getProperty("file.separat
or");
    public final static String CESConfigFilePath =
mainDirectory+System.getProperty("file.separator")+"CESdata.sc";


    //Security
    public final static int clientAsymmetricKeySize = 1024;
    public final static String CESPrivateKeyName  = Config.keydir +
System.getProperty("file.separator") + "CES_private.key";
    public final static String CESCertificateName = Config.certdir +
System.getProperty("file.separator") + "CES_certificate.crt";
    public final static String BLSPrivateKeyName  = Config.keydir +
System.getProperty("file.separator") + "BLS_private.key";
    public final static String BLSCertificateName = Config.certdir +
System.getProperty("file.separator") + "BLS_certificate.crt";
    public final static String RSPrivateKeyName   = Config.keydir +
System.getProperty("file.separator") + "RS_private.key";
    public final static String RSCertificateName  = Config.certdir +
System.getProperty("file.separator") + "RS_certificate.crt";

    //Proxy Configuration
    public static ProxyConfiguration configuration;

    /**
     * Maximum number of connections Through the IMAPProxy
     */
    public final static int maxIMAPConnections = 4;

    //Maximum number of bytes per header...
    public final static int maxHeaderSize = 100;

    //Expirationtime for a cryptographic session 20 min.
    public final static long expirationPeriod = 20 * 60 * 1000;

    // CurrencyExchangeServer IP address, port
    public final static String CESAddress = "localhost";
    public final static int CESPort = 41000;

    // BlackListServer IP address, port
    public final static String BLSAddress = "localhost";
    public final static int BLSPort = 42000;

    // VerificationServer IP address, port
    public final static String VSAddress = "localhost";
    public final static int VSPort = 43000;

    // RegistrationServer IP address, port
    public final static String RSAddress = "localhost";
    public final static int RSPort = 44000;

    //Progress indication
    public final static int numberOfUpdates = 100;
    public final static boolean hashEmailAdresses = true;

    public static void setProxyConfiguration(ProxyConfiguration config)
    {
        configuration = config;
    }


}
```

10.5.3 Cryptotools

```
/*

 BouncyCastle install...
 bcprov-jdk14-120.jar placeres under C:\j2sdk1.4.2_02\jre\lib\ext
 Derudover tilføjes linien:
"security.provider.6=org.bouncycastle.jce.provider.BouncyCastleProvider"
 uden "" til filen: "C:\j2sdk1.4.2_02\jre\lib\security\java.security".
 bcprov-jdk14-120.jar skal muligvis tilføjes til classpath.
 */

package SpamCash.Utilities;

import javax.crypto.*;
import javax.crypto.spec.*;
import java.security.cert.Certificate;
import java.security.interfaces.*;
import java.math.*;
import java.security.*;
import java.io.*;
import java.util.*;
import javax.crypto.*;
import SpamCash.NetworkPackets.*;
import SpamCash.Currency.Coin;
//import org.bouncycastle.crypto.params.RSAKeyParameters;
import org.logi.crypto.keys.RSABlindingFactor;
import org.logi.crypto.sign.BlindingFactor;
import org.logi.crypto.sign.Fingerprint;
import org.logi.crypto.sign.BlindSignature;
import org.logi.crypto.sign.BlindFingerprint;
import org.logi.crypto.keys.KeyException;
import SpamCash.Client.Control;
import SpamCash.Utilities.Debugger;
import java.security.KeyFactory;
import java.security.spec.PKCS8EncodedKeySpec;
import SpamCash.Currency.BlindedCoin;

public class Cryptotools
{
    private final static int encryptedKeySize = 128;

    private final static int signSize = 329;
    private static byte[] iv = new byte[16];
    private static SecureRandom sec_rand;

    private final static Class className = Cryptotools.class;

    public static void main(String[] s)
    {
        try
        {

            //generateCertificate("BLS", 2048);
            //generateCertificate("CES", 2048);
            extractPrivatekey();

            //generateRSAkeys();
            // testBase64Coding();
            //testEncryptWithPassword();
//                  testRSA();
//              testBlindSign2();
//                  testAES();
//                  testDSA();
//                  testEncrypt();

            /*                Email p1 = new Email("1test1","1sdftest2","tes
t3","test3");
               System.out.println("Email: "+p1);

               p1 = encryptAndSign(p1,"townhall","department2","citizen2");
```

```
                System.out.println("Encrypted mail: "+p1);

                p1 = partialDecrypt(p1,"distributer");

                System.out.println("Partial Encrypted mail: "+p1);

                p1 = partialEncrypt(p1,"distributer");

                System.out.println("Encrypted mail: "+p1);

                p1 = decryptAndVerify(p1,"townhall","department2",new
String[]{"citizen1","citizen2"});

                System.out.println("Email: "+p1);
             */

            Vector test = new Vector();
            test.addElement("1st element");
            Fingerprint hashed = SHA1(test);
            test.addElement("2nd element");
            merge(test, hashed);
        }
        catch(Exception e)
        {
            Debugger.debug(className, 3, "Initialization error!:\n\n" + e);
        }

    }

    public static void testEncryptWithPassword() throws Exception
    {
        String pass = "passwd";
        String data = "encrypted";

        Debugger.debug(className, 3, "data: " + data + "  decrypt(encrypt(data)): "
                       + decryptWithPassword(encryptWithPassword(data, pass), pass));

    }

    public static SecretKey createSecretKey(String data) throws Exception
    {
        byte[] temp = SHA1(data).getBytes();
        ;
        byte[] key = new byte[16];

        for(int i = 0; i < key.length; i++)
        {
            key[i] = temp[i];
        }
        return new SecretKeySpec(key, "AES");
    }

    public static byte[] encryptWithPassword(Object obj, String passwd) throws Exception
    {
        return encryptAES(obj, createSecretKey(passwd));
    }

    public static Object decryptWithPassword(byte[] data, String passwd) throws Exception
    {
        return byteArrayToObject(decryptAES(data, createSecretKey(passwd)));
    }

    public static RSABlindingFactor createBlindingFactor()
    {
        org.logi.crypto.Crypto.initRandom();
        return new RSABlindingFactor(new BigInteger(Config.blindingFactorBits, new
Random()));
    }
```

```
    public static BlindFingerprint blind(RSAPublicKey pub, RSABlindingFactor bf,
Fingerprint hash) throws Exception
    {
        org.logi.crypto.keys.RSAPublicKey rsaPub = new
org.logi.crypto.keys.RSAPublicKey(pub.getPublicExponent(),

pub.getModulus());
        return rsaPub.blind(hash, bf);
    }

    public static BlindSignature blindSign(RSAPrivateKey priv, BlindFingerprint blinded)
throws Exception
    {
        org.logi.crypto.keys.RSAPrivateKey rsaPriv = new
org.logi.crypto.keys.RSAPrivateKey(priv.getPrivateExponent(),

priv.getModulus());
        return rsaPriv.sign(blinded);
    }

    public static org.logi.crypto.sign.Signature unblind(RSAPublicKey pub, BlindSignature
bs, RSABlindingFactor bf) throws
        Exception
    {
        org.logi.crypto.keys.RSAPublicKey rsaPub = new
org.logi.crypto.keys.RSAPublicKey(pub.getPublicExponent(),

pub.getModulus());
        return rsaPub.unBlind(bs, bf);
    }

    public static org.logi.crypto.sign.Signature sign(RSAPrivateKey priv, Fingerprint
hash) throws Exception
    {
        org.logi.crypto.keys.RSAPrivateKey rsaPriv = new
org.logi.crypto.keys.RSAPrivateKey(priv.getPrivateExponent(),

priv.getModulus());
        return rsaPriv.sign(hash);
    }

    public static org.logi.crypto.sign.Signature sign(RSAPrivateKey priv, Object data)
throws Exception
    {
        Fingerprint fp;
        if(!(data instanceof Fingerprint))
        {
            ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
            ObjectOutputStream out = new ObjectOutputStream(byteStream);
            out.writeObject(data);
            byte[] temp = byteStream.toByteArray();

            byteStream.close();
            out.close();

            fp = Fingerprint.create(temp, 0, temp.length, "SHA1");
        }
        else
            fp = (Fingerprint) data;
        return sign(priv, fp);
    }

    public static boolean verify(RSAPublicKey pub, org.logi.crypto.sign.Signature signed,
Fingerprint hash) throws
        Exception
    {
        org.logi.crypto.keys.RSAPublicKey rsaPub = new
org.logi.crypto.keys.RSAPublicKey(pub.getPublicExponent(),

pub.getModulus());
        return rsaPub.verify(signed, hash);
```

```
    }

    public static boolean verify(RSAPublicKey pub, org.logi.crypto.sign.Signature signed,
Object data) throws Exception
    {
        ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(byteStream);
        out.writeObject(data);
        byte[] temp = byteStream.toByteArray();

        byteStream.close();
        out.close();

        return verify(pub, signed, Fingerprint.create(temp, 0, temp.length, "SHA1"));
    }

    public static Fingerprint merge(Object obj, Fingerprint fp) throws Exception
    {
        ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(byteStream);
        out.writeObject(obj);
        byte[] hashed = fp.getBytes();
        byte[] hashedObj = SHA1(byteStream.toByteArray()).getBytes();
        byte[] result = new byte[hashed.length + hashedObj.length];

        for(int i = 0; i < hashed.length; i++)
        {
            result[i] = hashed[i];
        }

        for(int i = 0; i < hashedObj.length; i++)
        {
            result[hashed.length + i] = hashedObj[i];
        }

        return Fingerprint.create(result, 0, result.length, "SHA1");
    }

    private static void testBlindSign2() throws Exception
    {
        Certificate cert = FileHandler.importCertificate(new
File(Config.BLSCertificateName));
        RSAPublicKey publ = (RSAPublicKey) cert.getPublicKey();
        RSAPrivateKey priv = (RSAPrivateKey) FileHandler.readObject(new
File(Config.BLSPrivateKeyName));

        RSABlindingFactor bf = createBlindingFactor();

        Coin coin = new Coin(Config.coinValue, 122434, System.currentTimeMillis(),
Config.maxNumberOfUsages, null, null);

        BlindedCoin bcoin = coin.getBlindedCoin(publ, bf, cert);

        Debugger.debug(className, 3, "Ok? " + coin.signAndVerify(publ, priv, cert, bcoin,
bf));
    }

    private static void testBlindSign() throws Exception
    {

        //generateRSAkeys();

        RSAPublicKey publ = (RSAPublicKey) FileHandler.importCertificate(new
File(Config.BLSCertificateName)).
            getPublicKey();
        RSAPrivateKey priv = (RSAPrivateKey) FileHandler.readObject(new
File(Config.BLSPrivateKeyName));

        Coin ballot = new Coin(Config.coinValue, 122434, System.currentTimeMillis(),
Config.maxNumberOfUsages, null, null);
//        byte[] ballot = coin;
```

```
        RSABlindingFactor bf = createBlindingFactor();

        byte[] t = FileHandler.writeObject(ballot);
        BlindFingerprint bfp = blind(publ, bf, Fingerprint.create(t, 0, t.length,
"SHA1"));

        BlindSignature bs = blindSign(priv, bfp);

        org.logi.crypto.sign.Signature signed = unblind(publ, bs, bf);

        boolean ok = verify(publ, signed, ballot);
        Debugger.debug(className, 3, "Verifying blind signature (true)? = " + ok); //must
return true

        org.logi.crypto.sign.Signature signedNormal = sign(priv, ballot);
        ok = equals(signed.getBytes(), signedNormal.getBytes());
        Debugger.debug(className, 3, "Testing blind signature = signature (true)? = " +
ok); //must return true

        ballot.setType(Coin.C_TYPE); //corrupting data
        // ballot[0] = 4;

        boolean ok2 = verify(publ, signed, ballot);
        Debugger.debug(className, 3, "Testing blind signature functions completed
successful(false)? = " + ok2); //must return false

        org.logi.crypto.sign.Signature signed2 = sign(priv, ballot);
        boolean ok3 = verify(publ, signed2, ballot);
        Debugger.debug(className, 3, "Testing blind signature functions completed
successful(true)? = " + ok3); //must return true
    }

    public static boolean equals(byte[] a, byte[] b)
    {
        if(a.length != b.length)
            return false;

        for(int i = 0; i < a.length; i++)
            if(a[i] != b[i])
                return false;
        return true;
    }

    public static String hexDigit(byte x)
    {
        StringBuffer sb = new StringBuffer();
        char c;
        // First nibble
        c = (char) ((x >> 4) & 0xf);
        if(c > 9)
        {
            c = (char) ((c - 10) + 'a');
        }
        else
        {
            c = (char) (c + '0');
        }
        sb.append(c);
        // Second nibble
        c = (char) (x & 0xf);
        if(c > 9)
        {
            c = (char) ((c - 10) + 'a');
        }
        else
        {
            c = (char) (c + '0');
        }
        sb.append(c);
        return sb.toString();
    }
```

```
/*-------------------------------------------------*/
/** convert Byte Array to Hex String */
/*-------------------------------------------------*/
public static String byteToHexString(byte[] content)
{
    if(content != null)
    {
        StringBuffer sBuffer = new StringBuffer();
        int contentLength = content.length;
        for(int i = 0; i < contentLength; i++)
        {
            sBuffer.append(hexDigit(content[i]));
        }
        return sBuffer.toString();
    }
    else
    {
        return "";
    }
}

/*-------------------------------------------------*/
/** convert Byte Array to String */
/*-------------------------------------------------*/
public static String byteToString(byte[] content)
{
    if(content != null)
    {
        StringBuffer sBuffer = new StringBuffer();
        int contentLength = content.length;
        for(int i = 0; i < contentLength; i++)
        {
            sBuffer.append(content[i]);
        }
        return sBuffer.toString();
    }
    else
    {
        return "";
    }
}

/*-------------------------------------------------*/
/** convert BigInteger to byte[] */
/*-------------------------------------------------*/
public static byte[] getBytes(BigInteger big)
{
    byte[] bigBytes = big.toByteArray();
    if((big.bitLength() % 8) != 0)
    {
        return bigBytes;
    }
    else
    {
        byte[] smallerBytes = new byte[big.bitLength() / 8];
        System.arraycopy(bigBytes, 1, smallerBytes, 0,
                         smallerBytes.length);
        return smallerBytes;
    }
}

/*-------------------------------------------------*/
/** convert String Array to Byte Array */
/*-------------------------------------------------*/
public static byte[] stringToByte(String[] string)
{
    StringBuffer sb = new StringBuffer();
    for(int i = 0; i < string.length; i++)
    {
        sb.append(string[i].getBytes());
```

```
    }
    return sb.toString().getBytes();
}

/*-------------------------------------------------*/
/** Message blinden */
/*-------------------------------------------------*/
public static byte[] blindMsg(byte[] msg, RSAPublicKey pub, BigInteger k)
{
    // Exponent des Public Keys
    BigInteger pub_EXP = pub.getPublicExponent();
    // Modulo des Public Keys
    BigInteger pub_MOD = pub.getModulus();
    // Zufallszahl k mit Public Key potenzieren
    BigInteger kP = k.modPow(pub_EXP, pub_MOD);
    // Message in BigInteger umwandeln
    BigInteger m = new BigInteger(1, msg);
    // Message m mit kP mulitplizieren
    BigInteger mKP = m.multiply(kP).mod(pub_MOD);
    return getBytes(mKP);
}

/*-------------------------------------------------*/
/** Message unblinden */
/*-------------------------------------------------*/
public static byte[] unblindMsg(byte[] msg, RSAPublicKey pub, BigInteger k)
{
    // Modulo des Public Keys
    BigInteger pub_MOD = pub.getModulus();
    // Message in BigInteger umwandeln
    BigInteger bM = new BigInteger(1, msg);
    // Unblind Message
    BigInteger m = bM.multiply(k.modInverse(pub_MOD)).mod(pub_MOD);
    return getBytes(m);
}

private static byte[] concat(byte[] a, byte[] b)
{
    byte[] result = new byte[a.length + b.length];

    int j = 0;
    for(int i = 0; i < b.length; i++)
    {
        result[j] = b[i];
        j++;
    }
    for(int i = 0; i < a.length; i++)
    {
        result[j] = a[i];

        j++;
    }
    return result;

}

private static byte[] getSection(byte[] a, int from, int to)
{
    byte[] result = new byte[to - from];
    for(int i = from; i < to; i++)
    {
        result[i - from] = a[i];
    }
    return result;
}

private static void print(byte[] b)
{
    for(int i = 0; i < b.length; i++)
        System.out.print(b[i]);
```

```
        System.out.println();
    }

    private static void testRSA() throws Exception
    {
        System.out.println("\nTest RSA...");
        SecretKey sessionKey = generateAESKey();
        Key publicKey = open("distributer_RSA_public");

        Debugger.debug(className, 3, "Key before: " + sessionKey);

        CryptoPacket encrypted = encryptRSA(sessionKey, publicKey);
        Debugger.debug(className, 3,
                       "Key after : " + sessionKey.equals(decryptRSA(encrypted,
open("distributer_RSA_private"))));
        Debugger.debug(className, 3, "\nTest RSA Done.\n");
    }

    private static void testAES() throws Exception
    {
        SecretKey key = generateAESKey();

        NetPacket np = new LogonPacket("Kurt", "password1");

        System.out.println("Encrypting...");
        CryptoPacket cp = encryptAES(np, key);
        System.out.println("Decrypting...");
        NetPacket np2 = decryptAES(cp, key);

        System.out.println("Passed test?: " + np.equals((LogonPacket) np2));
        System.out.println("Pack 1: " + np);
        System.out.println("Pack 2: " + np2);
    }

    public void testSafe() throws Exception
    {
        System.out.println("\nTest AES...");
        SecretKey sessionKey = generateAESKey();
        Coin vc1 = new Coin(3.3, 1243432, System.currentTimeMillis(), 10, null, null);
        Coin vc2 = new Coin(3.3, 12432, System.currentTimeMillis(), 10, null, null);
        Coin vc3 = new Coin(3.3, 12434, System.currentTimeMillis(), 10, null, null);
        Coin vc4 = new Coin(3.3, 1232, System.currentTimeMillis(), 10, null, null);

        Vector box = new Vector();
        box.addElement(vc1);
        box.addElement(vc2);
        box.addElement(vc3);
        box.addElement(vc4);

        ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(byteStream);
        out.writeObject(box);

    }

    public static CryptoPacket encryptRSA(SecretKey sessionKey, Key key) throws Exception
    {
        return new CryptoPacket(encryptRSA(sessionKey.getEncoded(), key));
    }

    public static byte[] encryptRSA(byte[] data, Key key) throws Exception
    {
        Cipher encryptCipher = Cipher.getInstance("RSA/ECB/PKCS1Padding", "BC");
        encryptCipher.init(Cipher.ENCRYPT_MODE, key);
        return encryptCipher.doFinal(data);
    }

    public static SecretKey decryptRSA(CryptoPacket cp, Key key) throws Exception
    {
        return(SecretKey)new SecretKeySpec(decryptRSA(cp.data, key), "AES");
    }
```

```
    public static byte[] decryptRSA(byte[] data, Key key) throws Exception
    {
        Cipher decryptCipher = Cipher.getInstance("RSA/ECB/PKCS1Padding", "BC");
        decryptCipher.init(Cipher.DECRYPT_MODE, key);
        return decryptCipher.doFinal(data);
    }

    public static byte[] encryptAES(Object pack, SecretKey key) throws Exception
    {
        ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(byteStream);
        out.writeObject(pack);

        return encryptAES(byteStream.toByteArray(), key);
    }

    public static byte[] encryptAES(byte[] data, SecretKey key) throws Exception
    {
        SecretKeySpec skeySpec = new SecretKeySpec(key.getEncoded(), "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec, new IvParameterSpec(iv), sec_rand);

        return cipher.doFinal(data);
    }

    public static CryptoPacket encryptAES(NetPacket pack, SecretKey key) throws Exception
    {
        ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(byteStream);
        out.writeObject(pack);

        return new CryptoPacket(encryptAES(byteStream.toByteArray(), key));
    }

    public static byte[] decryptAES(byte[] data, SecretKey key) throws Exception
    {
        SecretKeySpec skeySpec = new SecretKeySpec(key.getEncoded(), "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
        cipher.init(Cipher.DECRYPT_MODE, skeySpec, new IvParameterSpec(iv));
        return cipher.doFinal(data);
    }

    public static NetPacket decryptAES(CryptoPacket pack, SecretKey key) throws Exception
    {
        return(NetPacket) byteArrayToObject(decryptAES(pack.data, key));
    }

    public static Object byteArrayToObject(byte[] data) throws Exception
    {
        ByteArrayInputStream bs = new ByteArrayInputStream(data);
        ObjectInputStream in = new ObjectInputStream(bs);
        Object o = (Object) in.readObject();
        bs.close();
        in.close();
        return o;
    }

    private static void generateCertificates(String[] aliases, int keySize) throws
Exception
    {
        for(int i = 0; i < aliases.length; i++)
        {
            generateCertificate(aliases[i], keySize);
            System.out.println("Sleeping");
            //Thread.sleep(15000);
            System.out.println("Done sleeping");
        }
    }

    private static void extractPrivatekey() throws Exception
```

```
    {
        java.security.cert.Certificate cert2 = FileHandler.importCertificate(new
File("d:\\CA\\cacert.pem"));
        KeyFactory kf = KeyFactory.getInstance("RSA");
        PKCS8EncodedKeySpec pkcs = new PKCS8EncodedKeySpec(FileHandler.readBytes(new
File("d:\\CA\\cakey.key")));
        System.out.println(pkcs.getFormat());
        PrivateKey pk = kf.generatePrivate(pkcs);

        KeyStore ks = KeyStore.getInstance("JKS");
        //ks.load(new FileInputStream(new File(Config.OpenSSLpath + Config.name +
".keystore")),
        //                "spammail".toCharArray());
        ks.load(null,"spammail".toCharArray());

        ks.setKeyEntry("RS", pk, "spammail".toCharArray(), new
java.security.cert.Certificate[]
                       {cert2});

        ks.store(new FileOutputStream(new File(Config.OpenSSLpath + Config.name +
".keystore")), "spammail".toCharArray());

        FileHandler.writeObject((RSAPrivateKey) pk, new File(Config.OpenSSLpath +
"RS_private.key"));
    }

    private static void generateCertificate(String alias, int keySize) throws Exception
    {
        generateKeyPair(alias, Config.OpenSSLpath + Config.name + ".keystore", "rsa",
keySize, "spammail",
                        alias + "_password",
                        new String[]
                        {alias, "", "", "", "", "", ""});

        PrivateKey privKey = getKey(Config.OpenSSLpath + Config.name + ".keystore",
alias, "spammail",
                                    alias + "_password");

        FileHandler.writeObject(privKey, new File(Config.OpenSSLpath + alias +
"_private.key"));

        byte[] data = generateCertRequest(alias, Config.OpenSSLpath + Config.name +
".keystore", "spammail",
                                          alias + "_password");
        System.out.println(data.length);
        String filesep = System.getProperty("file.separator");

        FileOutputStream fos = new FileOutputStream(new File(Config.OpenSSLpath + alias +
"_certificaterequest.csr"));
        fos.write(data);
        fos.close();
        Runtime rt = Runtime.getRuntime();
        String command = "openssl ca -in " + Config.OpenSSLpath + alias +
"_certificaterequest.csr -out "
            + Config.OpenSSLpath
            + alias + "_certificate.crt -notext -batch -key spammail";
        Process proc = rt.exec(command);
        File generatedCert = new File(Config.OpenSSLpath + alias + "_certificate.crt");

        int checks = 0;
        while(!generatedCert.exists() || generatedCert.length()==0 ||
!generatedCert.canRead())
        {
            if(checks >= 10)
                throw new Exception("Error generating certificate!");

            Thread.sleep(300);
            Debugger.debug(Cryptotools.class, 3, "Waiting for certificate to be
generated");
            checks++;
        }
```

```
        Debugger.debug(Cryptotools.class, 3, "Certificate generated");

        Debugger.debug(Cryptotools.class, 3, "Command: " + command);

    }

    public static boolean generateKeyPair(String alias, String keystorefilename, String
algorithm, int keysize,
                                          String storepass, String keypass, String[]
data) throws Exception
    {
        //data = {firstname,lastname,orgUnit,orgName,locality,state,country}
        Runtime rt = Runtime.getRuntime();
        String cmd = "keytool -genkey -alias " + alias + " -keystore " + keystorefilename
+ " -keyalg " + algorithm
            + " -keysize " + keysize + " -dname \"CN=" + data[0] + " " + data[1] + ",
OU=" + data[2] + ", O=" + data[3]
            + ", L=" + data[4] + ", S=" + data[5] + ", C=" + data[6] + "\" -storepass " +
storepass + " -keypass "
            + keypass;
        Debugger.debug(Cryptotools.class, 3, "Command: " + cmd);
        Process proc = rt.exec(cmd);
        proc.waitFor();

        KeyStore ks = KeyStore.getInstance("JKS");
        File file = new File(keystorefilename);
        ks.load(new FileInputStream(file), storepass.toCharArray()); // must be changed
to "correct" path

        return ks.containsAlias(alias);
    }

    public static PrivateKey getKey(String keystorefilepath, String alias, String
keystorepass, String keypass) throws
        Exception
    {
        KeyStore ks = KeyStore.getInstance("JKS");
        ks.load(new FileInputStream(new File(keystorefilepath)),
keystorepass.toCharArray());
        return(PrivateKey) ks.getKey(alias, keypass.toCharArray());
    }

    public static byte[] generateCertRequest(String alias, String keystorefilename,
String keystorepass, String keypass) throws
        Exception
    {

        String cmd = "keytool -certreq -alias " + alias + " -keystore " +
keystorefilename + " -storepass "
            + keystorepass + " -keypass " + keypass + " -file " + alias + ".csr";
        Runtime rt = Runtime.getRuntime();
        Process proc = rt.exec(cmd);
        proc.waitFor();
        File file = new File(alias + ".csr");
        return FileHandler.readBytes(file);
    }

    public static SecretKey generateAESKey()
    {
        try
        {

            KeyGenerator kgen = KeyGenerator.getInstance("AES");
            kgen.init(encryptedKeySize); // 192 and 256 bits may not be available
            // Generate the secret key specs.
            return kgen.generateKey();
        }
        catch(Exception e)
        {
            System.out.println("Exception in generateKey: " + e);
```

```
        }
        return null;
    }

    public static void generateRSAkeys() throws Exception
    {
        System.out.print("generating....");
        KeyPairGenerator kg = KeyPairGenerator.getInstance("RSA", "BC");
        kg.initialize(2048);
        KeyPair kp = kg.generateKeyPair();
        RSAPublicKey publ = (RSAPublicKey) kp.getPublic();
        RSAPrivateKey priv = (RSAPrivateKey) kp.getPrivate();
        save("RSA_public_2048", publ);
        save("RSA_private_2048", priv);
        System.out.println("done.");
    }

    public static byte[] signRSA(Object data, PrivateKey key) throws Exception
    {
        ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(byteStream);
        out.writeObject(data);
        byte[] temp = byteStream.toByteArray();

        byteStream.close();
        out.close();

        return signRSA(temp, key);
    }

    public static byte[] signRSA(byte[] data, PrivateKey key) throws Exception
    {
        Signature rsa = Signature.getInstance("SHA1withRSA");
        rsa.initSign(key);
        rsa.update(data);

        return(rsa.sign());
    }

    public static boolean verifyRSA(Object data, byte[] signeddata,
java.security.cert.Certificate cert) throws
        Exception
    {
        ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(byteStream);
        out.writeObject(data);

        byte[] temp = byteStream.toByteArray();

        byteStream.close();
        out.close();

        return verifyRSA(temp, signeddata, cert);
    }

    public static boolean verifyRSA(byte[] data, byte[] signeddata,
java.security.cert.Certificate cert) throws
        Exception
    {
        java.security.Signature rsa = java.security.Signature.getInstance("SHA1withRSA");
        rsa.initVerify(cert);
        rsa.update(data);

        return rsa.verify(signeddata);

    }

    protected static Key openKey(String fileName)
    {

        try
```

```
        {
            FileInputStream iFile = new FileInputStream(fileName + ".key");
            ObjectInputStream iStream = new ObjectInputStream(iFile);

            return(Key) iStream.readObject();
        }
        catch(FileNotFoundException e)
        {
            Debugger.debug(className, 3, "The file could not be found." + e);
        }
        catch(ClassNotFoundException e)
        {
            Debugger.debug(className, 3, "An I/O Error accured - The file is corrupt." +
e);
        }
        catch(IOException e)
        {
            Debugger.debug(className, 3, "The file could not be opened. Please try
again." + e);
        }
        catch(Exception e)
        {
            Debugger.debug(className, 3, "The file could not be opened. Please try
again." + e);
        }
        return null;
    }

    public static Fingerprint SHA1(Object data) throws Exception
    {
        byte[] message;
        if(!(data instanceof Fingerprint))
        {
            ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
            ObjectOutputStream out = new ObjectOutputStream(byteStream);
            out.writeObject(data);
            message = byteStream.toByteArray();
        }
        else
            message = ((Fingerprint) data).getBytes();

        return Fingerprint.create(message, 0, message.length, "SHA1");
    }

    public static void save(String fileName, Key k)
    {
        try
        {
            FileOutputStream oFile = new FileOutputStream(fileName);
            ObjectOutputStream oStream = new ObjectOutputStream(oFile);

            oStream.writeObject(k);
        }

        catch(FileNotFoundException e)
        {
            Debugger.debug(className, 3, "Could not create file: " + e);
        }
        catch(IOException e)
        {
            Debugger.debug(className, 3, "The file could not be saved. Please try again."
+ e);
        }
        catch(Exception e)
        {
            Debugger.debug(className, 3, "The file could not be saved." + e);
        }
    }

    public static Key open(String fileName)
    {
```

```
        try
        {
            FileInputStream iFile = new FileInputStream(fileName);
            ObjectInputStream iStream = new ObjectInputStream(iFile);

            return(Key) iStream.readObject();
        }
        catch(FileNotFoundException e)
        {
            Debugger.debug(className, 3, "The file could not be found." + e);
        }
        catch(ClassNotFoundException e)
        {
            Debugger.debug(className, 3, "An I/O Error accured - The file is corrupt." +
e);
        }
        catch(IOException e)
        {
            Debugger.debug(className, 3, "The file could not be opened. Please try
again." + e);
        }
        catch(Exception e)
        {
            Debugger.debug(className, 3, "The file could not be opened. Please try
again." + e);
        }
        return null;
    }

    public static String asHex(byte buf[])
    {
        StringBuffer strbuf = new StringBuffer(buf.length * 2);
        int i;
        for(i = 0; i < buf.length; i++)
        {
            if(((int) buf[i] & 0xff) < 0x10)
                strbuf.append("0");
            strbuf.append(Long.toString((int) buf[i] & 0xff, 16));
        }
        return strbuf.toString();
    }

    // Used for debugging. The state of the control thread is printed to the terminal.
    private static String printState()
    {
        String s = "";
        /*            s += "\nisServer: " + isServer + ", isTempServer: " + isTempServer;
          s += "\nisWriter: " + isWriter + ", isTempWriter: " + isTempWriter;
          s += "\nisJoining: "+isJoining + ", isLeaving: "+isLeaving + ",
tryToLeaveAgain: "+tryToLeaveAgain;
          s += "\nserverError: "+serverError + ", writerError: "+writerError;
         s += "\nUsername = "+username + ", Groupname = "+groupname + ", Pass =
"+password + ", currentWriter = "+currentWriter;
          s += "\ntoDoList: ";
          if(toDoList != null)
           for(int i =0;i<toDoList.size();i++)
            s += toDoList.elementAt(i)+",";
          else
           s += "null";
          s += "\nQueue: "+groupQueue;
          s += "\nENDSTATE----------------------------------------";*/
        return s;
    }
}
```

10.5.4 Debugger

```
package SpamCash.Utilities;


import java.io.*;
import java.util.StringTokenizer;

public class Debugger
{
    //Used to print extra information in the money-status window
    public final static boolean debugging = false;

    /**
     * Debugging is done using the following level definitions
     * Levels for classlevels overrides package levels !
     *
     * Level 0: No output is generated
     *
     * Level 1: Information about behavior under normal operation is shown.
     *          Fatal exceptions is shown as well
     *
     * Level 2: Level 1 AND Information about behavior in general is shown
     *          All Exceptions and error situations is shown
     *
     * Level 3: Level 2 AND information about the state of the class is shown
     *          everytime an event/call happens.
     */

    private static String runningAplication;

    private final static int globalDebugLevel = 3;
    private final static boolean printLevel1 = true;
    private final static boolean printLevel2 = true;
    private final static boolean printLevel3 = true;
    private final static boolean printExceptions = true;
    private final static boolean printExceptionStacktrace = true;
    private final static boolean trim = false;

    private final static DebugDefinition[] debugDefinitions = new
        DebugDefinition[]
        {
        new DebugDefinition("SpamCash", 0),
        new DebugDefinition("Servers", 1),
        new DebugDefinition("Utilities", 0),
        new DebugDefinition("GUI", 0),
        new DebugDefinition("Control", 1),
        new DebugDefinition("IncomingMailProcessor", 1),
        new DebugDefinition("OutgoingMailProcessor", 1),
        new DebugDefinition("PackageQueue", 0),
        new DebugDefinition("IMAPProxy:NonBlokingProxyConnection", 0),
        new DebugDefinition("SMTPProxy:NonBlokingProxyConnection", 0),
        new DebugDefinition("AbstractMailProxy", 0),
        new DebugDefinition("AbstractNonBlockingMailProxy", 0),
        new DebugDefinition("IMAPProxy", 1),
        new DebugDefinition("SMTPProxy", 1),
        new DebugDefinition("POP3Proxy", 1),
        new DebugDefinition("session", 0),
        new DebugDefinition("Cryptotools", 0),
        new DebugDefinition("FileHandler", 0),
        new DebugDefinition("Currency", 0),
        new DebugDefinition("MimeTools", 0),
        new DebugDefinition("SafeHandler", 1),
        new DebugDefinition("Safe", 0)};

    private static boolean print = true;

    public static void setApplicationName(String name)
    {
        runningAplication = name;
        File f = new File(Config.logFolderPath);
```

```
        File files[] = f.listFiles();
        for(int i=0;i<files.length;i++)
            files[i].delete();
    }

    public static String debug(Class theclass, int level, String message)
    {
        String className = theclass.toString();
        return debug(className, level, message);
    }

    public static String debug(Class theclass, int level, String message, String
longMessage)
    {
        String className = theclass.toString();
        return debug(className, level, message + trim(longMessage));
    }

    public static String debug(String className, int level, String message, String
longMessage)
    {

        return debug(className, level, message + trim(longMessage));
    }

    public static String debug(String className, int level, String message)
    {
        if(globalDebugLevel == 0)
            return "";
        String out = "";
        out += getClassName(className) + " (" + level + "): " + trim(message);
        if(!out.endsWith("\n"))
            out += "\n";

        if(level < 2)
            log(out,runningAplication+"_level_1");
        if(level < 3)
            log(out,runningAplication+"_level_2");
        log(out,runningAplication+"_level_3");

        if(level > globalDebugLevel || level > getLevel(className))
            return "";

        if(print)
            print(out);
        return out;
    }

    public static void debug(Class theclass, int level, String message,
                             Throwable e)
    {
        if(globalDebugLevel == 0)
            return;
        boolean temp = print;
        print = false;
        String out = debug(theclass, level, message);
        print = temp;

        if(level == 1 && printLevel1 || level == 2 && printLevel2 ||
           level == 3 && printLevel3)
        {
            if(printExceptions)
            {
                out += e.getClass() + ": "+e.getLocalizedMessage()+"\n";
            }
            if(printExceptionStacktrace)
            {
                StackTraceElement[] se = e.getStackTrace();
                for(int i=0;i < se.length;i++)
                {
                    out += "\tat " +se[i] + "\n";
```

```
                }
            }
        }
        if(print)
            print(out);
    }

    private static int getLevel(String className)
    {
        String[] temp;
        String nextName = "";

        //Extract classnames and packagenames
        StringTokenizer st = new StringTokenizer(className, ". ");

        while(st.hasMoreTokens())
        {
            temp = new String[st.countTokens()];
            for(int j = 0; j < temp.length; j++)
                temp[j] = st.nextToken();

            for(int j = temp.length - 1; j >= 0; j--)
            {
                nextName = temp[j];
                for(int i = 0; i < debugDefinitions.length; i++)
                {
                    if(debugDefinitions[i].className.equals(nextName))
                    {
                        //Name found.
                        return debugDefinitions[i].debuglevel;
                    }
                }
            }
        }
        // Neither packages nor classname found in the DefinitionsList
        return -1;
    }

// Searches for the name in String of the form:
// SpamCash.Utilities.Debugger.
    private static String getClassName(String name)
    {
        //System.out.println("NAME: "+name);
        StringTokenizer st = new StringTokenizer(name, ". ");
        String[] temp = new String[st.countTokens()];
        for(int j = 0; j < temp.length; j++)
            temp[j] = st.nextToken();

            //System.out.println("Result: "+temp[temp.length - 1]);
        return temp[temp.length - 1];
    }

    private static String trim(String s)
    {
        if(!trim)
            return s;
        int maxLength = 90;
        String newline = System.getProperty("line.separator");
        String s2 = "";

        StringTokenizer st = new StringTokenizer(s, newline);
        s2 = st.nextToken();

        if(s2.length() < maxLength)
            return s2;
        else
            return s2.substring(0, maxLength - 3) + "...";

    }

    private static void print(String s)
```

```
    {
        //Make the normal log-file
        log(s,runningAplication);
        System.out.print(s);
    }

    private static void log(String m,String filename)
    {
        //Husk filendelsen .log
        File f = new File(Config.logFolderPath + filename + ".log");
        try
        {
            BufferedWriter writer = new BufferedWriter(new FileWriter(f,true));
            writer.write(m);
            writer.close();
        }
        catch(IOException ex)
        {
            System.out.println("Debugger: Error while trying to log to "+filename);
        }
    }

    private static class DebugDefinition
    {
        public String className;
        public int debuglevel;

        public DebugDefinition(String className, int debuglevel)
        {
            this.className = className;
            this.debuglevel = debuglevel;
        }
    }
}
```

10.5.5 FileHandler

```
package SpamCash.Utilities;


import java.io.*;
import java.security.cert.CertificateEncodingException;
import java.nio.charset.Charset;
import java.security.cert.Certificate;
import java.security.cert.CertificateFactory;
import java.security.cert.CertificateException;
import SpamCash.Utilities.Debugger;

public class FileHandler
{
    public static byte[] readBytes(File file) throws IOException
    {
        InputStream is = new FileInputStream(file);

        // Get the size of the file
        long length = file.length();

        // Create the byte array to hold the data
        byte[] bytes = new byte[(int) length];

        // Read in the bytes
        int offset = 0;
        int numRead = 0;
        while(offset < bytes.length

              && (numRead = is.read(bytes, offset, bytes.length - offset)) >= 0)
        {
            offset += numRead;
        }
```

```
        // Ensure all the bytes have been read in
        if(offset < bytes.length)
        {
            throw new IOException("Could not completely read file " + file.getName());
        }
        Debugger.debug(FileHandler.class,1,"Read "+offset+" bytes from file:
"+file.getAbsolutePath());
        // Close the input stream and return bytes
        is.close();

        return bytes;
    }

    public static void writeObject(Object data,File file) throws Exception
    {
        FileOutputStream outStream = new FileOutputStream(file);
        ObjectOutputStream out = new ObjectOutputStream(outStream);
        out.writeObject(data);

        outStream.close();
        out.close();
    }

    public static Object readObject(File file) throws Exception
    {
        FileInputStream stream = new FileInputStream(file);
        ObjectInputStream in = new ObjectInputStream(stream);
        Object o = in.readObject();

        stream.close();
        in.close();
        return o;
    }

    public static byte[] writeObject(Object data) throws Exception
    {
        ByteArrayOutputStream outStream = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(outStream);
        out.writeObject(data);

        byte[] result = outStream.toByteArray();
        outStream.close();
        out.close();
        return result;
    }

    public static Object readObject(byte[] source) throws Exception
    {
        ByteArrayInputStream stream = new ByteArrayInputStream(source);
        ObjectInputStream in = new ObjectInputStream(stream);
        Object o = in.readObject();

        stream.close();
        in.close();
        return o;
    }


    public static void writeBytes(byte[] bytes, File file) throws IOException
    {
        OutputStream os = new FileOutputStream(file);
        os.write(bytes);
        os.close();
    }

    // This method reads a certificate to a file. The certificate can be either
    // binary or base64 encoded.
    public static Certificate importCertificate(File file) throws FileNotFoundException,
CertificateException,
        IOException
```

```
    {
        FileInputStream is = new FileInputStream(file);

        CertificateFactory cf = CertificateFactory.getInstance("X.509");
        java.security.cert.Certificate cert = cf.generateCertificate(is);
        is.close();
        return cert;
    }

    // This method writes a certificate to a file. If binary is false, the
    // certificate is base64 encoded.
    public static void export(Certificate cert, File file) throws
CertificateEncodingException,
        FileNotFoundException, IOException
    {
        // Get the encoded form which is suitable for exporting
        byte[] buf = cert.getEncoded();

        FileOutputStream os = new FileOutputStream(file);

        // Write in text form
        Writer wr = new OutputStreamWriter(os, Charset.forName("UTF-8"));
        wr.write("-----BEGIN CERTIFICATE-----\n");
        wr.write(new sun.misc.BASE64Encoder().encode(buf));
        wr.write("\n-----END CERTIFICATE-----\n");
        wr.flush();

        os.close();
    }

    public static String findNewFile(String directory)
    {
        int messageNameCounter = 1;
        String path;
        File file;
        do
        {
            path = directory + File.separator + messageNameCounter
                + ".msg";
            file = new File(path);
            messageNameCounter++;
        }
        while(file.exists());

        return path;
    }

    /**
     * Deletes all files in the requested directory.
     * This could be mailfiles from a previus session
     * which was not deleted then because of an error.
     * @return true if directory is empty, false if the operation could not be completed.
     */
    public static boolean clearDirectory(String directoryName) throws IOException
    {
        File dir = new File(directoryName);
        if(!dir.isDirectory())
            throw new IOException("Cannot clear directory. Specified name " +
directoryName + " is not a directory");

        File[] files = dir.listFiles();
        for(int i = 0; i < files.length; i++)
            files[i].delete();
        return true;
    }
}
```

10.5.6 MailInformation

```
package SpamCash.Utilities;


import SpamCash.Currency.Coin;
import javax.mail.internet.MimeMessage;
import javax.mail.MessagingException;
import javax.mail.internet.MimeMultipart;
import javax.mail.Multipart;
import javax.mail.BodyPart;
import java.io.IOException;
import SpamCash.Utilities.Debugger;
import javax.mail.Address;
import javax.mail.Message;

public class MailInformation implements java.io.Serializable
{
    public Coin c;
    public String info,oneLineInfo;
    public Address sender,to;

    public MailInformation(Coin c,MimeMessage msg) throws MessagingException
    {
        this.c = c;
        this.sender = msg.getFrom()[0];

        String s = "";
        s += "   Sent: " + msg.getSentDate() +" Size: "+ msg.getSize() + "\n\n";
        s += "From: " + msg.getFrom()[0]+ "\n";
        Address[] to = msg.getRecipients(Message.RecipientType.TO);
        this.to = to[0];
        s += "To: " + to[0] + "\n";

        //Get cc-receivers...
        String senders = "";
        for(int i=1;i<to.length;i++)
        {
            senders += to[i];
            if(i < to.length -1)
                senders += ", ";
        }
        if( ! senders.equals(""))
            s += "Cc: " + senders + "\n";
        s += "Subject: " + msg.getSubject() + "\n";

        s += "Message: \n" + getText(msg);

        this.info = s;

        this.oneLineInfo = ""+msg.getSentDate()+" - "+msg.getSubject();
    }

    private String getText(MimeMessage msg)
    {
        try
        {
            Object object = msg.getContent();
            return getText(object);
        }
        catch(Exception e)
        {
            Debugger.debug(getClass(),3,"Error: ",e);
            return "Could not extract message contents.";

        }
    }

    private String getText(Object object) throws Exception
    {
```

```
        if(object instanceof Multipart)
        {
            Multipart mp = (Multipart) object;
            for(int i = 0, n = mp.getCount(); i < n; i++)
            {
                if(mp.getBodyPart(i).getContentType().indexOf("text/plain") != -1)
                    return (String) mp.getBodyPart(i).getContent();
            }
        }
        else if(object instanceof BodyPart)
        {
            BodyPart bp = (BodyPart) object;
            if(bp.getContentType().indexOf("text/plain") != -1)
                return (String) bp.getContent();
        }
        else
        // Just a String...
        {
            return (String) object;
        }
        throw new Exception("Error");
    }

    public String showContents()
    {
        return info;
    }

    public String toString()
    {
        return this.oneLineInfo;
    }
}
```

10.5.7 MimeTools

```
package SpamCash.Utilities;


import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
import java.io.*;
import javax.activation.*;
import java.lang.Math;

import SpamCash.Currency.*;
import SpamCash.Utilities.Debugger;

public class MimeTools
{
    private final static Class className = MimeTools.class;

    public static void main(String[] sdfh)
    {
        try
        {
            //Testing addAttachmentAsHeader...
            Debugger.debug(className,3,"Testing addAttachmentAsHeader...");
            //Coins to be attached are declared ...
            Coin[] coin = new Coin[]
                {new Coin(3.3,1243432,System.currentTimeMillis(),10,null,null),
                new Coin(3.3,1243432,System.currentTimeMillis(),10,null,null)};

            MimeMessage msg = readMimeFile("data/testMessages/testMessageHTML.msg");
            addAttachmentHeader(msg, coin);
            writeMimeFile(msg,
"data/testMessages/testResult/HeaderCoinAdded_to_testMessageHTML.msg");
            coin = (Coin[])getCoinsHeader(msg);
            removeAttachmentHeader(msg);
```

```
            writeMimeFile(msg,
"data/testMessages/testResult/HeaderCoinRemoved_from_testMessageHTML.msg");

            //printing information from coin[] to ensure that the Coin[] was succesfully
restored.

            //System.out.println(coin[0].getValue());
            //System.out.println(coin[1].getValue());
            //Debugger.debug(className,3,coin[0]+"");

            //Testing addAttachment... ------------------------------------------------
--------
            Debugger.debug(className,3,"Testing addAttachment...");
            //Adding to an HTML message
            Debugger.debug(className,3,"Adding to an HTML message...");
            test("testMessageHTML", false);

            //Adding to an HTML message with attachment
            Debugger.debug(className,3,"Adding to an HTML message with attachment");
            test("testMessageHTML", true);

            //Adding to an Plain text message
            Debugger.debug(className,3,"Adding to a Plain text message");
            test("testMessagePlain", false);

            //Adding to an Plain text message with attachment
            Debugger.debug(className,3,"Adding to a Plain text message with attachment");
            test("testMessagePlain", true);

            //Testing remove attachment...
            testAddRemoveCoin("testMessageHTML");
            testAddRemoveCoin("testMessageHTMLandAttachment");
            testAddRemoveCoin("testMessagePlain");
            testAddRemoveCoin("testMessagePlainandAttachment");

            Debugger.debug(className,3,"Test complete.");
        }
        catch(Exception e)
        {
            Debugger.debug(className,3,"Error ... \n"+e,e);
        }
    }

    public static void test(String name, boolean attachment) throws Exception
    {
        MimeMessage msg = readMimeFile("data/testMessages/" + name + (attachment ?
"andAttachment" : "") + ".msg");
        msg = addAttachment(msg, "data/testMessages/" + name + (attachment ?
"andAttachment" : "") + ".msg");
        writeMimeFile(msg,
                      "data/testMessages/testResult/" + name + (attachment ?
"andAttachment" : "") + "_testedAttach"
                      + ".msg");
    }

    private static void testAddRemoveCoin(String filename) throws Exception
    {
        Debugger.debug(className,3,"Testing add/remove coin to " + filename + "...");
        Coin[] coins = new Coin[]
            {new Coin(3.3,1243432,System.currentTimeMillis(),10,null,null),
            new Coin(3.3,1243432,System.currentTimeMillis(),10,null,null)};
        MimeMessage msg = readMimeFile("data/testMessages/" + filename + ".msg");
        MimeMessage orig = msg;
        //Debugger.debug(className,3,"Coin 1 before: " + coins[0]);
        //Debugger.debug(className,3,"Coin 2 before: " + coins[1]);
        msg = addAttachments(msg, coins);
        writeMimeFile(msg, "data/testMessages/testResult/" + filename +
"_tested_attach_coin.msg");
        Coin[] coin2 = (Coin[])getCoins(msg);
        MimeMessage msg2 = removeAttachment(msg);
        //Debugger.debug(className,3,"Coin 1 after: " + coin2[0]);
```

```
        //Debugger.debug(className,3,"Coin 2 after: " + coin2[1]);
        Debugger.debug(className,3,"Restored message? : " + orig.equals(msg2));
        writeMimeFile(msg2, "data/testMessages/testResult/" + filename + "2.msg");
    }

    public static MimeMessage addAttachmentHeader(MimeMessage message, Object o) throws
Exception
    {
        String encodedcoins = encodeCoins(o);

        int numberofheaders = (int)(encodedcoins.length()/Config.maxHeaderSize)+1;

        for (int i=0;i<numberofheaders;i++)

message.addHeader(Config.MimeDescriptionOfCoinAttachment+i,encodedcoins.substring(i*Confi
g.maxHeaderSize,Math.min((i+1)*Config.maxHeaderSize,encodedcoins.length())));

        message.saveChanges();

        return message;
    }

    public static MimeMessage removeAttachmentHeader(MimeMessage message) throws
MessagingException
    {
        for(Enumeration e = message.getAllHeaders(); e.hasMoreElements(); )
        {
            Header temp = (Header) e.nextElement();
            if(temp.getName().startsWith(Config.MimeDescriptionOfCoinAttachment))
            {
                message.removeHeader(temp.getName());
            }
        }

        message.saveChanges();
        return message;

    }

    public static Object getCoinsHeader(MimeMessage message) throws Exception
    {
        Enumeration e = message.getAllHeaders();
        boolean foundcoin = false;
        String result = "";
        Header temp;

        while(e.hasMoreElements())
        {
            temp = (Header) e.nextElement();

            if(temp.getName().startsWith(Config.MimeDescriptionOfCoinAttachment))
            {
                foundcoin = true;
                result += temp.getValue();
            }
        }
        if (!foundcoin)
        throw new Exception("No coin found on message.");

        return decodeCoins(result);
    }

    public static MimeMessage addAttachment(MimeMessage message, String filename) throws
Exception
    {
        //Make a copy of the message
        MimeMessage newmessage = copyHeaders(message);
        MimeMultipart newmp = copyPartsToMultiPart(message.getContent(), true);

        // Add the attachment
        newmp = addAttachmentToMultiPart(filename, newmp);
```

```
        //Save changes
        newmessage.setContent(newmp);
        newmessage.saveChanges();

        return newmessage;
    }

    public static MimeMessage addAttachments(MimeMessage message, Coin[] coins) throws
Exception
    {
        //Make a copy of the message
        MimeMessage newmessage = copyHeaders(message);
        MimeMultipart newmp = copyPartsToMultiPart(message.getContent(), true);

        // Add the attachment
        newmp = addAttachmentsToMultiPart(coins, newmp);

        //Save changes
        newmessage.setContent(newmp);
        newmessage.saveChanges();

        return newmessage;
    }

    public static MimeMessage removeAttachment(MimeMessage message) throws Exception
    {
        //Debugger.debug(className,3,"REMOVING...");
        //Make a copy of the message, and dont include the coin !
        MimeMessage newmessage = copyHeaders(message);

        Multipart mp = (Multipart) message.getContent();
        Object np = copyParts(mp.getBodyPart(0));

        if(np instanceof MimeMultipart)
        {
            MimeMultipart oldmp = (MimeMultipart) np;
            newmessage.setContent(oldmp, oldmp.getContentType());
        }
        else if(np instanceof MimeBodyPart)
        {
            MimeBodyPart bp = (MimeBodyPart) np;
            newmessage.setContent(bp, bp.getContentType());
        }
        else
        // Just a String...
        {
            newmessage.setText((String) np);
        }

        //Save changes
        newmessage.saveChanges();

        return newmessage;
    }

    private static Object copyParts(Object content) throws Exception
    {
        if(content instanceof MimeMultipart)
        {
            MimeMultipart newmp = new MimeMultipart();
            MimeMultipart oldmp = (MimeMultipart) content;
            Object temp;
            MimeBodyPart temp1;
            for(int i = 0, n = oldmp.getCount(); i < n; i++)
            {
                temp1 = new MimeBodyPart();
                temp = oldmp.getBodyPart(i);

                if(temp instanceof MimeBodyPart)
                {
```

```
                    newmp.addBodyPart((MimeBodyPart) copyParts(temp));
                }
                else
                {
                    temp1.setContent((MimeMultipart) copyParts(temp));
                    newmp.addBodyPart(temp1);
                }
            }
            return newmp;

        }
        else
            return((MimeBodyPart) content).getContent();
    }

    public static Object getCoins(MimeMessage message) throws Exception
    {

        Object part = message.getContent();

        // If it is NOT a multipart in 'multipart/mixed' mode the coin is nowhere to be
found
        if(!(part instanceof Multipart))
        {
            throw new MessagingException(
                "MimeReadWrite.getCoin(): Could not get the coin. Not a multipart message
!");
        }

        MimeBodyPart tempBodyPart = null;

        MimeMultipart mp = (MimeMultipart) part;
        tempBodyPart = (MimeBodyPart)
mp.getBodyPart(Config.MimeDescriptionOfCoinAttachment);

        if(tempBodyPart == null)
            throw new MessagingException(
                "MimeReadWrite.getCoin(): Could not get the coins. Checked all the parts
but did not find any coins !");

        ByteArrayOutputStream out = new ByteArrayOutputStream();
        tempBodyPart.writeTo(out);
        out.close();
        return decodeCoins(new String(out.toByteArray()));

    }

    private static MimeMessage copyHeaders(MimeMessage message) throws Exception
    {
        Properties props = new Properties();
        props.put("mail.smtp.host", Config.configuration.outgoingMailserver);

        Session session = Session.getInstance(props, null);

        // Create the message to return
        MimeMessage newmessage = new MimeMessage(session);

        //Copy headers..
        Enumeration enum = message.getAllHeaderLines();
        while(enum.hasMoreElements())
        {
            newmessage.addHeaderLine((String) enum.nextElement());
        }

        //TEST
        /*String h = "SPAMFORTUNE: ";
                 for(int i=0;i<1000;i++)
            h += "kjhksjhksjdfksjdfksjdfkjhsdfjw";
                 newmessage.addHeaderLine(h);
         */
```

```
        return newmessage;
    }

    private static MimeMultipart copyPartsToMultiPart(Object part, boolean includeAll)
throws Exception
    {
        MimeMultipart newmp = new MimeMultipart("mixed");
        BodyPart tempBodyPart = new MimeBodyPart();

        // If it is a multipart in 'multipart/mixed' mode
        // we just add en extra part containing the attachment
        /*        if(part instanceof Multipart && ((Multipart)
part).getContentType().equals("multipart/mixed"))
                {
                    // Copy the old parts...
                    Multipart oldmp = (Multipart) part;
                    for(int i = 0, n = oldmp.getCount(); i < n; i++)
                    {
                        tempBodyPart = oldmp.getBodyPart(i);
                        newmp.addBodyPart(tempBodyPart);
                    }
                }
                // It is a multipart but not in 'mixed'-mode
                // we put this multipart as bodypart 1
                // and the attachment as part 2
                // in a new 'mixed' multipart
                else */
        if(part instanceof Multipart)
        {
            //Add the old part
            tempBodyPart = new MimeBodyPart();
            tempBodyPart.setContent((Multipart) part);
            newmp.addBodyPart(tempBodyPart);
        }
        //It is just a bodypart with text.
        else
        {
            MimeBodyPart bp = new MimeBodyPart();
            bp.setText((String) part);
            newmp.addBodyPart(bp);
        }
        return newmp;
    }

    private static MimeMultipart addAttachmentToMultiPart(String filename, MimeMultipart
multipart) throws
        MessagingException
    {
        MimeBodyPart tempBodyPart = new MimeBodyPart();
        DataSource source = new FileDataSource(filename);
        tempBodyPart.setDataHandler(new DataHandler(source));
        tempBodyPart.setFileName(filename);
        multipart.addBodyPart(tempBodyPart);

        return multipart;
    }

    private static MimeMultipart addAttachmentsToMultiPart(Object[] coins, MimeMultipart
multipart) throws Exception
    {
        byte[] encoded = encodeCoins(coins).getBytes();

        ByteArrayInputStream in = new ByteArrayInputStream(encoded);
        MimeBodyPart bodyPart = new MimeBodyPart(in);
        bodyPart.setContentID(Config.MimeDescriptionOfCoinAttachment);
        multipart.addBodyPart(bodyPart);

        return multipart;
    }

    public static String encodeCoins(Object coins) throws Exception
```

```
    {
        ByteArrayOutputStream bOut = new ByteArrayOutputStream();
        ObjectOutputStream oOut = new ObjectOutputStream(bOut);

        oOut.writeObject(coins);

        byte[] out = bOut.toByteArray();

        bOut.close();
        oOut.close();

        //System.out.println("BEFORE: "+Cryptotools.byteToHexString(out));

        String coded = Base64.encodeBytes(out);
        return coded;
    }

    public static Object decodeCoins(String partBytes) throws Exception
    {
        byte[] decoded = Base64.decode(partBytes);

        //System.out.println("AFTER : \n"+Cryptotools.byteToHexString(decoded)+"
\n\nBytes: "+partBytes.length());

        ByteArrayInputStream bin = new ByteArrayInputStream(decoded);
        ObjectInputStream oin = new ObjectInputStream(bin);

        Object object = oin.readObject();

        bin.close();
        oin.close();
        return object;
    }

    private static Multipart addMessageStartToMultiPart(Multipart oldmp, String
extratext) throws Exception
    {
        // Use the same MIME-subtype for the new message
//             StringTokenizer s = new StringTokenizer(oldmp.getContentType(),"/;");
//             s.nextToken();
//             MimeMultipart newmp = new MimeMultipart(s.nextToken());

        BodyPart tempBodyPart = oldmp.getBodyPart(0);
        if(tempBodyPart.getContent() instanceof Multipart)
        {
            tempBodyPart.setContent(addMessageStartToMultiPart((Multipart)
tempBodyPart.getContent(), extratext));
        }
        else if(tempBodyPart.getContent() instanceof String)
        {
            tempBodyPart.setText(extratext + (String) tempBodyPart.getContent());
        }
        else
        {
            Debugger.debug(className,3,"CLASS: " + tempBodyPart.getContent().getClass());
            throw new Exception("Failed to add Message Start:\n");
        }
        return oldmp;
    }

    public static MimeMessage readMimeFile(String filename) throws Exception
    {
        return readMimeFile(new File(filename));
    }

    public static MimeMessage readMimeFile(File file) throws Exception
    {

        Properties props = new Properties();
        props.put("mail.smtp.host", "someHost");
```

```
        Session session = Session.getInstance(props, null);
        //session.setDebug(true);

        try
        {
            FileInputStream iFile = new FileInputStream(file);
            MimeMessage msg = new MimeMessage(session, iFile);
            iFile.close();
            return msg;
        }
        catch(IOException e)
        {
            Debugger.debug(className,3,"The file could not be read. Please try again." +
e);
            throw e;
        }
        catch(Exception e)
        {
            Debugger.debug(className,3,"The file could not be read. Please try again." +
e);
            throw e;
        }
    }

    public static void writeMimeFile(MimeMessage msg, String filename)
    {
        try
        {
            FileOutputStream oFile = new FileOutputStream(filename);
            //ObjectOutputStream oStream = new ObjectOutputStream(oFile);

            msg.writeTo(oFile);
            oFile.close();

        }
        catch(FileNotFoundException e)
        {
            Debugger.debug(className,3,"Could not create file" + e);
        }
        catch(IOException e)
        {
            Debugger.debug(className,3,"The file could not be saved. Please try again." +
e);
        }
        catch(Exception e)
        {
            Debugger.debug(className,3,"The file could not be saved." + e);
        }
    }
}
```

10.5.8 PackageQueue

```
package SpamCash.Utilities;


import java.util.Vector;
import SpamCash.Utilities.Debugger;
import SpamCash.Utilities.MimePacket;

public class PackageQueue extends Thread
{
    private boolean isWaiting;
    private Vector queue;

    public PackageQueue()
    {
        isWaiting = false;
        queue = new Vector();
```

```
    }

      public synchronized MimePacket getPacket()
      {
          isWaiting = true;
          while(queue.size() == 0)
          {
              try
              {
                  wait();
              }
              catch(Exception e)
              {
                  Debugger.debug(getClass(), 2, " Interruptedexception: ");
              }
          }
          isWaiting = false;
          return (MimePacket)queue.remove(0);
      }

      public synchronized void addPacket(MimePacket pack)
      {
          queue.add(pack);
          if(isWaiting)
              notify();
      }


}
```

10.5.9 ProxyConfiguration

```
package SpamCash.Utilities;


public class ProxyConfiguration implements java.io.Serializable
{
    public String outgoingMailserver,incomingMailserver;//,name,email,loginName,password;

    public ProxyConfiguration(String outgoingMailserver,String
incomingMailserver/*,String name,String email,String loginName,String password*/)
    {
        this.outgoingMailserver = outgoingMailserver;
        this.incomingMailserver = incomingMailserver;
        /*this.name = name;
        this.email = email;

        this.loginName = loginName;
        this.password = password;*/
    }
}
```

10.5.10 Session

```
package SpamCash.Utilities;


import javax.crypto.*;
import java.net.*;
import java.io.*;

import SpamCash.NetworkPackets.*;
import SpamCash.Utilities.Debugger;
import java.security.Key;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;

public class session implements Serializable
{
```

```
    // Tells wether this connection is established or not
    public boolean closed,useEncryption;

    private Socket socket;
    private InetAddress remoteAddress;
    private SecretKey secretKey;
    private ObjectOutputStream outputStream;
    private ObjectInputStream inputStream;

    private long expirationTime;

    public session(Socket socket, boolean server, Key RSAkey, boolean useEncryption)
throws Exception
    {
        try
        {
            this.useEncryption = useEncryption;
            this.socket = socket;
            remoteAddress = socket.getInetAddress();

            if(server)
            {
                inputStream = new ObjectInputStream(socket.getInputStream());
                outputStream = new ObjectOutputStream(socket.getOutputStream());
            }
            else
            {
                outputStream = new ObjectOutputStream(socket.getOutputStream());
                inputStream = new ObjectInputStream(socket.getInputStream());
            }

            closed = false;
            expirationTime = Config.expirationPeriod + System.currentTimeMillis();
            socket.setSoTimeout((int)Config.expirationPeriod);

            if(useEncryption)
            {
                Debugger.debug(getClass(), 3, "Establishing session with " +
remoteAddress + " using encryption.");

                if(server)
                {
                    CryptoPacket cp = (CryptoPacket) inputStream.readObject();
                    secretKey = Cryptotools.decryptRSA(cp, (RSAPrivateKey) RSAkey);
                    send(new StatusPacket(true, ""));
                }
                else
                {
                    secretKey = Cryptotools.generateAESKey();
                    outputStream.writeObject(Cryptotools.encryptRSA(secretKey,
(RSAPublicKey) RSAkey));
                    StatusPacket ack = (StatusPacket) receive();
                    if(!ack.ok)
                        throw new Exception("Error initializing connection");
                }
            }
            Debugger.debug(getClass(),3,"Session is established with: " + remoteAddress);
        }
        catch(IOException e)
        {
            //First packet was not an encrypted session-key, so we close connection and
return
            try
            {
                closed = true;
                socket.close();
            }
            catch(IOException ex)
            {
                Debugger.debug(getClass(),3,"Could not close connection to
"+remoteAddress);
```

```
            }
            Debugger.debug(getClass(),3,"Exception ",e);
            throw new Exception("Could not establish session with "+remoteAddress);
        }
    }

    public void send(NetPacket pack) throws Exception
    {
        //test if session has expired.
        if(expirationTime < System.currentTimeMillis())
        {
            Debugger.debug(getClass(),3,"Session expired. Expirationtime:
"+expirationTime+" Current time: "+System.currentTimeMillis());
            close();
        }

        try
        {
            if(useEncryption)
            {
                CryptoPacket cp = Cryptotools.encryptAES(pack, secretKey);
                outputStream.writeObject(cp);
            }
            else
                outputStream.writeObject(pack);
            Debugger.debug(getClass(),3,"Sent "+pack.toString());
        }
        catch(Exception ex)
        {
            Debugger.debug(getClass(),3,"Exception in send: ",ex);
            close();
            throw ex;
        }
    }

    public NetPacket receive() throws Exception
    {
        //test if session has expired.
        if(expirationTime < System.currentTimeMillis())
            close();

        //read packet from client
        try
        {
            NetPacket np;
            if(useEncryption)
            {
                CryptoPacket cp = (CryptoPacket) inputStream.readObject();

                np = Cryptotools.decryptAES(cp, secretKey);
            }
            else
            {
                np = (NetPacket) inputStream.readObject();
            }

            Debugger.debug(getClass(),3,"Received "+np.toString());
            return np;
        }
        //if client is sending something else than a Cryptopacket, close connection
immediately.
        catch(Exception e)
        {
            close();
            throw e;
        }
    }

    public void close()
    {
        closed = true;
```

```
        Debugger.debug(getClass(), 3, "Closing connection to " + remoteAddress);
        try
        {
            socket.close();
            return;
        }
        catch(IOException e)
        {
            try
            {
                socket.close();
            }
            catch(IOException ex)
            {
            }
            Debugger.debug(getClass(), 3, "Error closing connection to " +
remoteAddress);
        }
    }
}
```

10.5.11 Test

```
package SpamCash.Utilities;

import javax.mail.internet.*;
import javax.mail.*;
import java.util.Properties;
import java.util.Date;
import SpamCash.Currency.Coin;
import java.security.cert.Certificate;
import java.security.interfaces.RSAPrivateKey;
import org.logi.crypto.sign.Fingerprint;
import SpamCash.Currency.Transactionlist;
import SpamCash.Currency.UnencryptedTransactionlist;
import SpamCash.Currency.TransactionlistElement;
import SpamCash.Client.Safe.SafeHandler;
import java.util.Random;
import java.net.Socket;
import java.net.InetAddress;
import java.security.interfaces.RSAPublicKey;
import java.io.File;
import SpamCash.Servers.BLS.Blacklist;
import SpamCash.NetworkPackets.GetBlacklistPacket;
import SpamCash.NetworkPackets.BlacklistPacket;
import SpamCash.NetworkPackets.ReportCoinsPacket;

public class Test
{
    Certificate BLSCertificate;

    public Test()
    {
    }

    public static void main(String[] s)
    {
        Test t = new Test();
        //t.testBlacklistServer();
        t.sendMail(2,1000);
    }

    private void testBlacklistServer()
    {
        int threads = 200;

        try
        {
            BLSCertificate = FileHandler.importCertificate(new
File(Config.BLSCertificateName));
```

```
            SafeHandler safe = new SafeHandler();
            safe.openSafe(Config.defaultSafeKeyFilePath, Config.defaultLocalPassword);
            Coin[] coins = safe.getCoins(2);

            BlacklistThread bl;

            for(int i=0;i<threads;i++)
            {
                bl = new BlacklistThread(coins[0],coins[1],i);
                bl.start();
            }
        }
        catch(Exception ex)
        {
            System.out.println("Error reading coins..");
            ex.printStackTrace();
        }

    }

    private static void sendMail(int mails, int size)
    {
        try
        {
            Config.setProxyConfiguration(new ProxyConfiguration("localhost",
"localhost"));
            System.out.println("Generating messages to send....");
            long t = System.currentTimeMillis();
            MimeMessage[] m = generateMessages(mails, size);
            long time = System.currentTimeMillis() - t;
            System.out.println("Generation completed in: " + time);
            System.out.println("Sending...");
            t = System.currentTimeMillis();
            massMail(m, new InternetAddress("user1@127.0.0.1"), "localhost");
            time = System.currentTimeMillis() - t;
            System.out.println("Sending completed in: " + time);
            System.out.println("Time pr. message: " + time / mails);
        }
        catch(Exception ex)
        {
            System.out.println("Error...");
            ex.printStackTrace();
        }
    }

    private static MimeMessage[] generateMessages(int numberOfMails, int sizeInKilobytes)
throws Exception
    {
        MimeMessage[] messages = new MimeMessage[numberOfMails];

        Properties props = System.getProperties();
        props.put("mail.smtp.host", Config.configuration.outgoingMailserver);
        Session session = Session.getDefaultInstance(props, null);

        for(int i=0;i<messages.length;i++)
        {
            // -- Create a new message --
            messages[i] = new MimeMessage(session);

            messages[i].setSubject(Config.name + "Test mail " + i);
            messages[i].setText("...");
            messages[i] = MimeTools.addAttachment(messages[i],Config.mainDirectory +
System.getProperty("file.separator") + "testMessages" +
System.getProperty("file.separator") + sizeInKilobytes+ ".bin");
            messages[i].saveChanges();
        }
        return messages;

    }
```

```
    private static void massMail(MimeMessage[] messages, Address recipient, String
mailserver)
    {
        //Send a lot of mails...
        try
        {
            System.out.println("Mass-mailing to " + recipient + " started...");
            Properties props = System.getProperties();

            // -- Attaching to default Session, or we could start a new one --

            props.put("mail.smtp.host", mailserver);
            Session session = Session.getDefaultInstance(props, null);

            for(int i=0;i<messages.length;i++)
            {
                messages[i].setFrom(recipient);
                messages[i].setRecipients(Message.RecipientType.TO, new Address[]
                                  {recipient});

                Transport.send(messages[i]);
               System.out.println("Mail " + i + " / " + messages.length + " sent
succesfully to: " + recipient);
            }
            System.out.println("Mass-mailing to " + recipient + " completed.");
        }
        catch(Exception ex)
        {
            System.out.println("Error sending mail to: " + recipient);
            ex.printStackTrace();
        }

    }


    public class BlacklistThread extends Thread
    {
        private double getList = 0.985, report = 0.005, getWholeList = 0.01;
        private Blacklist blacklist;
        private session serverSession;
        private Coin a,b;

        public BlacklistThread(Coin a, Coin b, int index) throws Exception
        {
            this.a = a;
            this.b = b;
            blacklist = new Blacklist();
            System.out.println("Thread "+index+" started.");
        }

        public void run()
        {
            Random r = new Random();
            double rand = r.nextDouble();

            try
            {
                while(true)
                {
                    rand = r.nextDouble();
                    if(rand < getList)
                    {
                        //getList
                        updateBlacklist();
                    }
                    else if(rand < (getList + report))
                    {
                        //report
                        reportCoins(a, b);
                    }
                    else
```

```
                    {
                        blacklist = new Blacklist();
                        updateBlacklist();
                        //get whole list
                    }
                }
            }
            catch(Exception e)
            {
                System.out.println("Exception....");
                e.printStackTrace();
            }
        }

        private session createBLSSession() throws Exception
        {
            Debugger.debug(getClass(), 3,
                           "Creating new session with the BLS: " + Config.BLSAddress +
":" + Config.BLSPort);
            Socket socket = new Socket(InetAddress.getByName(Config.BLSAddress),
Config.BLSPort);
            return new session(socket, false, (RSAPublicKey)
BLSCertificate.getPublicKey(),false);
        }

        public void updateBlacklist()
        {
            Debugger.debug(getClass(), 1, "Updating blacklist....");

            try
            {
                serverSession = createBLSSession();
                serverSession.send(new GetBlacklistPacket(blacklist.elements.size()));
                BlacklistPacket responsePacket = (BlacklistPacket)
serverSession.receive();
                blacklist.merge(responsePacket.blacklist);
                serverSession.close();
                Debugger.debug(getClass(), 1, "Blacklist updated succesfully. New
list:\n" + blacklist);
            }
            catch(Exception ex1)
            {
                Debugger.debug(getClass(), 1, "Error updating blacklist.", ex1);
            }
        }

        public void reportCoins(Coin a, Coin b)
        {
            Debugger.debug(getClass(), 1, "Reporting coins.... ");
            Debugger.debug(getClass(), 3, "Coin a: " + a);
            Debugger.debug(getClass(), 3, "Coin b: " + b);
            try
            {
                serverSession = createBLSSession();
                a.encryptTransactionlist((RSAPublicKey)BLSCertificate.getPublicKey());
                b.encryptTransactionlist((RSAPublicKey)BLSCertificate.getPublicKey());
                serverSession.send(new ReportCoinsPacket(a, b));
                serverSession.close();
                Debugger.debug(getClass(), 1, "Reported coins succesfully.");
            }
            catch(Exception ex1)
            {
                Debugger.debug(getClass(), 1, "Error reporting coin.");
            }
        }

    }
}
```