Wibephone

Jens Henrik Skuldbøl

Kongens Lyngby 2014
IMM-B.Eng-2014-????

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk IMM-B.Eng-2014-????

Summary (English)

Wibephone is a suggestion on how to modernize the entry phone used in many Danish appartment buildings today. The goals of Wibephone includes the ability to let people into the stairway, even though you are not home. This project will document the development of a prototype of part of this system, particularly a backend module, which will handle authentication of users' identity, and the challenges of computer security set by this feat.

The main focus of the project is safe identification, communication, non-repudiation, and handling and exchange of digital keys. This report documents the development of a module handling these things. Thsi module is a prototype developed as a webservice, with the goal of examining whether this approach would be feasible in the context of the complete project.

This report will analyze which threats a solution such as the proposed one will face, and how to handle them. It will then look at different suggestions on how to actually do this, and choose which approach to take.

Summary (Danish)

Wibephone er et bud på en modernisering af dørtelefonen, som bl.a. vil tillade åbning af en dør, selvom man ikke er hjemme, fra f.eks. en smartphone. Dette projekt omhandler prototypeudviklingen af en del af dette projekt, specifikt et backend-modul, som håndterer verificering af brugeres identitet og de sikkerhedsmæssige udfordringer, som er forbundet med dette.

Projektet hovedfokus er sikker identifikation, kommunikation, uafviselighed, og udveksling og håndtering af digitale nøgler. Denne rapport dokumenterer udviklingen af et modul, som håndterer disse ting. Modulet er en prototype udviklet som en webservice for at se, om denne løsning er gangbar i det samlede billede. En webservice er valgt for at sikre platform-uafhængighed.

Jeg vil i denne rapport analysere, hvilke trusler en løsning som denne vil stå overfor, og hvordan man kan imødekomme dem på en sikker måde. Derefter vil jeg se på forskellige implementerings-specifikke løsninger, og se hvilke af disse der bedst opfylder kravene opstillet fra analysen.

Forord

Denne opgave er skrevet ved DTU Compute på Danmarks Tekniske Universtet, DTU, som en afsluttende bachelor-opgave på linjen Diplom IT (B.Eng.) på 20 ECTS-point. Pojektet er skrevet under vejledning af lektor Christian Damsgaard Jensen og i samarbejde med Wibephone under overordnet vejledning af Kristian Wichmann Larsen og teknisk vejledning af Malthe Øhlers.

Denne opgave består, udover denne rapport, af et stykke udviklet software, vedlagt i komprimeret format ved elektronisk aflevering, samt på CD-ROM ved fysisk aflevering.

Lyngby, 19-September-2014

Jens Henrik Skuldbøl

Indhold

Kapitel 1

Indledning

Denne indledning giver en overordnet beskrivelse af, hvad projektet går ud på. Det beskriver den overordnede idé bag Wibephone, for derefter at afgrænse projektet ved at definere dets hovedfokus, samt beskrive hvilke relaterede emner betragtes som liggende uden for projektets omfang. Slutteligt vil indledningen kort opsummere resultatet af projektet og give et overblik over rapportens struktur.

1.1 Dørtelefoner i dag

I de fleste danske byer benyttes i dag traditionelle dørtelefoner i mange boligbyggerier. Formålet med disse dørtelefoner er at begrænse adgangen til det fælles opgangsareal – en opgave de løser, men med begrænsninger og komplikationer.

Opgangsarealet i et boligbyggeri kan i dag ses som et trin mellem den offentlige, frit tilgængelige gade og lejlighedens private rum. Dette giver nogle praktiske anvendelsesmuilgheder, hvor man som beboer kan have brug for at give adgang til opgangen, uden nødvendigvis at lukke folk ind i lejligheden. Dette kan for eksempel være pakkebudet, som skal aflevere en pakke, eller en bekendt som skal hente en glemt taske eller jakke. Hvis dette sker, mens man som beboer er

hjemme i lejligheden, er dette ikke noget problem med det eksisterende system, da man så kan lukke buddet ind og modtage pakken med det samme, eller selv overlevere den glemte jakke.

Hvis man som beboer derimod ikke er hjemme, når posten ringer på, er historien en anden. Buddet skal udfylde og aflevere en postanmeldelse, pakken skal afleveres på et afhentningssted, og modtageren kan hente den der – typisk 1-2 arbejdsdage senere end han ellers ville have modtaget den.

Hvis pakken blot kan stilles uden for døren til modtagerens lejlighed, kan buddet naturligvis bare lukkes ind af en anden beboer i opgangen, såfremt der er nogen hjemme. Dette er dog ikke en optimal løsning. Hvad hvis nu det slet ikke er posten, som ringer på, og opgangen bliver offer for hærværk eller indbrud? Mange åbner i dag døren for folk, som de reelt ikke ved, om har et anliggende i opgangen. Det bliver anset for uhøfligt ikke at gøre det, og derfor mener mange, at de ikke kan tillade sig at lade være, selvom vi reelt set ikke burde sættes i en situation, hvor vi er tvunget til at træffe det valg.

Udover de sikkerhedsmæssige problemer nævnt ovenfor er der en række andre problemer ved den traditionelle løsning. Navnetavler skal manuelt vedligeholdes i de enkelte opgange, hvilket er en relativt ressoursekrævende opgave, udbyttet taget i betragtning. Hvis en beboer mister en nøgle skal alle låsene som den pågældende nøgle åbner skiftes, og de berørte beboere skal have udstedt nye nøgler. Dette bliver hurtigt en omkostningsfuld affære, da en nøgle i nogle tilfælde åbner flere døre i en boligblok eller endda boligforeningen.

1.2 Wibephone

Wibephone vil forsøge at løse disse problemer ved at digitalisere dørtelefonen. Ved at installere tablets i stedet for dørtelefoner kan man åbne op for muligheden for videosamtale, når der ringes på døren. Samtidig skal det være muligt at modtage disse videoopkald på sin smartphone, hvis man ikke er hjemme, og åbne døren fra denne, hvis man ønsker dette.

I tilfælde af at man skulle miste sin digitale nøgle, eller enheden den befinder sig på, som for eksempel ens telefon, vil det være muligt at spærre for adgangen for den enkelte nøgle, fremfor at skulle udskifte låse og udstede nye nøgler.

Samtidig vil de digitale dørtelefoner give mulighed for automatisk opdatering af navnetavler og central vedligeholdelse af disse gennem et register over beboere.

Sidst men ikke mindst vil korrekt implementation af digitale nøgler gøre systemet uafviseligt (eng. *non-repudiation*), hvilket, med tilstrækkelig logning, vil gøre det muligt altid at korrekt påvise hvem, der har åbnet en dør på et givent tidspunkt.

1.3 Fokusindsnævring

Dette projekt vil omhandle prototypeudviklingen af et modul, der skal indgå i den samlede Wibephone-løsning. Dette modul vil håndtere verificering af personers identitet, når de forsøger at åbne en given dør, og bekræfte, at de har de fornødne rettigheder til at åbne denne dør.

Følgende områder er dele af den komplette Wibephone-løsning, som dette projekt *ikke* vil behandle

- Videoopkald
- Notifikation til mobile enheder ved opkald fra dørtelefon
- Fysisk åbning af lås

Projektet vil diskutere mulige strategier for netværksopsætning, men at komme med en specifik løsning til dette er ikke et mål.

Ydermere vil denne rapport diskutere funktionalitet, som lader brugere udstede midlertidige, tidsbegrænsede nøgler. Dette forestilles at være relevant i forhold til for eksempel håndværkere eller andre, som har brug for at kunne tildeles fri adgang til et område i en bestemt tidsperiode. Dette betragtes som værende udenfor projektets omfang, men der vil trods dette forsøges at tage hensyn til det under systemets udarbejdelse med henblik på eventuel fremtidig implementation.

1.4 Resultat af projektet

Resultatet af dette projekt er en webservice, som vil verificere brugeres identitet og bekræfte deres rettigheder til at tilgå en bestemt ressourse. Denne webservice

er udviklet i Django[1] som en applikation, der kan køre enten på en isoleret server eller som en integreret del af en større Django webserver.

1.5 Rapportstruktur

Denne rapport kan inddeles i tre overordnede punkter. Det første af disse punkter er den analytiske del, bestående af afsnit 2 og 3. Disse omhandler det foregående arbejde, hvor det undersøges hvilke produkter, der er på markedet og hvilke teknologier, der er tilgængelige. Derefter analyseres projektets overordnede problemstillinger, som nedbrydes i mindre problemer.

Det andet hovedpunkt er den praktiske del. Her beskrives et overordnet design for, hvordan problemerne opstillet ovenfor kan løses, og derefter mere specifikt hvordan de er løst. Dette er afsnit 4 og 5.

Sidst vurderes resultatet af projektet i form af tests, og der ses på hvor vidt de opstillede mål for projektet er nået. Dette sker i afsnit 6 og 7.

Appendix A indeholder et uddrag af Wibephones business case, som beskriver den overordnede idé bag Wibephone. Dette er inkluderet for at sætte dette projekt i et større perspektiv.

[1] http://www.djangoproject.com

Kapitel 2

Baggrund

Dette afsnit vil kigge på nogle af de eksisterende løsninger for digitale låsesystemer, som enten allerede findes på markedet i dag, eller vurderes at være på vej dertil. Derudover vil det se på nogle af de bagvedliggende teknologier til digital identitetshåndtering, som muliggør disse systemer. Formålet med dette er at identificere, hvordan Wibephone skal adskille sig fra andre produkter, hvilke det er en mere direkte konkurrent til, samt hvilke bagvedliggende teknologier det giver mening at anvende.

2.1 Låsesystemer

Dette afsnit vil se på nogle af de låsesystemer, der i dag findes på det danske marked eller er på vej dertil.

2.1.1 MVC AccessZone

AccessZone er en Bluetooth-baseret løsning, udviklet til både private, virksomheder og foreninger. Som supplement til Bluetooth-løsningen tilbyder de

også kode-baseret adgangskontrol, så adgangskoder kan tildeles brugere, som har brug for adgang uden en nøgle. Systemet understøtter fjernbetjent adgang via. både PC, udstedelse af tidsbegrænsede nøgler, granulerede rettigheder og logning af aktivitet.

Produktinformation for AccessZone for foreninger findes på `http://www.mvc-data.com/accesszone/index.php/forening`.

2.1.2 BEKEY

BEKEY er et firma startet i 2008 af FK Distribution. BEKEY blev startet som et resultat af, at beboere var utrygge ved at lukke fremmede ind, når FK Distributions omdelere skulle dele reklamemateriale ud i boligblokke. Systemet fungerer ved at man fra en central webportal kan uddele tidsbegrænsede nøgler til folk med begrænset erinde i bygningen. Systemet bruger Bluetooth som transportprotokol og lader brugere åbne låse gennem en smartphone applikation.

BEKEY tilbyder løsninger til både offentlige institutioner, private, og etageejendomme. Systemet lader dog til på nuværende tidspunkt at være mest udbredt hos reklameomdelere og ældreplejen.

Produktinformation findes på `http://bekey.dk/`

2.1.3 Poly-Control

Poly-Control er et dansk familiefirma, som står bag Danalock. Danalock er en digital lås, styret via en smartphone applikation over Bluetooth Smart eller Z-Wave. Danalock lader til at være rettet mod et privat marked, men Poly-Control reklamerer også for forretningsorienterede løsninger på deres hjemmeside.

Poly-Control er på vej med yderligere produkter til Danalock, bl.a. et keypad, som åbner for muligheden for at tildele pinkoder til folk, samt en key-fob, som det blandt andet kendes fra moderne biler.

Produktinformation findes på `http://www.poly-control.com/#danalock`

2.1.4 Verisure SmartLock

Verisure SmartLock er et produkt udviklet af Secure Direct i samarbejde med Ruko. SmartLock lader kunden fjernstyre sine låse via en applikation til smartphones, og inkluderer desuden funktionalitet til at udstede midlertidige nøgler til håndværkere og lignende.

Ifølge udtalelser fra sikkerhedskonsulent Thomas Wong, via en artikel i Ingeniøren [3] kan det udledes, at systemet tilbydes gennem Secure Directs servere, altså en Software as a Service-løsning, over Internettet.

SmartLock er integreret med Secure Directs alarmløsninger, og synes markedsført til det private marked.

Produktinformation findes på `http://www.verisure.dk/las.html`

2.1.5 Wibephones plads på markedet

Størstedelen af ovenstående produkter beror sig på teknologier, der kræver, at brugeren er inden for en vis afstand af låsen for at kunne åbne den. Dette er netop ikke et mål for Wibephone, som derfor adskiller sig væsentligt fra disse produkter af netop denne grund.

Verisure SmartLock synes som den nærmeste konkurrent. Denne ses dog ikke som en direkte konkurrent, da målgruppen synes at være en anden, og anvendelsesområdet også er forskelligt. Den stærkt centraliserede arkitektur bag SmartLock har desuden mødt en del kritik. Hvor vidt Wibephone skal benytte en lignende arkitektur eller ej diskuteres i afsnit 3.1

MVC AccessZone tilbyder også mange af de funktionaliter beskrevet i Wibephones forretningsidé. Hvor Wibephone profilerer sig på at tillade fjernstyring via smartphones, lader AccessZones primære salgspunkt dog til at være Bluetooth-adgangen, mens fjernstyringen lader til at være tænkt som en administrativ feature.

2.2 Digital identitet

En identitet defineres som en række egenskaber, som unikt kan tilknyttes én person. Dette er også tilfældet for en digital identitet, og ofte bruges mange af

de samme egenskaber, såsom f.eks. navn, CPR-nummer eller biometriske data, til at udgøre en digital identitet.

En identitet skal altid ses i en kontekst. Det er i denne kontekst, identiteten skal kunne unikt overføres til en enkelt person. For at dette er muligt, er det i nogle tilfælde nødvendigt at koble flere egenskaber sammen for at skabe en identitet. Hvor det formenligt er nok blot at bruge et fornavn som identitet i en enkelt husstand, er det tvivlsom, at denne ene egenskab er nok til unikt at identificere en person så snart, man bevæger sig udenfor denne husstand.

En identitet skal ikke betragtes som hemmelig, og derfor er det nødvendigt at kunne bekræfte påstande om identiteter. Eksempler på dette er tilknytningen af et løsen til et id, en løsning der længe har været vidt udbredt på Internettet, eller fremvisning af et gyldigt bevis udstedt af en betroet tredjepart. Sidstenævnte kendes fra den fysiske verden, hvor vi har blandt andet pas udstedt af Udenrigsministeriet, som dermed garanterer for vores identitet, her bekræftet overfor den forespørgende enhed ved et billede af indehaveren.

Eksempler på forkert håndtering af identiteter blev bragt frem i lyset i den danske presse i tiden efter offentliggørelsen af flere prominente, danske politikkeres CPR-numre, samt hackerangrebet mod CSC, hvor politiets registre blev tilgået. Efterfølgende er det blevet kritiseret, at virksomheder har brugt CPR-nummeret som en bekræftelse af identitet, i stedet for blot et unikt id, hvilket det reelt set er. Datatilsynet har sidenhen udtalt, at CPR-nummeret ikke betragtes som en følsom oplysning[2], og ikke kan bruges til at verificere en identitet.

Ovenstående viser ikke blot hvor vigtigt, det er, at vi korrekt kan identificere folk digitalt, men også hvor vigtigt det er at skelne mellem identifikation og verifikation.

2.3 Identitetsstyring

Identitetsstyring er håndteringen af digitale identiteter. Dette inkluderer udstedelse, verifikation og beskyttelse af disse. Det er vigtigt at bemærke, at identitsstyring ikke er det samme som adgangskontrol. Identitetsstyring siger i sig selv intet om, hvilke rettigheder en bruger har, men udelukkende noget om hvem han er. Identiter kobles dog ofte sammen med rettigheder for belejlighedens skyld, så i praksis er de to for det meste tæt relateret.

Federated Identity Management er en metode, som lader brugeres identitet

kobles sammen på tværs af forskellige identitetsstyringssystemer. Hvor identitsstyringssystemer ofte er tiltænkt til at fungere indenfor et afgrænset domæne, som f.eks. en virksomhed eller et website, lader Federated Identity Management brugere koble deres identiteter på tværs af domæner.

Idéen bag Federated Identity Management er, at brugere kun skal vedligeholde deres digitale identitet ét sted, og at denne så kan bruges på tværs af domæner. Paul Madsen[1] beskriver som et eksempel brug af Federated Identity Management til at tillade brugere at oprette og vedligeholde en enkelt identitet på tværs af diverse online sociale services.

Den overordnede struktur i et Federated Identity Management system bygger på, at en *Relying Party* kontakter en *Service Provider*. Når en Relying Party så skal bruge nogle oplysninger om en bruger, så hentes disse fra den givne Service Provider, under forudsætning af at de nødvendige rettigheder er mødt.

2.3.1 Single Sign-On

Single Sign-On er en service, som leverer en delmændge af den funktionalitet som ligger i Federated Identity Management. Hvor Federated Identity Management fungerer som serviceudbyder for en komplet, digital identitet, så er Single Sign-On begrænset til funktionalitet som omhandler authentication.

2.3.2 Kerboros-baseret

Implementationen af et Single Sign-On system laves ofte, dog ikke nødvendigvis altid, som et Kerboros-lignende system. Her består systemet af tre aktører – en bruger, og, som med Federated Identity Management, en Relying Party og en Service Provider. Når en bruger forsøger at logge ind hos en Relying Party, skal denne bruger kunne fremvise en gyldig token, som verificerer, at brugeren er, hvem han udgiver sig for at være. Såfremt brugeren ikke har sådan en token viderestilles forespørgslen til en Service Provider. Denne vil så verificere brugerens identitet, og udstede en, typisk tidsbegrænset, token, hvor Service Provideren garanterer brugerens identitet overfor Relying Party.

2.3.3 SAML

SAML står for *Security Assertion Markup Language*, og er en åben, XML-baseret standard for udveksling af data mellem *Relying Parties* og *Service Providers* i authentication forespørgsler, udviklet af OASIS Security Services Technical Committee. Bemærk at parternes navne i SAML-regi er anderledes. Her vil en typisk forespørgsel ofte foregå ved at en bruger kontakter en *Service Provider* (tidligere omtalt som Relying Party) for at tilgå en bestemt ressource. Herefter vil brugeren blive henvist til en *Identity Provider* (tidligere kaldet Service Provider), som udleveret en token til brugeren, som denne herefter fremviser til serviceudbyderen som et gyldigt bevis for sin identitet.

Grundet dets rødder i XML anvendes SAML ofte i webservices baseret på SOAP. SAML er i sig selv fuldstændigt uafhængigt af, hvilken metode identitetsudbyderen bruger til at bekræfte en brugers identitet, da det er en standard udelukkende til transport af data.

Kapitel 3

Analyse

Problemet opstillet i dette projekt er at begrænse adgangen til et område til visse personer. Dette inkluderer at sikre sig, at folk er, hvem de udgiver sig for at være, samt at gøre det umuligt for brugere at kunne benægte, at de har tilgået det afspærrede område.

Samtidig skal dette ske på så belejlig en måde som muligt. Brugere skal have mulighed for at have flere nøgler, og såfremt de skulle gå hen og miste en af disse, skal det være muligt at spærre for adgangen for denne nøgle uden, at andre brugere påvirkes af dette.

Dette projekt vil forsøge at løse disse problemstillinger ved at implementere et modul til Wibephones backend. Dette modul vil sørge for netop dette – at bekræfte brugeres identitet og kontrollere, om de har rettigheder til at foretage de operationer, de forsøger.

3.1 Systemarkitektur

Da Wibephone ikke skal kunne servere forskellige oplysninger om brugeres identitet på tværs af forskellige systemer, kan systemet opfattes som fungerende på

ét domæne. Det er derfor ikke nødvendigt at implementere som en fuld Federated Identity Manager. Da en identitet skal have mulighed for at have tilknyttet flere nøgler, og potentielt skal kunne authenticates på flere systemer, alt efter hvordan arkitekturen besluttes opbygget, vil en Single Sign-On-løsning antages at være en god løsning.

3.1.1 Software as a Service-arkitektur

En mulighed er, ligesom Secure Direct gør med VeriLock, at tilbyde løsningen som Software (Platform?) as a Service. Dette vil i realiteten betyde, at al hosting klares af Wibephone, og derfor er det kun dørtelefonerne som skal installeres hos kunderne (boligforeningerne).

En sådan arkitektur vil mindske behovet for opsætning og vedligeholdelse hos kunden. Samtidig vil den dog skabe en central flaskehals, hvorigennem al trafik går, og dermed skabe et *single point of failure*.

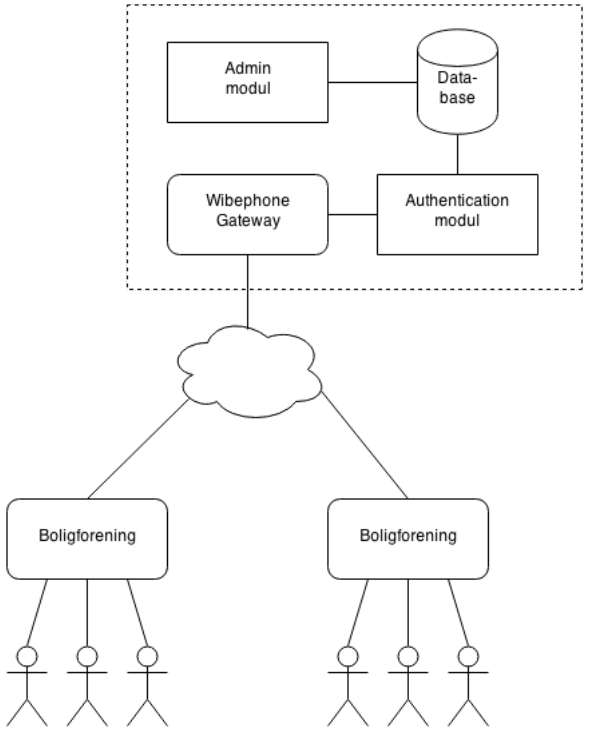
Et eksempel på, hvordan Wibephone kunne se ud med denne arkitektur, er vist i figur 3.1. Som det ses her ligger størstedelen af funktionaliteten hos Wibephone, der fungerer som serviceudbyder. De vedligeholder en database over systemets brugere. Når en bruger forsøger at åbne en dør, går denne forespørgsel gennem Wibephones gateway og behandles af deres system.

3.1.2 Centraliseret arkitektur

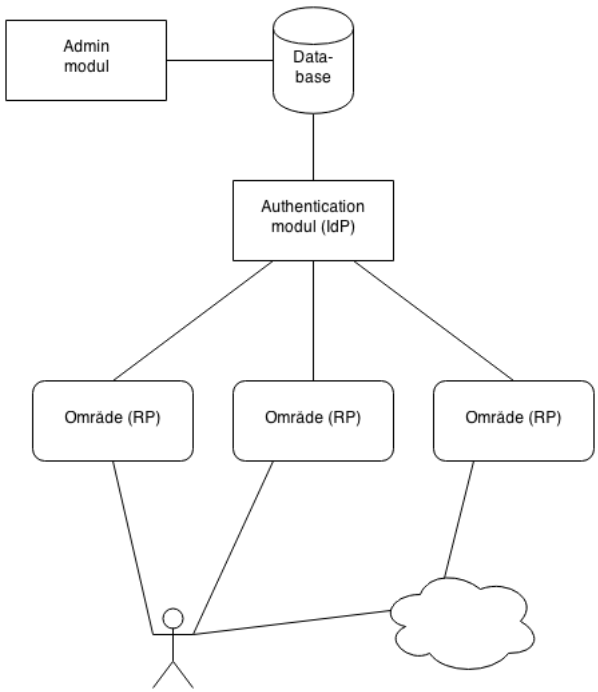
En centraliseret arkitektur er kendetegnet ved, at et element fungerer som centralt omdrejningspunkt for systemet. I en Single SignOn-løsning vil det centrale element oftest være Identity Provideren.

Et grundlæggende eksempel på, hvordan en simpel, centraliseret opsætning af Wibephone kunne se ud, kan ses i figur 3.2. Tilgangen gennem skyen repræsenterer her, at en bruger vil kunne fjernbetjene systemet gennem for eksempel en smartphone, mens den direkte kobling mellem brugere og områder repræsenterer interaktion med systemet gennem fastmonterede enheder, som for eksempel dørtelefoner opsat i beboernes lejligheder.

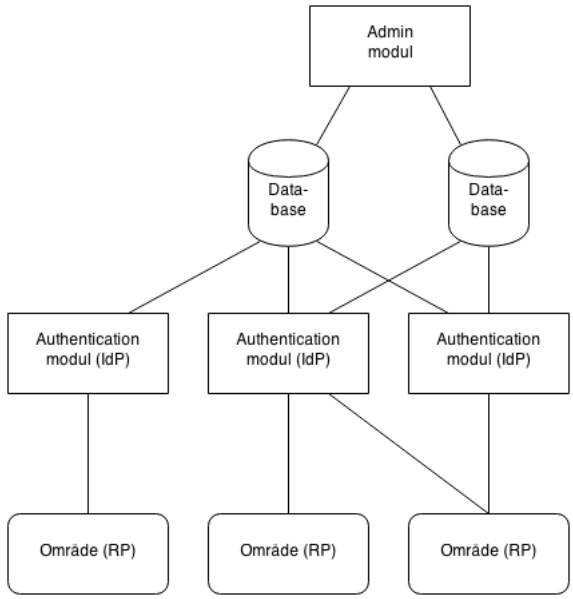
Implementeret korrekt vil en mere avanceret opsætning også være mulig. Det vil for eksempel være muligt at sætte systemet op på en sådan måde, at en Relying Party kan kobles op til flere Identity Providers, samt at en Identity Provider kan kobles op på flere brugerdatabaser. Dette kan for eksempel være



Figur 3.1: Wibephone som Software as a Service-løsning.



Figur 3.2: Generelt eksempel på lokalt centraliseret arkitektur.



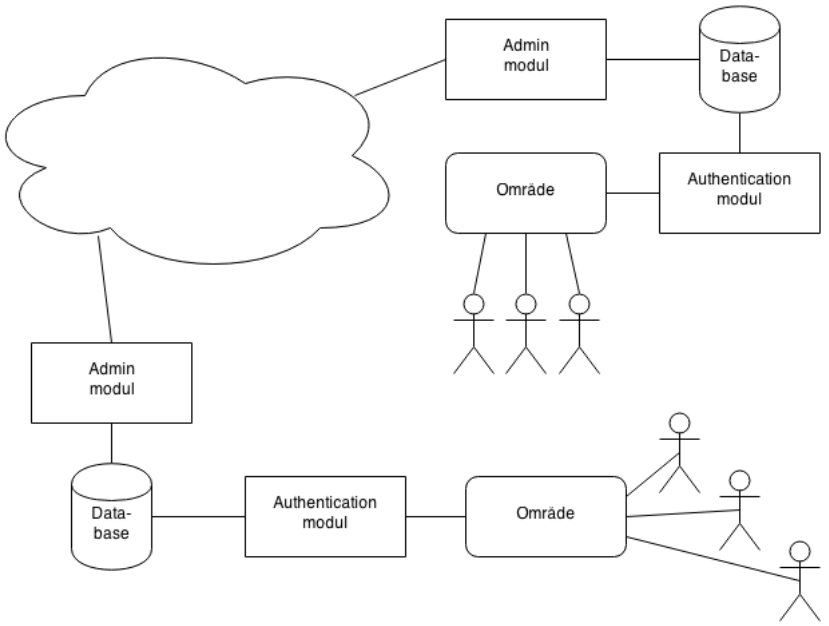
Figur 3.3: Avanceret eksempel på hvordan en større, centraliseret arkitektur evt. kunne se ud.

relevant i en større forening, hvor man ønsker, at individuelle afdelinger hver især vedligeholder deres egne brugerdatabaser, men man ønsker en samlet gateway til at styre adgangen gennem. Det modsatte tilfælde, hvor man samler brugerdata i én database, men opsætter Identity Providers individuelt i hver enkelt afdeling, som så hver især authencater brugere op mod den centrale database. Sådan en arkitektur vil bibeholde det centrale element, men vil samtidig låne elementer fra en distribueret arkitektur. Et eksempel på en sådan arkitektur er demonstreret i figur 3.3.

3.1.3 Distribueret arkitektur

Det komplette modstykke til den centraliserede arkitektur beskrevet ovenfor er en distribueret arkitektur. Her fungerer hvert enkelt delsystem fuldstændigt uafhængigt af hverandre.

Et eksempel på hvordan Wibephone kunne se ud med denne arkitektur er vist i figur 3.4. Her har hvert enkelt afgrænset område et modul til authentication af brugere og en tilknyttet database over hvem, der har adgang. På denne måde er et enkelt område ikke afhængigt af det større system, og vil således være upåvir-



Figur 3.4: Wibephone med distribueret arkitektur.

ket af eventuelle nedbrud. Et system som dette har dog en tungere arbejdsbyrde hvad angår opsætning og administration.

3.1.4 Wibephones arkitektur

Den valgte arkitektur til Wibephone tager udgangspunkt i den centraliserede arkitektur beskrevet ovenfor. Dette går fint i spænd med den opsætning, der er beskrevet i appendix A, hvor der er budgetteret med en server pr. boligforening. Det forventes, at denne ene server vil fungere som Identity Provider, og derfor være det centrale omdrejningspunkt i arkitekturen. Som det også beskrives ovenfor vil arkitekturen dog understøtte mulighed for at skalere, såfremt det skulle være nødvendigt. Det kan for eksempel være i større afdelinger eller i foreninger, hvor det er upraktisk med et samlet adgangspunkt, for eksempel på grund af geografiske afstande.

3.1.5 Valg af platform

Valget af platform til dette projekt er faldt på Django [1]. Django er et webframework i Python, bygget på MVC[2] paradigmet. Denne beslutning bygger på flere ting, som vil blive uddybet herunder.

3.1.5.1 MVC

MVC giver generelt god adskillelse af datamodel, logik og præsentation. I Djangos tilfælde er det muligt helt at udelade præsentationslaget. Da størstedelen af projektet består af en webservice, vil mængden af præsentationsdele være minimal, og det vil derfor være fordelagtigt at kunne udelukke denne helt eller delvist.

3.1.5.2 Webservice

Projektet er valgt udviklet som en webservice. Dette er primært for at sikre funktionalitet på tværs af platforme, da den forventes at skulle kommunikere med mindst to platforme, nemlig Android og iOS. Ved at sikre platformuafhængighed vil klienter til flere platforme lettere kunne udarbejdes på længere sigt.

Platformuafhængighed kan dog også opnås på andre måder, og dette er heller ikke den eneste grund til, at projektet er valgt lavet som en webservice. Ved at lave projektet som en webservice kan der abstraheres væk fra nogle aspekter, som ikke er en del af projektets hovedfokum, såsom samtidige brugere, skalérbarhed og lignende. Disse vil i stedet håndteres af den valgte webserver og opsætningen af denne.

3.1.5.3 Django

Django er valgt af flere grunde. En af disse er dets administrationsinterface, som forventes at kunne tilpasses til brug i det endelige system. Det forventes at daglig adminstration af projektet, såsom opsætning af områder, oprettelse af nye nøgler, spærring af nøgler og lignende, vil kunne klares gennem dette administrative system.

[1] https://www.djangoproject.com/
[2] Model, View, Controller

Django har, i forbindelse med dets administrationsinterface, en datamodel for brugere og brugergrupper. Disse kan bruges af administratorer til at styre, hvilke brugere har adgang til hvilke funktioner på et website udviklet i Django. Det forventes, at disse kan bruges i udviklingen af systemet, hvilket vil blive beskrevet nærmere i afsnit 4 og 5 af denne rapport.

Foruden ovenstående er Python desuden et glimrende sprog til prototype-udvikling. Valget af Django og Python er desuden truffet på baggrund af personlige præferencer og læringsmæssige mål. Django kører på en lang række webservere, inklusiv Apache HTTP Server og Nginx.

3.2 Kryptografisk nøglesystem

Eftersom verifikation af identitet er hovedmålet for systemet, er fortrolighed ikke et krav. For sikkert at kunne identificere en bruger er et andet CIA[3] princip, integritet, dog nødvendigt.

Samtidig er tilgængelighed et naturligt krav. Systemet forventes at skulle fungere i både et lukket miljø, i form af de digitale dørtelefoner, samt i et åbent, i form af smartphone-udviddelsen. Omend sidstnævnte kan forventes at være mest udsat for angreb, så er tilgængeligheden vital for begge miljøer.

Sidst skal systemet give uafviselighed. Det skal være umuligt for en bruger at så tvivl om, hvor vidt han har åbnet en dør på et givent tidspunkt. Dette er nødvendigt for at forhindre insiderangreb, hvor en beboer bevist giver adgang til uvedkommende, evt. med urene hensigter.

3.2.1 Symmetrisk kryptering

Symmetrisk kryptografi er det flest mennesker forbinder med kryptografi. Her bliver de kommunikerende parter enige om en fælles nøgle. Denne nøgle bruges så til både at kryptere og dekryptere beskeder.

Et af de tidligste eksempler på symmetrisk kryptering tilskrives Julius Cæsar. Det siges, at han krypterede sin beskeder ved at flytte alfabetet n tegn, og modtageren skulle herefter flytte alfabetet n tegn i den anden retning for at dekryptere. I denne algoritme, kaldet et *Ceasar Cipher*, er algoritmen at flytte alfabetet, mens n er nøglen.

[3] Confidentiality, Integrety, Availability

3.2.1.1 Nøglegeneration

For hvert sæt af kommunikerende parter, der skal kunne kommunikere fortroligt skal der, såfremt de bruger den samme krypteringsalgoritme, genereres en nøgle. Det vil sige, at hver part skal generere $n-1$ nøgler, hvor n er antallet af parter, og for at alle parter skal kunne kommunikere sikkert med hverandre skal der derfor genereres $n(n-1)$ nøgler. Altså stiger antallet af unikke nøgler, der skal genereres i et system, eksponentielt med antallet af kommunikerende parter.

3.2.1.2 Nøgledistribution

Symmetrisk kryptografis natur taget i betragtning er nøgledistribution essentiel. Da den samme nøgle bruges til både at kryptere og dekryptere beskeder, er det vigtigt, at denne distribueres mellem de kommunikerende parter. Hvis ikke dette er tilfældet, kan man ikke garantere, at kommunikationen foregår fortroligt. En almindelig metode til udveksling af nøgler er *Diffie-Hellman key exchange*. Denne algoritme sikrer en fortrolig kanal mellem to parter, men stiller i sin oprindelige form ingen garantier for identiten af, hvem man kommunikerer med[4]. Oftest er den sikreste metode, at parterne mødes personligt, og udveksler nøgler. Dette er dog ikke praktisk eller i nogle tilfælde endda muligt.

3.2.1.3 Brug af nøgler

Når data krypteres med en symmetrisk algoritme, bruges den samme nøgle til at dekryptere dem igen. Da alle indvolverede parter har denne nøgle, er det derfor ikke muligt at bestemme hvem, der har krypteret dem. Symmetrisk kryptografi er derfor ikke velegnet til signering af data.

Symmetriske, krypteringsalgoritmer betragtes generelt som værende hurtigere end asymmetriske.

3.2.2 Asymmetrisk kryptering

Grundprincippet i et asymmetrisk nøglesystem er, at hver part har to nøgler – en privat (hemmelig) nøgle, som kun han kender, og en offentlig. Udtrykket

[4] Dette kan dog sikres ad andre veje

asymmetrisk kommer af, at i nogle implementeringer, f.eks. RSA, er de to nøgler hinandens modstykker. Beskeder krypteret med den ene kan derfor kun dekrypteres med den anden. Dette betyder i praksis, at hvis A kan dekryptere en besked med Bs offentlige nøgle, så må denne besked nødvendigvis være krypteret med Bs private nøgle. Såfremt Bs private nøgle ikke er blevet kompromitteret, kan A altså være sikker på, at beskeden er signeret af B. På samme måde kan A kryptere en besked til B med dennes offentlige nøgle, hvilket sikrer, at kun Bs private nøgle kan dekryptere den.

3.2.2.1 Nøglegeneration

Da nøgler i asymmetrisk kryptering er knyttet til en enkelt part, og ikke et sæt af kommunikerende parter, er den nødvendige mængde af nøgler langt mindre end i symmetrisk kryptering. Dette skyldes, at hver part kan bruge den samme nøgle til at kommunikere uforstyrret med flere parter, jævnfør ovenstående eksempel på brug.

3.2.2.2 Nøgledistribution

Da asymmetrisk kryptografi indvolverer to nøgler, en hemmelig og en offentlig, er distribution ofte mindre besværligt end ved symmetrisk kryptografi. Da det kun er offentlige nøgler, som skal distribueres mellem parterne, er det ikke nødvendigt at oprette en sikker kanal at distribuere dem over. Det er dog stadig væsentligt at vide præcis, hvem man kommunikerer med ved udveksling af nøgler. Hvis ikke dette er tilfældet, kan man risikere, at en utilsigtet tredjepart opsnapper nøglerne, bytter dem ud med sine egne, og derved ikke bare gør det muligt for ham at lytte med, men også gør det umuligt for de to parter at kommunikere uden ham som mellemled, da beskeder vil være krypterede med hans nøgle.

Distribution af offentlige nøgler foregår derfor ofte gennem såkaldte offentlige nøgleservere. Derved mindskes antallet af betroede parter, der skal holdes styr på, da en bruger kun har behov for at kende serverens identitet, hvorefter denne garanterer for identiteten på de udleverede nøgler.

3.2.2.3 Brug af nøgler

To typiske strategier i asymmetrisk kryptering er *Forward Public Key Encryption* og *Enveloped Public Key Encryption*. Forward Public Key Encryption er,

hvor man, som beskrevet ovenfor, krypterer sin besked med modtagerens offentlige nøgle for at sikre dens fortrolighed. Navnet kommer af, at afsenderen på forhånd har anskaffet sig modtagerens offentlige nøgle.

Enveloped Public Key Encryption er, hvor man først udfører en Forward Public Key- kryptering, og derefter signerer beskeden ved at krypere den med sin egen private nøgle. Dette gør, at modtageren kan sikre at beskeden kommer fra den påståede afsender. Analogisk set svarer dette til middelalderens laksegl, som på den tid var både en garanti for afsenderens identitet, samt at beskeden var kommet uåbnet frem.

I begge ovenstående strategier er det vitalt, at man er sikker på, at man har de korrekte nøgler. Hvis man ikke kan garantere identiten på vedkommende, man udveksler nøgler med, har man ikke længere nogen garanti for identiten på hvem, man kommunikerer med.

Som det fremgår af ovenstående eksempler er en af asymmetrisk krypterings fordele, fremfor symmetrisk kryptering, muligheden for at signere data.

3.3 Datamodel

Baseret på den traditionelle nøgleløsning forventes det, at beboere kan have flere nøgler, også i det nye system. Sammenholdt med kravet om, at det skal være muligt at inddrage enkelte nøgler, kan følgende krav til datamodellen opstilles

- Der skal kunne knyttes flere nøgler til samme bruger.
- Der skal kunne differentieres mellem nøgler, også dem tilhørende samme bruger.

Ligeledes kan der adopteres nogle krav fra den traditionelle løsning omkring differentiering af adgang til områder på brugerniveau. Mens det kan forventes, at ansatte i en boligforening har adgang til alle de fælles områder, er dette ikke nødvendigvis tilfældet for beboerne. Samtidig er Wibephone tænkt som en generel løsning, og skal derfor tage højde for, at forskellige boligforeninger har forskellige politikker omkring dette. Det kan derfor forventes, at et krav til systemets datamodel er, at den skal tage højde for, at de enkelte boligforeninger kan tilpasse dette.

3.4 Kravspecifikation

Resultatet af ovenstående problemanalyse er, at systemet skal leve op til følgende krav

- Brugere skal med sikkerhed kunne identificeres.
- Systemet skal sikre, at beskeder ikke er blevet ændret under transport.
- Systemet skal sikre uafviselighed.
- Systemet skal give mulighed for, at brugere kan have flere nøgler.
- Systemet skal give mulighed for, at nøgler kan inddrages enkeltvis.
- Systemet skal give mulighed for, på brugerniveau, at definere hvilke områder, der skal gives adgang til.

Derudover kan der opstilles en række ikke-funktionelle krav

- Systemet skal imødekomme en vis tilgængelighed.
- Systemet skal kunne håndtere et vist antal samtidige brugere.
- Systemet skal have rimelige svartider.

Disse krav er dog sekundære ifht. dette projekt, da det udviklede system er en prototype, og er derfor ikke stærkt definerede.

Kapitel 4

Design

Dette afsnit beskriver designet bag den udviklede prototype. Som nævnt tidligere er prototypen udviklet som en Single Sign-On løsning, og tager udgangspunkt i en protokol arbejdet med i et tidligere kursus.

4.1 Protokol

Protokollen benyttet i dette system bygger på AMP. AMP er en protokol udleveret som en del af en obligatorisk opgave i *02239 – Datasikkerhed* i efterårssemestret 2013. Opgaven gik ud på at beskrive protokollen, samt at finde og rette en brist i protokollens sikkerhed. Studerende fik efterfølgende fortalt, at protokollen er en simplificeret udgave af Googles Single Sign-On protokol.

Opgavebesvarelsen er vedlagt i appendix B.

4.1.1 AMP

AMP er en Single Sign-On protokol, som lader en bruger modtage data fortroligt fra en Service Provider, efter at være blevet authenticated gennem en tredjepart,

```
1  Protocol: AMP
2
3  Types: Agent C,s,RP;
4         Number Request,ReqID,Data;
5         PublicKey K
6
7  Knowledge:
8         C: C,s,RP,pk(s),pk(C),inv(pk(C)),pk(RP);
9         s: C,s,pk(s),inv(pk(s)),pk(C);
10        RP: s,RP,pk(s),pk(RP),inv(pk(RP))
11
12 Actions:
13
14 C->RP: {C,RP,Request, K}pk(RP)
15 RP->C: {C,s,RP,ReqID,Request}K
16
17 C->s:  { {C,s,RP,ReqID,Request}inv(pk(C)) }pk(s)
18
19 s->C:  { {C,s, Request, ReqID}inv(pk(s)) }pk(C)
20
21 C->RP: { {C,s, Request, ReqID}inv(pk(s)) }pk(RP)
22 RP->C: {Data}K
23
24 Goals:
25
26 RP authenticates C on Request
27 C authenticates RP on Data
28 Data secret between RP,C
```

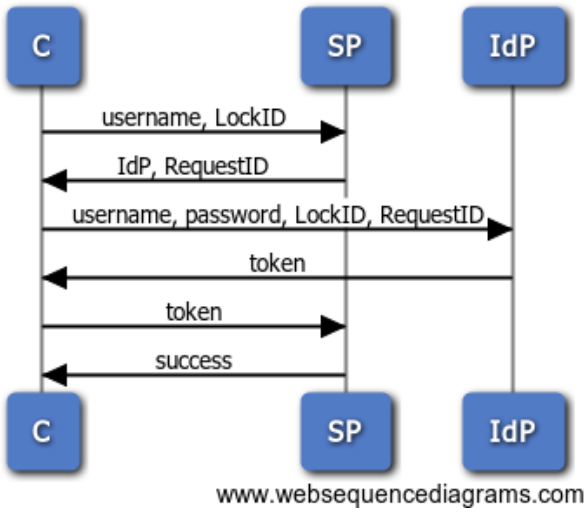
Figur 4.1: AMP protokollen i AnB-notation.

Identity Provideren.

Protokollen er beskrevet i AnB-notation i figur 4.1. Rollerne her er klienten, C, Service Provideren (her Relying Party), RP, og Identity Provider, s. Den offentlige nøgle, K, nævnt under typedeklarationen i linje 5 skal betragtes som klientens, C, offentlige nøgle. Det vil sige, at beskederne fra Service Provideren til klienten i linjerne 15 og 22 er krypteret med klients offentlige nøgle, og derfor kun kan læses af klienten.

En anden ting, som er værd at bide mærke i, er hvordan Identity Providerens svar til klienten, i linje 19, er digitalt signeret af Identity Provideren. Dette gør, at klienten ikke kan ændre i disse data, og derfor ikke kan forfalske hverken hele beskeder eller dele af dem.

På figur 4.2 ses et sekvensdiagram, der beskriver de data, der sendes frem og tilbage, som klienten ser dem. Det er især værd at bemærke svaret fra Identity Provider til klienten. Denne er beskrevet som blot *token*, da denne er fortrolig



Figur 4.2: Sekvensdiagram som viser data og transaktioner, som klienten ser dem i en succesfuld forespørgsel.

mellem Identity Provider og Service Provider. Klienten har derfor ikke mulighed for at ændre i tokens og dermed forfalske dem.

4.1.2 Sikkerhedsmæssige antagelser

For at protokollen fungerer, er der en række forudsætninger, som skal være opfyldt. Vigtigst af disse er, at Identity Provideren er betroet hos Service Provideren. Hvis ikke dette er tilfældet, vil ægtheden af godkendte tokens ikke kunne garanteres. Dette er uddybet i appendix B.

En anden nødvendighed er, at Identity Provideren på forhånd kender klientens offentlige nøgle. Dette gør, at forespørgsler signeret af klienten kan verificeres af Identity Provideren. I dette projekt er det antaget, at der ved oprettelse af nye nøgler på sikker vis er overført en offentlig nøgle til Identity Provideren, og at denne er tilknyttet den korrekte bruger.

4.2 Systemarkitektur

Systemet består grundlæggende af to parter – en Identity Provider, og en Service Provider. Service Provideren fungerer som en indgang til systemet udefra. Når en bruger ønsker at åbne en dør, sender han en forespøgsel til den pågældende Service Provider. Han får herefter et svar, som indeholder identiteten på den Identity Provider, som Service Provideren godtager bekræftelser fra. Brugeren kontakter herefter denne Identity Provider med sine legitimationsoplysninger, samt hvilken ressourse han ønsker at tilgå, i dette tilfælde, hvilken dør han gerne vil åbne.

4.2.1 Service Provider

Service Provideren er en webservice, som har to funktioner. Den første er den, brugeren kalder, når han indleder en forespørgsel. Denne tager brugerens identitet og den ønskede ressource som input. Derefter returnerer den et request ID, sammen med identiteten på Identity Provideren. Dette bruger brugeren så til at kontakte Identity Provideren, som giver ham en token, såfremt han authenticates og hans forespørgsel om adgang godkendes. Denne token videresendes herefter til Service Provideren, som reagerer på den ved enten at melde fejl eller åbne døren. Fra brugerens synspunkt kan en transaktion altså opdeles i tre trin

1. Lav forespørgsel hos Service Provider
2. Verificér identitet, og godkend forespørgsel hos udpeget Identity Provider
3. Bekræft forespørgsel hos Service Provider med legitimationsoplysninger fra Identity Provider

4.2.1.1 init

`init`-funktionen er brugerens indgangspunkt til systemet. En forespørgsel startes ved, at brugeren kalder denne funktion med information om, hvem han er, og hvilken dør han gerne vil åbne som argumenter. Funktionen genererer derefter et `requestID`, hvilket den returnerer til brugeren, samt information om hvor han skal henvende sig for at identificere sig. `init`s signatur kan altså opstilles som

```
(requestID, IdP) init(username, lockID, pubkey)
```

hvor `pubkey` er klientens offentlige nøgle. Denne medsendes for, at Service Provideren kan kryptere svaret til klienten. Argumenterne sendt til `init` krypteres med Service Providerens offentlige nøgle for at sikre sig, at uvedkommende ikke kan ændre i forespørgslen uden at risikere, at den korrupteres.

Inden `init` returnerer data til klienten, gemmes forespørgslen på serveren. Disse data bruges til at verificere den token, Service Provideren modtager fra Identity Provideren, når denne har verificeret brugerens identitet og godkendt hans forespørgsel. Disse data er specifikt det returnerede `requestID`, klientens brugernavn, samt hvilken lås der ønskes åbnet. Kald til `init` er første trin i de tre trin beskrevet ovenfor.

Bemærk at en forespørgsel ikke signeres på dette tidspunkt. Dette ville være omsonst, da Service Provideren ikke har nogen garanti for afsenderens identitet, eftersom den nøgle, en eventuel signatur skulle verificeres med, er inkluderet i forespørgslen.

4.2.1.2 request

`request` kaldes som det sidste led, fra brugerens synspunkt, i en transaktion. Klienten videresender her den token, den har modtaget fra Identity Provideren, efter dens forespørgsel er blevet godkendt af denne. Denne token indeholder data, som her matches med de data, der blev gemt, da forespørgslen blev oprettet i `init`. Såfremt data matcher, godkendes forespørgslen, og den ønskede dør åbnes. `request`s signatur bliver dermed

```
bool request(token)
```

Returværdien er dog valgfri. Klienten har ikke nødvendigvis behov for at modtage et svar på forespørgslens udfald, da det antages, at dette kan observeres fysisk, i og med at den ønskede dør enten åbnes eller ikke åbnes.

Den token, som `request` tager som argument, indeholder data, som matcher dem Service Provideren gemte i kaldet til `init`, nu signeret af Identity Provideren. Såfremt disse data matcher, kan Service Provideren altså udlede, at den pågældende forespørgsel er godkendt af Identity Provideren.

4.2.2 Identity Provider

Identity Provideren er systemets centrale element. Det er denne, der står for at verificere brugeres identitet, samt godkende deres forespørgsler. Identity Provideren har en enkelt funktion, `authenticate`, som kaldes som andet led i de tre trin beskrevet ovenfor. `authenticate`s signatur er

```
(username, lockID, requestID) authenticate(
    username, password, keyID, requestID, lockID)
```

Returværdien returneres her som en krypteret token. Denne token er først signeret med Identity Providerens private nøgle, og herefter krypteret med Service Providerens offentlige nøgle. Som beskrevet tidligere i rapporten sikrer dette, at beskeden kun kan læses af Service Provideren, samt at denne kan verificere, at beskeden oprinder fra Identity Provideren.

4.3 Datamodel

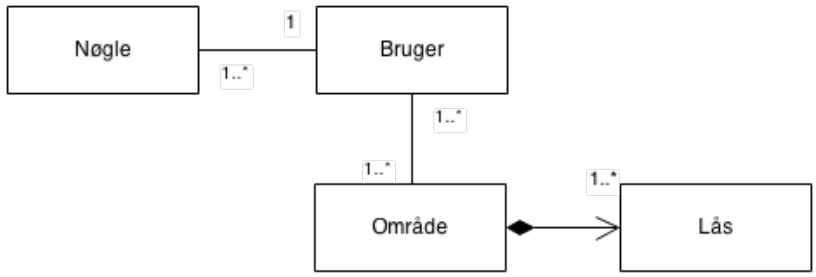
Datamodellen for projektet er relativt simpel, da den primært består af enkelte, simple objekter med få relationer.

4.3.1 Begreber

For at kunne definere datamodellen fastlægges først en række begreber.

Lås En lås i Wibephone defineret som en ressource, en bruger enten har eller ikke har adgang til at tilgå.

Nøgle En nøgle knyttet til en bestemt bruger, men en bruger kan godt have flere nøgler. Det er indforstået, at en nøgle, i dette projekts sammenhæng, er knyttet til en fysisk enhed, som for eksempel en brugers smartphone. Begrebet nøgle bruges derfor både, når der tales om den digitale nøgle, samt den enhed den opbevares på.



Figur 4.3: Grundlæggende datamodel.

Bruger En bruger er en person med fast adgang til systemet. Dette vil typisk være beboere eller ansatte, og vil formentligt ikke inkludere folk, som har fået tildelt midlertidige nøgler.

Område Et område er i Wibephone-sammenhæng et afspærret område med et eller flere adgangspunkter. Disse adgangspunkter styres af låse. Et område kan for eksempel være en boligblok, en hel boligforening eller en enkelt opgang, alt efter hvordan det administrerende organ vælger at opsætte systemet.

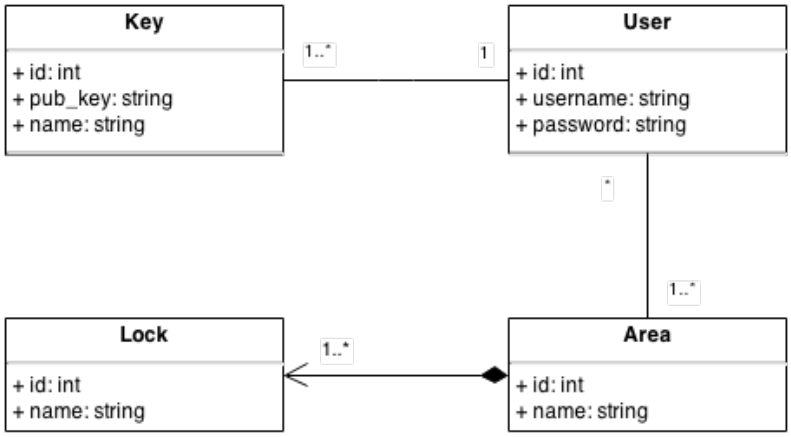
4.3.2 Datamodel

Udfra ovenstående definitioner af begreber og kravene opstillet i afsnit 3.3 kan en grundlæggende datamodel udarbejdes. Denne kan ses på figur 4.3.

Forholdet mellem områder og låse er defineret som en komposition. Mens man i den virkelige verden sagtens kan have et afgrænset område med kun ét adgangspunkt, så er det stadig et afgrænset område, hvis låsen skal have den ønskede effekt. Med hensyn til implementationen, ville det desuden komplicere opgaven unødigt, hvis der skulle være mulighed for både at tilknytte enkelte låse og områder til brugere i forhold til deres rettigheder.

Som det fremgår af kardinaliteten, er det muligt for en bruger at have flere nøgler tilknyttet, mens forholdet mellem brugere og områder er mange-til-mange, for at indikere at en bruger kan have adgang til flere områder, og at et område kan tilgås af flere brugere.

En udviddet datamodel ses på figur 4.4. Her ses, hvilke attributter hver del er modelleret med.



Figur 4.4: Wibephone datamodel med attributter.

Kapitel 5

Implementation

Dette projekt er, som nævnt i afsnit 3, udviklet i Django. Den overordnede struktur i en Django-side er et projekt, som består af en række web-applikationer. De to services, dette projekt består af, Service Provider og Identity Provider, er udviklet som to af disse applikationer. Dette gør, at de kan afvikles både på samme webserver, som en del af samme projekt, men også på hver sin server i en større implementation af arkitekturen.

I forhold til den overordnede MVC struktur Django benytter, så er der nogle afvigelser i begreberne. Modellen i MVC hedder stadig en model i Django, mens en MVC controller i Django-termer kaldes et *View*, og et MVC view er i Django en *Template*. Dette afsnit vil referere til begreberne ved deres Django-termer. De implementationsspecifikke detaljer af hver af disse applikationer gennemgås herunder.

5.1 Kryptografiske mangler

Til dette projekt er modulet `pycrypto` benyttet til at håndtere alle kryptografiske opgaver. Dette består primært af kryptering og signering af data ved brug af RSA. Implementationen af `pycrypto` har dog vist sig at være mangelfuld, da

længden af plaintext, der kan krypteres, ikke kan overskride den anvendte nøglestørrelse. Det har derfor ikke været muligt at kryptere nøgler således, at de kan sendes sikkert mellem parter.

Implementationen af denne prototype afviger derfor fra designet i mindre grad, specifikt de steder hvor en nøgle skal sendes som en del af en krypteret besked. Denne implementation kan derfor *ikke* betragtes som kryptografisk sikker, og er derfor, naturligvis, ikke egnet til at blive sat i produktion.

Fremfor at rette denne fejl ved at ændre i designet bør dette løses ved at finde et kryptografi-modul bedre egnet til et produktionsmiljø i forbindelse med fremtidig videreudvikling på projektet.

5.2 Hjælpefunktioner

For at gøre koden i de udviklede applikationer mere læselig er der impementeret nogle få hjælpefunktioner. Disse er primært til indlæsning af nøgler fra filer og kodning af tekststrenge til og fra base64 kodning. Base64 kodning har været nødvendig for at kunne transportere krypterede beskeder via HTTP. Disse hjælpefunktioner kan ses i figur 5.1.

Særligt bør dog nævnes `getClientKey`. Denne funktion bruges af Identity Provideren, når denne skal lave et opslag for at finde en brugers nøgle. Argumentet til denne funktion er ID'et på den gågældende nøgle, hvilket brugeren inkluderer i sit kald til Identity Provideren.

5.3 Service Provider

Service Providerens ansvar er at fungere som adgangspunkt for brugere. Når en bruger henvender sig med en forespørgsel godkendt af en Identity Provider, står denne for at verificere, at data i den godkendte token matcher data i den oprindelige forespørgsel. Service Provideren er en webservice, og har derfor ingen templates.

```
1  from models import WibeKey, WibeUser
2  from Crypto.PublicKey import RSA
3  import base64
4  import json
5
6  def importKey(path):
7          f = open(path, 'r')
8          key = RSA.importKey(f.read())
9          f.close()
10         return key
11
12 def getClientKey(key_id):
13         kid = int(key_id)
14         try:
15                 return RSA.importKey(WibeKey.objects.get(pk=kid).
                       key)
16         except WibeKey.DoesNotExist:
17                 return None
18
19 def getSPKey():
20         return importKey('sp/public.pem')
21
22 def getWibeUser(uname):
23         try:
24                 return WibeUser.objects.filter(user__username=uname
                       ).first()
25         except WibeUser.DoesNotExist:
26                 return None
27
28 def parseToken(token):
29         return json.loads(token)
30
31 def encode(string):
32         return base64.urlsafe_b64encode(string)
33
34 def decode(string):
35         return base64.urlsafe_b64decode(string.encode('ascii'))
```

Figur 5.1: Pythonmodul med hjælpefunktioner anvendt i de udviklede applikationer.

```
1  from django.db import models
2
3  class Request(models.Model):
4          requestid = models.CharField(max_length=100)
5          username = models.CharField(max_length=200)
6          lockid = models.CharField(max_length=10)
7          timestamp = models.DateTimeField(auto_now_add=True)
8
9          def __unicode__(self):
10                 return self.requestid
```

Figur 5.2: Kildekode for Service Providerens `Request`-model

5.3.1 Datamodel

Datamodellen for Service Provideren består af en enkelt klasse, `Request`. Denne klasse modellerer en forespørgsel og oprettes, når Service Provideren modtager en sådan i `init` funktionen.

En forespørgsel består af et brugernavn, et lås-id, et timestamp, samt et id for forespørgslen. Dette id genereres som en base64 kodning af brugernavn, lås-id samt tidspunkt, hvilket sikrer, at dette id er unikt. Timestamp gemmes for at kunne ugyldiggøre forespørgsler efter en defineret tidsperiode, og derved forhindre replay-angreb.

Koden for denne klasse ses i figur 5.2. Klassen arver fra `Model`, defineret i modulet `django.db.model`, som gør den genkendelig som en datamodel i Django. Da Python benytter dymaniske typer, men dette ikke er muligt i de fleste databasesystemer, oprettes felter på datamodellerne som instanser af klasser i frameworket. Disse klasser fungerer som abstraktionslag mellem Python-koden og Djangos ORM[1]-værktøj. Som det kan ses fra koden, er de første tre felter tekstfelter, angivet med deres maksimale længde som argument til klassernes konstruktører, mens det sidste er et tidspunkt. Argumentet givet med til dennes konstruktør gør at værdien initialiseres til tidspunktet, hvor den givne instans af `Request`-klassen er blevet oprettet.

5.3.2 View

Service Provideren har to views. Disse er funktionerne `init`, som starter en forespørgsel, og `request` som afslutter den igen.

[1] Object-Relation Mapping

5.3.2.1 init

Denne funktion starter en forespørgsel. Først dekodes input fra base64, hvorefter det dekrypteres med Service Providerens offentlige nøgle. Bemærk at klientens offentlige nøgle sendes ukrypteret, og denne implementation derfor ikke er sikker, som beskrevet ovenfor i 5.1. Kildekoden kan ses i figur 5.3.

Herefter generes et request id i linjerne 22 og 23, hvor brugernavn sammensættes med låsens id, samt det nuværende tidspunkt. Denne tekststreng kodes herefter i base64. Der genereres og gemmes herefter et `Request`-objekt med de pågældende data i linjerne 26-30.

Herefter genereres et svar indeholdende request id'et, samt en URL, hvorpå den ønskede Identity Provider kan kontaktes. Dette svar signeres af Service Provideren og krypteres derefter med brugerens offentlige nøgle. Sidst kodes både det krypterede svar samt signaturen til base64, og returneres som JSON til klienten. Dette sker i linjerne 33-52.

5.3.2.2 request

Kaldet til `request`, hvis kildekode findes i figur 5.4 viewet er det sidste trin i en anmodning om at åbne en dør. Som med `init` er input igen kodet i base64. Dette dekodes og dekrypteres herefter med Service Providerens private nøgle. Herefter verificeres Identity Provideren som afsender, ved at verificere signaturen med dennes offentlige nøgle, hvilket sker i linje 18. Bemærk at signaturen gemmes som en `long` som første element i en 2-tuple i linje 10. Dette skyldes formatet brugt af `pycryptos` `RSA` implementation. Dette er formatet, dennes `sign` metode returnerer, og `verify` tager som input. For at transportere signaturen over HTTP kodes denne i base64, og skal derfor konverteres til en tekststreng. Det andet element i signatur-tuplen er altid `None`, jævnfør dokumentationen[4].

Såfremt signaturen af input valideres indlæses de gemte data omkring forespørgslen fra oprettet i `init`. Hvis denne findes, og data matcher, godkendes forespørgslen, og de gemte data slettes. Hvis ikke returneres en fejlbesked til brugeren.

```
def init(request):

        SPKey = getSPKey()
        # Decrypt incoming request
        try:
                cipher_request = aux.decode(request.GET['request'])
                key = aux.decode(request.GET['key'])
        except MultiValueDictKeyError:
                return HttpResponse(json.dumps('Request malformed')
                    ,'application/json')

        plain_request = SPKey.decrypt(cipher_request)
        try:
                req = json.loads(plain_request)
                username = req['username']
                lock = req['lockID']
        except (MultiValueDictKeyError, ValueError):
                return HttpResponse(json.dumps('Request malformed')
                    , 'application/json')

        CKey = RSA.importKey(key)

        # Generate reqID
        reqid = username + str(lock) + time.strftime("%Y-%m-%dT%H:%
            M:%SZ",time.gmtime())
        reqid = aux.encode(reqid)

        # Save request in session
        r = Request()
        r.requestid = reqid
        r.username = username
        r.lockid = lock
        r.save()

        # Generate response
        response = {}
        response['requestID'] = reqid
        response['idpurl'] = idpurl
        response = json.dumps(response)

        # Sign and encrypt response
        signature = str(SPKey.sign(response, '')[0])
        ciphertext = CKey.encrypt(response, 32)[0]

        # Encode signature and cihpertext
        b64_sign = aux.encode(signature)
        b64_cipher = aux.encode(ciphertext)

        # Return response
        response = {}
        response['response'] = b64_cipher
        response['sign'] = b64_sign
        response = json.dumps(response)

        return HttpResponse(response, 'application/json')
```

Figur 5.3: Kildekode til Service Providerens `init` view.

```
def request(request):
        try:
                b64_token = request.GET['token']
                b64_signature = request.GET['signature']
        except MultiValueDictKeyError:
                return HttpResponse('Malformed request')

        ciphertext = aux.decode(b64_token)
        signature = aux.decode(b64_signature)
        signature = (long(signature),None)

        # Decrypt token
        SPKey = getSPKey()
        plaintext = SPKey.decrypt(ciphertext)

        print("input: {0}".format(plaintext))

        # Verify token authenticity
        IdPpublic = getIdPKey()
        verified = IdPpublic.verify(plaintext, signature)

        if not verified:
                return HttpResponse('Invalid token')

        # Check authorization
        # If request id, lock id, and username matches session, ok
        req = json.loads(plaintext)
        reqid = req['requestID']

        # Find request session
        r = Request.objects.filter(requestid=reqid).first()
        if r is None:
                return HttpResponse('Request not found')
        else:
                print('Session found')

        # Validate request against session data
        if r.username == req['username'] and \
                        int(r.lockid) == req['lockID']:
                                r.delete()
                                response = json.dumps('Pod doors
                                    open')
        else:
                response = json.dumps('I\'m sorry Dave, I\'m afraid
                    I can\'t do that')

        return HttpResponse(response,'application/json')
```

Figur 5.4: Kildekode til Service Providerens `request` view.

5.4 Identity Provider

Identity Providerens funktion er at authenticate brugeren og at checke, hvorvidt han har de fornødne rettigheder til at tilgå den ressource, han beder om, specifikt den lås han ønsker at åbne. For at muliggøre dette er der implementeret en datamodel som beskrevet tidligere, og et view som modtager en forespørgsel, og godkender eller afviser denne.

5.4.1 Datamodel

Identity Providerens datamodel beskriver hvordan brugere, områder, låse og nøgler er modelleret i softwaren. Denne datamodel er illustreret i figur 5.5.

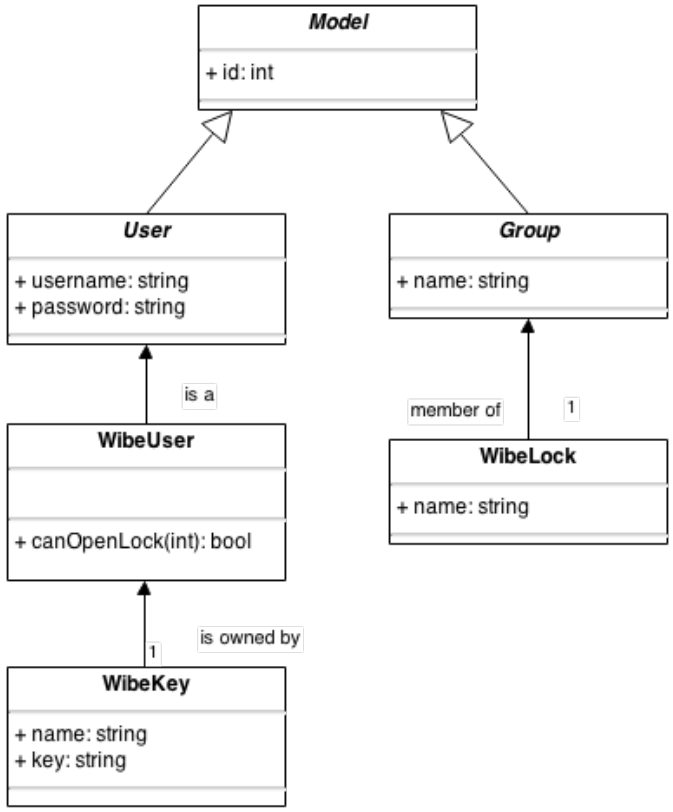
Denne datamodel tager et par af Djangos indbyggede modeller i brug, specifikt `User` og `Group` klasserne, fundet i modulet `django.contrib.auth.models`.

5.4.1.1 WibeUser

Ved at bruge `User` klassen til at modellere brugere kan Djangos medfølgende authentication backends bruges til at authenticate et brugernavn op mod et password, hvilket udnyttes i Identity Providerens authencation-view, som beskrevet nedenfor. Derudover giver dette nogle muligheder i forhold til adgang til Djangos medfølgende administrationsinterface for fremtidige tilføjelser til projektet. Dette kan for eksempel være at give en bruger mulighed for at logge ind her, og oprette nye nøgler. Kildekoden til denne klasse kan ses på figur 5.6.

Af denne kildekode fremgår det at det eneste felt i klassen er en henvisning til Djangos indbyggede `User` klasse. Denne indeholder al den nødvendige data, vigtigst brugernavn, password, og hvilke grupper brugeren er medlem af. Disse grupper bruges normalt til at styre hvilke rettigheder en given bruger har på det aktuelle website. Her bruges de dog til at modellere hvilke områder en bruger har adgang til.

Udover henvisningen til `User` har klassen også en metode, `canOpenLock`, der returnerer hvor vidt den pågældende bruger har de fornødne rettigheder til at åben en given lås. Dette afgøres ved at se på hvilke grupper brugeren er medlem af, og om nogen af dem har tilknyttet den pågældende lås.



Figur 5.5: Den endelige datamodel indarbejdet i Django Frameworket. Klassenavnene angivet i kursiv er fra frameworket, og de inkluderede felter på disse er dem som er relevante for datamodellen.

```
1 class WibeUser(models.Model):
2         user = models.ForeignKey(User)
3
4         def __unicode__(self):
5                 return self.user.username
6
7         def canOpenLock(self, lock_id):
8                 return self.user.groups.filter(id=lock_id).exists()
```

Figur 5.6: Kildekode til datamodellen for en bruger.

```
class WibeUser(models.Model):
        user = models.ForeignKey(User)

        def __unicode__(self):
                return self.user.username

        def canOpenLock(self, lock_id):
                return self.user.groups.filter(id=lock_id).exists()
```

Figur 5.7: Kildekode til datamodellen for en lås.

```
class WibeUser(models.Model):
        user = models.ForeignKey(User)

        def __unicode__(self):
                return self.user.username

        def canOpenLock(self, lock_id):
                return self.user.groups.filter(id=lock_id).exists()
```

Figur 5.8: Kildekode til datamodellen for en nøgle.

5.4.1.2 WibeLock

En lås består primært af to felter – et navn til at gøre det administrative arbejde lettere, samt en forbindelse til en gruppe, repræsenteret ved `Group`. Denne opbygning giver et en-til-mange forhold mellem grupper, hvilke repræsenterer områder, og låse, hvilket er præcis hvad vi ønsker jævnfør datamodellem i figur 5.5. Udover det menneskeligt læselige navn modellem får gennem `name` feltet, så tildeler Djangos ORM også automatisk et ID til alle dets modeller. Dette ID er hvad klienten bruger i forespørgsler til at identificere hvilken lås der ønskes åbnet.

5.4.1.3 WibeKey

Som det er tilfældet med låse, så identificeres nøgler også primært ved deres automatisk tildelte ID, men har ligeledes et navn, for at gøre administrationen lettere. Udover dette har de kun et enkelt felt, `key`, som indeholder den offentlige halvdel af det asymmetriske nøglepar, eksporteret til tekst. Kildekoden for definitionen af `WibeKey` kan ses på figur ??.

5.4.2 View

Identity Provideren har et enkelt view. Dette er repræsenteret ved funktionen `auth`. Dennes kildekode er ikke inkluderet i denne rapport, da den er meget lig koden vist for Service Provideren. Koden kan ses i kildekoden afleveret sammen med rapporten, hvis den ønskes studeret. I stedet gives her et lettere abstraheret overblik over hvilke trin der gennemgås i funktionen. Hvert trin beskrevet herunder antager at det forrige er lykkedes, såfremt der er mulighed for at det fejler, ellers afbrydes eksekveringen, og der returneres en fejlbesked.

1. Funktionen modtager et krypteret input og en signatur fra klienten gennem en HTTP forespørgsel. Disse er base64 kodede, og dekodes derfor.
2. Identity Provideren dekrypterer derefter inputtet med sin private nøgle.
3. Ud fra nøgle-id'et inkluderet i det krypterede input laves et opslag for at finde klientens offentlige nøgle i Identity Providerens database.
4. Såfremt denne nøgle findes, verificeres afsenderen af beskeden med denne offentlige nøgle og den medsendte signatur.
5. Der laves endnu et opslag for at se hvilket område den medsendte lås er tilknyttet. Herefter ses det om dette område er et brugeren har rettigheder til at åbne.
6. Såfremt dette er tilfældet dannes et svar, som indeholder brugernavn, lås-id og request-id.
7. Dette svar signeres med Identity Providerens private nøgle, og krypteres derefter med Service Providerens offentlige nøgle, og returneres til klienten (base64 kodet).

Som nævnt tidligere er denne token ulæselig for klienten. Det er derfor ikke muligt at ændre i den eller forfalske den, antaget at Identity Provideren har den korrekte nøgle fra Service Provideren, hvilket er en forudsætning beskrevet i afsnit 4.

Kapitel 6

Tests

Dette afsnit beskriver hvilke tests der er udført for at undersøge systemets korrekthed. Dette inkluderer end-to-end tests, blackbox test af de individuelle komponenter, samt en symbolsk analyse af protokollen.

6.1 Testplan

Systemet er testet i tre niveauer. Først er der lavet en analyse af protokollen, hvor denne er tjekket for eventuelle svagheder. Derefter er de individuelle komponenter af systemet testet individuelt, primært ved black box testing, for at sikre at de opfører sig som forventet, og sidst er der lavet end-to-end testing af systemet, hvor forespørgsler sendes hele vejen gennem systemet, og resultatet holdes op mod det forventede output.

6.2 Protokol

Protokollen er testet for sikkerhedshuller i Open-source Fixed-point Model Checker, OFMC, resultatet af hvilket kan ses i figur 6.1. OFMC er udviklet med det

```
1  Open-Source Fixedpoint Model-Checker version 2013a
2  INPUT:
3      git/bachelor/doc/figures/AMP.anb
4  SUMMARY:
5    NO_ATTACK_FOUND
6  GOAL:
7    as specified
8  DETAILS:
9    BOUNDED_NUMBER_OF_SESSIONS
10 BACKEND:
11   Open-Source Fixedpoint Model-Checker version 2013a
12 STATISTICS:
13   TIME 11852768 ms
14   parseTime 4 ms
15   visitedNodes: 0 nodes
16   depth: 1000000 plies
```

Figur 6.1: Output af en test af protokollen i OFMC.

formål at lave symbolanalyse på sikkerhedsprotokoller, og gennem disse finde frem til eventuelle svagheder i protokollens design.

Den opstillede test er udført med tre mål

- Service Provideren skal kunne authenticate klienten på basis af et request ID,
- Klienten skal kunne authenticate Service Provideren på basis af svaret på hans forespørgsel, og
- Resultatet af en forespørgsel, hvorvidt en dør er blevet åbnet eller ej, skal holdes fortroligt mellem klienten og Service Provideren.

og resultatet viser, at disse tre mål er opnåelige med den specificerede protokol.

For en mere uddybende gennemgang af processen bag OFMCs test, og resultaterne heraf, henvises til dennes dokumentation[5]. Denne indeholder også en uddybende beskrivelse af AnB-notationen, hvilken protokollen er beskrevet i i figur 4.1.

6.3 Komponenttests

Her testes de individuelle komponenter i systemet, Service Provideren og Identity Provideren. På hver af disse testes hvert enkelt view, for at sikre at et givent input producerer det forventede output.

6.3.1 Service Provider

Service Provideren består som tidligere nævnt af to views – `init` og `request`. Disse testes her individuelt.

6.3.1.1 init

`init` tager et input, og returnerer en værdi baseret på dette input. Det giver derfor ikke mening at teste hvorvidt en forespørgsel her lykkedes eller fejler, men der testes i stedet på værdien af den returnerede værdi holdt op mod hvad der var forventet på baggrund af inputtet. Som beskrevet i afsnit 4 er det returnerede `requestID` en sammensætning af brugernavn, lås-id og tid.

Input	Forventet output	Faktisk output	Resultat af test
'jhs', 2	'jhs2'+tid	'jhs222014-09-19T12:28:39Z'	ok
'test', 1	'test1'+tid	'test122014-09-19T12:29:47Z	ok

Da tidspunktet gemmes af serveren, og dermed ikke er det samme som det tidspunkt input sendes, kan disse ikke sammenlignes direkte. Ud fra outputtet er disse værdier dog tæt nok på de forventede input-tidspunkter til at vurderes korrekte.

Udover at der her sammenholder værdier for in- og output er disse samtidig krypterede. Det kan derfor antages, at eftersom værdierne ser korrekte ud, at både krypteringen og dekrypteringen foretages korrekt.

6.3.1.2 request

Hvis en forespørgsel når til at kalde `request` betyder det at brugren er ahtuenticated, hans forespørgsel er godkendt, og han har fået en token. Denne token matches så op mod de data der blev gemt under kaldet til `init`. Hvis disse ikke matcher skyldes det at der er blevet pillet ved klientens token, og denne er dermed ugyldiggjort. Testdata for denne findes i tabel 6.1.

6.3.2 Identity Provider

Som med Service Providerens `init` view testes Identity Provideren også ved at sammenligne input med output. Her er der dog den ekstra vinkel, at en forespørgsel kan fejle. Data for de opstillede testcases findes i tabel 6.2.

6.4 Systemtest

Sidst testes systemet som en helhed, for at sikre at integrationen mellem de individuelle komponenter er korrekt udført. I denne forbindelse er der tre scenarier som er interessante. Disse er hvor en bruger laver en forespørgsel som gennemføres korrekt, hvor en forespørgsel afvises af Service Provideren på grund af en ukorrekt token, og sidst et scenarie hvor en bruger afvises af Identity Provideren på grund af forkerte eller utilstrækkelige legitimationsoplysninger.

Test	Forventet output	Faktisk output	Resultat af test
Gennemført forespørgsel	'Pod doors open'	'Pod doors open'	ok
Ukorrekt token	'I'm sorry Dave, I'm afraid I can't do that'	'I'm sorry Dave, I'm afraid I can't do that'	ok
Fejl i authorisation	'User not authenticated'	'User not authenticated'	ok

Test	Input	Forventet output	Faktisk output	Resultat af test
Godkendt	{"lockID": 1, "username": "jhs", "requestID": "amhzMTIwMTQtMDktMTUtMTM6MjY6NDVa"}	"Pod doors open"	"Pod doors open"	ok
Ikke matchende data	{"lockID": 2, "username": "jhs", "requestID": "amhzMTIwMTQtMDktMTUtMTM6MjY6NDVa"}	I'm sorry Dave, I'm afraid I can't do that'	I'm sorry Dave, I'm afraid I can't do that'	ok

Tabel 6.1: Data for test af `request`

Test	Input	Forventet output	Faktisk output	Resultat af test
Ok	{"username": "jhs", "lockID": 1, "password": "test", "keyID": 1, "requestID": "amhzMTIwMTQtMDk-tMTlUMTM6MDI6MTRa"}	{"lockID": 1, "username": "jhs", "requestID": "amhzMTIwMTQtMDk-tMTlUMTM6MDI6MTRa"}	{"lockID": 1, "username": "jhs", "requestID": "amhzMTIwMTQtMDk-tMTlUMTM6MDI6MTRa"}	ok
Forkert password	{"username": "jhs", "lockID": 2, "password": "forkert", "keyID": 1, "requestID": "amhzMjIwMTQtMDk-tMTlUMTM6MDQ6NTJa"}	'User not authenticated'	'User not authenticated'	ok
Ikke adgang	{"username": "jhs", "lockID": 2, "password": "test", "keyID": 1, "requestID": "amhzMjIwMTQt-MDktMTlUMTM6MTI6Mzla"}	'Not authorized'	'Not authorized'	ok

Tabel 6.2: Data for test af auth.

Kapitel 7

Konklusion

Produktet af dette projekt er et system som lever op til nogle, men ikke alle kravene opstillet i afsnit 3. Dette afsnit vil give et overblik over hvad gik godt, hvad gik mindre godt, og hvilke punkter der skal arbejdes videre med, ved en eventuel fortsættelse af projektet.

Dette projekt har undersøgt muligheden for at anvende et Single Sign-On system som backend for en modernisering af den traditionelle dørtelefon. Produktet af dette projekt er en prototype af netop sådan et system. Denne prototype er en implementation af et Single Sign-On system på HTTP, hvor det er muligt for brugere at logge ind på systemer på tværs af et domæne, gennem centrale Identity Providere. Når en bruger er logget ind på dette system har en han mulighed for at åbne døre, alt efter hvilke rettigheder denne bruger har.

I forhold til kravene opstillet i afsnit 3 lever systemet op til de fleste. Datamodellen giver mulighed for at knytte flere nøgler til hver enkelt bruger. Samtidig kan der skelnes mellem disse nøgler, hvilket gør det muligt at inddrage, eller spærre for, hver enkelt nøgle, uden at dette har nogen konsekvenser for andre brugere, eller endda den samme brugers andre nøgler.

Datamodellen giver også mulighed for, i denne implementation gennem Djangos administrative webinterface, at gruppere låse i områder, og på brugerniveau at styre hvilke brugere der har adgang til hvilke områder.

Desværre har det ikke været muligt, indenfor projektets nuværende rammer, at indfri alle de opstillede mål. Den største mangel ved systemet er uden tvivl `pycrypto`s manglende evne til at kryptere nøgler. Dette gør, at systemet ikke kan betragtes som kryptografisk sikkert, da der ikke kan stilles garanti for hverken fortrolighed eller integritet. Udover disse to store mangler, så giver implementationen uafviselighed i forhold til hvem der har åbnet hvilke døre hvornår. Dog mangler logning af dette, så denne funktionalitet kan benyttes ikke i systemets nuværende tilstand.

På trods af disse mangler i implementationen mener jeg at den forudgående research, kombineret med hvad der rent faktisk er lykkedes at implementere, viser at det ønskede system rent faktisk er muligt at implementere. Denne implementation kan betragtes som det første skridt i den rigtige retning mod dette system, men der er dog et stykke vej igen.

Det åbentlyst første skridt i et eventuelt fortsat arbejde med projektet bør klart være at udskifte brugen af`pycrypto` med et andet, mere produktionsklart kryptografisk bibliotek. Samtidig kan det overvejes, om der bør implementeres etablerede standarder, som f.eks. SAML til transport af data, brug af signerede certifikater, og lignende.

Herefter bør det tilstræbes at gøre systemet mere robust. Prototypen udviklet i dette projektforløb fokuserer på at processere en successfuld forespørgsel, og der er derfor nogle scenarier som er mindre komplet implementerede end andre. Dette gælder især fejl i systemet, både internt mellem komponenterne og fejlmelding til klienterne, men kan også kobles på en diskution om de etablerede standarder, nævnt ovenfor.

Bilag A

Wibephone businesscase

Business Case for WibePhone

Din opgangs personlige havelåge…

Indledning:

Wibephone er navnet på den nye standard indenfor dørtelefoner. Med Wibephone forbedres alle funktionerne fra den traditionelle dørtelefon, i et ny og mere indbydende design baseret på touch-skærme, og med en lang række udvidelsesmuligheder, blandt andet til Smart-Phones og boligadministrationssystemer.

For snart 50 år siden blev de første dørtelefoner installeret i Danmark. Her var det ikke muligt at føre en samtale eller se personen der ville ind – blot muligt at åbne hoveddøren. I et stort antal boligforeninger finder man stadig de oprindelige dørtelefoner, der gør det svært at kontrollere, hvem man lukker ind, der er ustabile, har løse forbindelser og ligner ofte uelegante fortidslevninger i ellers ny-renoverede lejligheder. Selv de nyeste dørtelefonsystemer, der installeres i dag, er, med nogle få undtagelser, næsten identiske udgaver af den oprindelige dørtelefon.

Introduktionen af WibePhone gør det let for boligforeninger at skifte til et flot nyt system, der er langt mere stabilt, kræver et minimum af service og hvor de fleste, omend sjældne, problemer kan klares fjernbetjent via nettet. WibePhones tableauer kan afpasses, så de matcher bygningens udseende og let tilpasses de enkelte boligers ønsker til design og specifikationer, så hver bolig kan få tilpasset systemet til præcis dennes ønsker, uafhængigt af de andre boliger i opgangen – Wibephone vil ofte være billigere end traditionelle løsninger – og giver desuden boligforeningerne mulighed for at tjene penge på at installere Wibephone i deres opgange, vha. reklamer på skærmene.

WibePhone – konceptet

WibePhone gør op med 50 års traditionel tankegang om at et dørtelefonsystem skal bestå af en metalplade ved indgangsdøren med uaktuelle navneskilte og en gammeldags skrattende kabelforbundet telefon i hver enkelt lejlighed.

Med WibePhone får brugeren alle de kendte funktioner fra den traditionelle dørtelefon, leveret i et lækkert design, der er mere stabilt og med dets modulære og brugercentrerede design, let kan tilpasses til den enkelte bruges behov og funktionsønsker.

Komponenter og funktioner:


1 Indgangstableau

Indgangstableau: en trådløs tableau/skærm ved indgangsdøren.

- **Udseende:** Skærmen består af en glasplade samt en ramme, der kan komme i et udvalg af materialer i alt fra messing, ædeltræ til klassisk børstet aluminium. Softwaredesignet kan tilpasses til den enkelte boligforening eller opgangs ønsker.
- **I brug:** WibePhone tilbyder de traditionelle funktioner fra den klassiske dørtelefon, men kan også udvides til at inkludere 1- og 2-vejs (video)samtale og man kan vælge at ringe efter en bestemt person i boligen. Derudover er der mulighed for at afspille reklamer, tilpasset den enkelte opgang eller området. (Indtægtskilde for boligforeningen)
- **Standby:** Når det trådløse indgangstableau ikke har været aktiveret i en defineret periode, fx 30 sekunder, går det på standby. Her kan det enten slukke for skærmen, overgå til at vise et udvalgt design eller vise reklamer. Dette uddybes under indtægtskilder.


2 Boligtableau

Boligtableau: i hver bolig opsættes et mindre tableau.

- **Udseende:** Boligtableauet består ligeledes af en glasplade samt individualiseret ramme og designs.

- **I brug:** WibePhones boligtableau tilbyder ligeledes funktionerne fra de traditionelle dørtelefoner, men kan også udvides til at håndtere videosamtaler, design og ringetoner kan tilpasses, så hver beboer i boligen har sin egen ringetone, fungere som interaktiv opslagstavle for boligforening*[1]. Systemet er modulært bygget op, og nye funktioner vil løbende blive tilbudt.

Smartphoneudvidelse: foruden eller som erstatning for et tableau i boligen, kan personerne i boligen vælge at installere WibePhone-applikationen på eksisterende smartphones og computer-tablets med samme funktioner som på boligtableauet.

3 Smartphone App

- **I brug:** Wibephone kan erstatte udgiften til et boligtableau eller kan bruges som back-up, så man kan høre og bruge "dørtelefonen" også selvom man er ude i haven, i et rum langt væk fra hoveddøren. Du kan med andre ord svare din dørtelefon i din bolig i Ballerup, selvom du selv er i Bacelona eller Berlin, hvis du har en smartphone med netadgang. Det åbner for en lang række nye muligheder.
- **Enestående muligheder:** Foruden at supplere indgangstableauet gør en smartphoneudvidelse det muligt at lukke pakkeposten ind, selvom man er på arbejde, så man slipper for at skulle forbi posthuset.
  Derudover sænker det risikoen for tyveri, da en eventuel indbrudstyv, vil tro du er hjemme, når du svarer dørtelefonen, når denne ringer for at se, om der er fri bane. Og du vil have mulighed for at se, hvem der ringer på, selvom du er i sommerhuset.
  Løbende forbedringer: WibePhone er dedikeret til at tilbyde boligforeningerne et unikt produkt, og der vil løbende blive udviklet nye services, såsom tilbud om internetopkobling til hele boligforeningen via Wibephones' internetopkobling, mulighed for gæsteadgang, lettere adgang for fx hjemmeplejepersonale og postbude, der slipper for at bære flere kg nøgler.

Generelt: Da alle komponenterne i boligforeningens system har internetadgang, vil der let, remote, kunne installeres opdateringer til hele systemet

[1] Mulighed for integration med bolignettet www.socialliving.dk

Specifikationer mv.:

Per Bolig:	8" Boligtableau (touchskærm) med trådløs internetadgang (eksisterende ethernet kan ligeledes benyttes) og tilhørende monteringsudstyr til hver bolig. Kan fravælges til fordel for eller udvides med smartphoneapplikation.
Per opgang:	12" Indgangstableau (touchskærm) med trådløs internetadgang (kan køre via eksisterende kabler/ethernet, hvis dette foretrækkes) og tilhørende monteringsudstyr til hver opgang.
Per boligforening:	Nøgle og back-end server, der kører på et standard-setup.

Bilag B

Protocol Security Lab besvarelse

Protocol Security Lab

Jens Henrik Skuldbøl, s070145

October 25th, 2013

The implemented solutions in this assignment were discussed with Brian Lynnerup Pedersen, s042457, during development.

1 Kerberos PKInit

1.1 Describe the protocol

This protocol requires the client to comunicate with two servers, the authentication server, `ath`, and the ticket issuing server, `g`, before gaining access to the requested ressource. First, the client contacts the authentication server, requesting access to the desired ressource. Assuming that the clients identity is verified, the server replies with a session key, and a key the client shares with `g`. This key is encrypted using a key unknown to the client, a key shared between `ath` and `g`, which prevents it being tampered with.

The client then passes this session key on to `g`, along with his identity, encrypted with his and `g`'s shared key. Assuming the client is authorized to access the requested ressource, `g` issues and returns a ticket to the client. This ticket is also encrypted with a key unknown to the client, a key shared between `g` and the host of the ressource, `s`, to prevent tampering.

The client can now send this ticket to the server he wishes to access, which then replies with the requested ressource.

Q – Which of the messages can the client C *decrypt?*
A – Strictly speaking, the client can decrypt all of the messages. He cannot, however, decrypt all parts of some of them, these begin particularly the session key, sent to him from `ath` encrypted using `skag`, the shared key between `ath` and `g`. Also, he cannot decrypt the ticked, sent to him from `g` encrypted using `skgs`, the shared key between `g` and `s`.

Q – Which keys does C *ever learn?*
A – From the beginning, the client knows the following keys

- `pk(C)/inv(pk)c))` – his own public/private keypair
- `pk(ath)` – the public key of `ath`

Communicating with `ath`, the client learns the following keys

- `Ktemp` – A temporary key used by `ath` to encrypt `KCG`
- `KCG` – The shared key between the client and `g`

Communicating with `g`, the client learns `KCS`, the key shared between him and `s`, the server he wishes to access.

Q – How does the protocol prevent illegitimate access?
A – The protocol prevents illigitimate access by verifying both the users identity, and that he is authorized to access the requested ressource. The integrety of this authentication is kept by giving the ticket an expiration date, and by keeping both session keys and tickets encrypted between the authenticating servers, thus preventing them from being tampered with by anyone, including the user.

1.2 Explain the attack

The output of OFMC after finding the attack is included in A.2. As can be seen from the first two messages of the attack, found in lines 17 and 18, the intruder performs a 'man in the middle'-attack between the client and the authentication server. The intruder then subsequently exchanges the identity of the servers with his own, making sure `C` keeps sending messages to him.

The protocol is fixed by encrypting the first message from the `C` to `ath` with `ath`'s public key, thereby making it decipherable by `ath` only.

The fix is verified for two sessions using OFMC, with the output listed in A.3. The first message then becomes

```
C -> ath: {C,g,N1,{T0,N1,hash(C,g,N1)}inv(pk(C))}pk(ath)
```

1.3 Design a variant

The code for the variant is included in A.4. Note that this variant builds on the changes made in the fix listed above. First of all, there are some changes to the types and knowledge. Added to the types are three numbers, `p`, `x` and `y`, and a function, `exp`.

`p` is the public number, known by both `C` and `ath`. `x` and `y` are the secrets of `C` and `ath` respectively, and the exponential function, `exp` is known by both of them.

Changes in communication are in the first message from `C` to `ath`, and the reply. First of all, `C` includes in his message the calculation `exp(p,x)`, which is his half of the Diffie-Hellman key. In its reply, `ath` now includes `exp(p,y)`, but instead of encrypting `KCG` with a temporary key, it now encrypts it with the complete Diffie-Hellman key, `enc(enc(p,x),y)`.

The verifying output of OFMC is listed in **??**.

2 AMP

2.1 Describe and explain the protocol

Q – What security relationship does the parties have initially?
A – Looking at the initial knowledge, `C` knows the public keys of `s` and `RP`, `s` knows the public key of `C`, and `RP` knows the public key of `s`. In the cases where the knowledge of public keys are mirrored, that is, the recipient knows the public key of the sender, and the other way around, a secure channel can be established. This is achieved by the sender encrypting messages in the recipients public key, providing confidentiality, and his own private key, providing authentication.

In cases where the recipient does not know the public key of the sender, only confidentiality is ensured.

In this specific case, a secure channel can be established between `C` and `s`. Additionally, confidential channels can be established between

- `C` and `RP`

- `RP` and `s`

with the sender being the former, and the recipient the latter.

Q – What new security relationship does the protocol try to establish?
A – The protocol tries to establish a secure connection between `C` and `RP` on which they can share a secret, `Data`.

Q – How does the protocol try to achieve this?
A – The protocol attempts this by letting `C` be verified by `s`. After making a request to `RP`, `C` receives a reply containing a request ID, `ReqID`. `C` then sends a message containing the request and ID to `s`, who verifies it, and sends back a confirmation.

`C` can now send a message to `RP` telling it, that he is athenticated by `s`, in return for the `Data`.

2.2 Analyse and explain the attack

`i` imposes as `RP`, thereby intercepting the first message from `C` to `RP`. He then exchanges `Request` and `K` with his own values, and reencrypts using `RP`'s public key, and sends the request.

He then sends the `ReqID` to his own request back to `C` who gets it authenticated from `s`. `C` then returns `i`'s authenticated token to him, which he then sends to `RP` in order to get the `Data` from it.

2.3 Suggest and verify a fix

The attack occurs, as `C` thinks `i` is in fact a trusted instance of the provider, `RP`. Therefor, nothing can be done about `i` intercepting the request from `C`.

Looking at the messages, though, `i` has no way of decrypting the communication between `C` and `s`. These messages are simply relayed through `i`. A fix is therefor implemented by including `C`'s original `Request` and `ReqID` in the messages between these two parties. Now `i` has no way of slipping in his forged `Request` and `ReqID`, and only the original `Request` by `C` is verified by `s`.

The changes occur in lines 17, 18 and 20, and the resulting code can be found in B.3, with the verified output in B.4.

2.4 Revoke *S*' trustworthy status

The attack in the previously fixed protocol, when `s` is no longer considered honest, happens for the same reason `i` was able to intercept the original request previously. `S` is no longer guaranteed to be who he says he is, and therefor, `C` sends the request to an intruder, thinking he is an honest provider.

Seeing as neither `RP` nor `S` can be authenticated in the new protocol, I believe no fix can be made with the given goals and initial knowledge.

A PKInit

A.1 Handout code

```
Protocol: Kerberos_PKINIT

Types: Agent C,ath,g,s;
       Number N1,N2,T0,T1,T2,Payload,tag;
       Function pk,hash,sk;
       Symmetric_key KCG,KCS,Ktemp,skag,skgs

Knowledge: C: C,ath,g,s,pk(ath),pk(a),pk,inv(pk(C)),hash,tag;
           ath: ath,g,pk,inv(pk(ath)),hash,skag,tag;
           g: ath,g,skag,skgs,hash,tag;
           s: g,s,skgs,hash,tag

where C!=ath

Actions:
C -> ath: C,g,N1,{T0,N1,hash(C,g,N1)}inv(pk(C))

ath -> C: C,
          ({|ath,C,g,KCG,T1|}skag),
          ({|g,KCG,T1,N1|}Ktemp),
          { tag,{Ktemp}inv(pk(ath))}pk(C)

C -> g: s,N2,
          ({|ath,C,g,KCG,T1|}skag),
          ({|C,T1|}KCG)

g -> C: C,
          ({|C,s,KCS,T2|}skgs),
          {|s,KCS,T2,N2|}KCG

C -> s: ({|C,s,KCS,T2|}skgs),
          {|C,hash(T2)|}KCS

s -> C: ({|hash(T2)|}KCS), {|tag,Payload|}KCS

Goals:
s *->* C: Payload
```

A.2 Attack trace

```
Open-Source Fixedpoint Model-Checker version 2013a
INPUT:
    PKINIT.AnB
SUMMARY:
  ATTACK_FOUND
GOAL:
  weak_auth
BACKEND:
  Open-Source Fixedpoint Model-Checker version 2013a
STATISTICS:
  TIME 908 ms
  parseTime 8 ms
```

```
13    visitedNodes: 13 nodes
14    depth: 4 plies
15
16  ATTACK TRACE:
17  (x701,1) -> i: x701,g,N1(1),{T0(1),N1(1),hash(x701,g,N1(1))}_inv(pk
        (x701))
18  i -> (ath,1): i,g,N1(1),{x304,N1(1),hash(i,g,N1(1))}_inv(pk(i))
19  (ath,1) -> i: i,{|ath,i,g,KCG(2),T1(2)|}_skag,{|g,KCG(2),T1(2),N1
        (1)|}_Ktemp(2),{tag,{Ktemp(2)}_inv(pk(ath))}_(pk(i))
20  i -> (x701,1): x701,x406,{|g,KCG(2),T1(2),N1(1)|}_Ktemp(2),{tag,{
        Ktemp(2)}_inv(pk(ath))}_(pk(x701))
21  (x701,1) -> i: s,N2(3),x406,{|x701,T1(2)|}_KCG(2)
22  i -> (x701,1): x701,x511,{|s,x512,x513,N2(3)|}_KCG(2)
23  (x701,1) -> i: x511,{|x701,hash(x513)|}_x512
24  i -> (x701,1): {|hash(x513)|}_x512,{|tag,x614|}_x512
25
26
27  % Reached State:
28  %
29  % request(x701,s,pCsPayload,x614,1)
30  % state_rC(x701,4,tag,hash,inv(pk(x701)),pk,pk(a),pk(ath),s,g,ath,
        N1(1),T0(1),x701,g,N1(1),{T0(1),N1(1),hash(x701,g,N1(1))}_inv(
        pk(x701)),x406,T1(2),KCG(2),{|g,KCG(2),T1(2),N1(1)|}_Ktemp(2),{
        T0(1),N1(1),hash(x701,g,N1(1))}_inv(pk(x701)),Ktemp(2),{Ktemp
        (2)}_inv(pk(ath)),{tag,{Ktemp(2)}_inv(pk(ath))}_(pk(x701)),x701
        ,x406,{|g,KCG(2),T1(2),N1(1)|}_Ktemp(2),{tag,{Ktemp(2)}_inv(pk(
        ath))}_(pk(x701)),N2(3),s,N2(3),x406,{|x701,T1(2)|}_KCG(2),x511
        ,{|x701,T1(2)|}_KCG(2),x513,x512,{|s,x512,x513,N2(3)|}_KCG(2),
        x701,x511,{|s,x512,x513,N2(3)|}_KCG(2),x511,{|x701,hash(x513)|}
        _x512,{|x701,hash(x513)|}_x512,x614,{|tag,x614|}_x512,{|hash(
        x513)|}_x512,{|hash(x513)|}_x512,{|tag,x614|}_x512,1)
31  % state_rg(g,0,tag,hash,skgs,skag,ath,1)
32  % state_rs(s,0,tag,hash,skgs,g,1)
33  % state_rath(ath,1,tag,skag,hash,inv(pk(ath)),pk,g,hash(i,g,N1(1)),
        N1(1),x304,{x304,N1(1),hash(i,g,N1(1))}_inv(pk(i)),i,i,g,N1(1)
        ,{x304,N1(1),hash(i,g,N1(1))}_inv(pk(i)),KCG(2),T1(2),Ktemp(2),
        i,{|ath,i,g,KCG(2),T1(2)|}_skag,{|g,KCG(2),T1(2),N1(1)|}_Ktemp
        (2),{tag,{Ktemp(2)}_inv(pk(ath))}_(pk(i)),1)
```

A.3 Fix verification

```
1  Open-Source Fixedpoint Model-Checker version 2013a
2  INPUT:
3      PKINIT_fix.AnB
4  SUMMARY:
5    NO_ATTACK_FOUND
6  GOAL:
7    as specified
8  DETAILS:
9    BOUNDED_NUMBER_OF_SESSIONS
10 BACKEND:
11   Open-Source Fixedpoint Model-Checker version 2013a
12 STATISTICS:
13   TIME 11424 ms
14   parseTime 8 ms
15   visitedNodes: 7333 nodes
16   depth: 14 plies
```

A.4 Diffie-Hellman variant

```
Protocol: Kerberos_PKINIT

Types: Agent C,ath,g,s;
       Number N1,N2,T0,T1,T2,Payload,tag,p,x,y;
       Function pk,hash,sk,exp;
       Symmetric_key KCG,KCS,Ktemp,skag,skgs

Knowledge: C: C,ath,g,s,pk(ath),pk(a),pk,inv(pk(C)),hash,tag,p,x,
    exp;
           ath: ath,g,pk,inv(pk(ath)),hash,skag,tag,p,y,exp;
           g: ath,g,skag,skgs,hash,tag;
           s: g,s,skgs,hash,tag

where C!=ath

Actions:
C -> ath: {C,g,N1,{T0,N1,hash(C,g,N1), exp(p,x)}inv(pk(C))}pk(ath)

ath -> C: C,exp(p,y),
          ({|ath,C,g,KCG,T1|}skag),
          ({|g,KCG,T1,N1|}exp(exp(p,x),y))

C -> g: s,N2,
          ({|ath,C,g,KCG,T1|}skag),
          ({|C,T1|}KCG)

g -> C: C,
          ({|C,s,KCS,T2|}skgs),
          {|s,KCS,T2,N2|}KCG

C -> s: ({|C,s,KCS,T2|}skgs),
          {|C,hash(T2)|}KCS

s -> C: ({|hash(T2)|}KCS), {|tag,Payload|}KCS

Goals:
s *->* C: Payload
```

A.5 Variant verification

```
Open-Source Fixedpoint Model-Checker version 2013a
INPUT:
    PKINIT_dh.AnB
SUMMARY:
  NO_ATTACK_FOUND
GOAL:
  as specified
DETAILS:
  BOUNDED_NUMBER_OF_SESSIONS
BACKEND:
  Open-Source Fixedpoint Model-Checker version 2013a
STATISTICS:
  TIME 6028 ms
  parseTime 8 ms
  visitedNodes: 3187 nodes
```

```
16   depth: 8 plies
```

B AMP

B.1 Handout code

```
1  Protocol: AMP
2
3  Types: Agent C,s,RP;
4         Number Request,ReqID,Data;
5         PublicKey K
6
7  Knowledge:
8         C: C,s,RP,pk(s),pk(C),inv(pk(C)),pk(RP);
9         s: C,s,pk(s),inv(pk(s)),pk(C);
10        RP: s,RP,pk(s),pk(RP),inv(pk(RP))
11
12 Actions:
13
14 C->RP: {C,RP,Request,K}pk(RP)
15 RP->C: {C,s,RP,ReqID,Request}K
16
17 C->s:  { {C,s,RP,ReqID,Request}inv(pk(C)) }pk(s)
18
19 s->C:  { {C,s}inv(pk(s)) }pk(C)
20
21 C->RP: {{C,s}inv(pk(s))}pk(RP)
22 RP->C: {Data}K
23
24 Goals:
25
26 RP authenticates C on Request
27 C authenticates RP on Data
28 Data secret between RP,C
```

B.2 Attack trace

```
1  Open-Source Fixedpoint Model-Checker version 2013a
2  INPUT:
3     AMP.AnB
4  SUMMARY:
5    ATTACK_FOUND
6  GOAL:
7     weak_auth
8  BACKEND:
9    Open-Source Fixedpoint Model-Checker version 2013a
10 STATISTICS:
11   TIME 2488 ms
12   parseTime 8 ms
13   visitedNodes: 1369 nodes
14   depth: 5 plies
15
16 ATTACK TRACE:
17 (x802,1) -> i: {x802,i,Request(1),K(1)}_(pk(i))
18 i -> (x801,1): {x802,x801,x306,x307}_(pk(x801))
19 (x801,1) -> i: {x802,s,x801,ReqID(2),x306}_x307
```

```
20 i -> (x802,1): {x802,s,i,x506,Request(1)}_K(1)
21 (x802,1) -> i: {{x802,s,i,x506,Request(1)}_inv(pk(x802))}_(pk(s))
22 i -> (s,1): {{x802,s,i,x506,Request(1)}_inv(pk(x802))}_(pk(s))
23 (s,1) -> i: {{x802,s}_inv(pk(s))}_(pk(x802))
24 i -> (x802,1): {{x802,s}_inv(pk(s))}_(pk(x802))
25 (x802,1) -> i: {{x802,s}_inv(pk(s))}_(pk(i))
26 i -> (x801,1): {{x802,s}_inv(pk(s))}_(pk(x801))
27 (x801,1) -> i: {Data(6)}_x307
28
29
30 % Reached State:
31 %
32 % request(x801,x802,pRPCRequest,x306,1)
33 % state_rRP(x801,2,inv(pk(x801)),pk(x801),pk(s),s,x307,x306,x802,{
       x802,x801,x306,x307}_(pk(x801)),ReqID(2),{x802,s,x801,ReqID(2),
       x306}_x307,{x802,s}_inv(pk(s)),{{x802,s}_inv(pk(s))}_(pk(x801))
       ,Data(6),{Data(6)}_x307,1)
34 % witness(x801,x802,pCRPData,Data(6))
35 % secrets(Data(6),secrecyset(x801,1,pData),i)
36 % contains(secrecyset(x801,1,pData),x801)
37 % contains(secrecyset(x801,1,pData),x802)
38 % state_rs(s,1,pk(x802),inv(pk(s)),pk(s),x802,Request(1),x506,i,{
       x802,s,i,x506,Request(1)}_inv(pk(x802)),{{x802,s,i,x506,Request
       (1)}_inv(pk(x802))}_(pk(s)),{{x802,s}_inv(pk(s))}_(pk(x802)),1)
39 % state_rC(x802,3,pk(i),inv(pk(x802)),pk(x802),pk(s),i,s,Request(1)
       ,K(1),inv(K(1)),{x802,i,Request(1),K(1)}_(pk(i)),x506,{x802,s,i
       ,x506,Request(1)}_K(1),{{x802,s,i,x506,Request(1)}_inv(pk(x802)
       )}_(pk(s)),{x802,s}_inv(pk(s)),{{x802,s}_inv(pk(s))}_(pk(x802))
       ,{{x802,s}_inv(pk(s))}_(pk(i)),1)
40 % state_rC(x28,0,pk(x801),inv(pk(x28)),pk(x28),pk(s),x801,s,2)
41 % state_rRP(x32,0,inv(pk(x32)),pk(x32),pk(s),s,2)
42 % state_rs(s,0,pk(x31),inv(pk(s)),pk(s),x31,2)
43 % witness(x802,i,pRPCRequest,Request(1))
```

B.3 Fixed AMP

```
1  Protocol: AMP
2
3  Types: Agent C,s,RP;
4         Number Request,ReqID,Data;
5         PublicKey K
6
7  Knowledge:
8         C: C,s,RP,pk(s),pk(C),inv(pk(C)),pk(RP);
9         s: C,s,pk(s),inv(pk(s)),pk(C);
10        RP: s,RP,pk(s),pk(RP),inv(pk(RP))
11
12 Actions:
13
14 C->RP: { C,RP,Request,K }pk(RP)
15 RP->C: { C,s,RP,ReqID,Request }K
16
17 C->s:  { { C,s,RP,ReqID,Request }inv(pk(C)) }pk(s)
18 s->C:  { { C,s,Request,ReqID }inv(pk(s)) }pk(C)
19
20 C->RP: { { C,s,Request,ReqID }inv(pk(s)) }pk(RP)
21 RP->C: { Data }K
```

```
22
23 Goals:
24
25 RP authenticates C on Request
26 C authenticates RP on Data
27 Data secret between RP,C
```

B.4 Fix Verification

```
1  Open-Source Fixedpoint Model-Checker version 2013a
2  INPUT:
3      AMP.AnB
4  SUMMARY:
5    NO_ATTACK_FOUND
6  GOAL:
7    as specified
8  DETAILS:
9    BOUNDED_NUMBER_OF_SESSIONS
10 BACKEND:
11   Open-Source Fixedpoint Model-Checker version 2013a
12 STATISTICS:
13   TIME 13560 ms
14   parseTime 8 ms
15   visitedNodes: 10957 nodes
16   depth: 14 plies
```

Litteratur

[1] Madsen, Paul, *Liberty Alliance Project White Paper*, 5. december 2005, http://www.projectliberty.org/liberty/content/download/387/2720/file/Liberty_Federated_Social_Identity.pdf

[2] Møllerhøj, Jakob, *Datatilsynet: CPR-nummeret er ikke en følsom personoplysning, men du må alligevel ikke offentliggøre andres*, Version2.dk, 24. juni 2014, http://www.version2.dk/artikel/cpr-nummeret-er-ikke-en-foelsom-personoplysninger-men-du-maa-alligevel-ikke-o

[3] Mortensen, Henrik Nordstrøm, *Nu kan du låse hoveddøren med din mobil*, Ing.dk, 20. august 2013, http://ing.dk/artikel/nu-kan-du-laase-hoveddoeren-med-din-mobil-161074

[4] https://www.dlitz.net/software/pycrypto/

[5] Mödersheim, Sebastian og Vibanò, Luca, *The Open-source Fixed-point Model Checker for Symbolic Analysis of Security Protocols*, http://imm.dtu.dk/~samo/ofmc-fosad.pdf

[6] Pfleeger, Charles P. og Pfleeger, Shari Lawrence, *Security in Computing*, 4th edition, Prentice Hall, 2006