Danmarks Tekniske Universitet

Kryptologisk Legestue

DTU Informatik, Bachelor Projekt 2011, IMM-B.Sc.-2011-20

Noel Vang (s082961) Martin Metz (s082713)

27. Juni 2011
Bachelor projekt 2011
DTU, Institut for Informatik og Matematisk Modellering
Vejleder: Christian D. Jensen cdj@imm.dtu.dk

This page intentionally left blank

Abstract

Denne rapport omhandler processen omkring udviklingen af en Kryptologisk legestue. Der er en gennemgang af de overordnede design principper og en redgørelse for hvilke kryptologiske muligheder der er blevet valgt.

Executive Summary

This report is about the development of a cryptographic playground. The idea with this playground is to give younger engineering student a better understanding of different kinds of encryption. Our primary goal is to give the user a better practical understanding on how the different algorithms behave under various situations. We have tried with our best effort to keep the report as simple as possible and stay above code level. We have also tried to justify the steps we have taken and tried to explain it as good as possible. In this report you will also find a discussion about the chosen cryptographic functionality that was chosen but also a discussion about the non-cryptographic functionality. Since this is a program that should be used, it has to be bulletproof and not crash. So alot of effort also went into securing the uptime of the program.

Preface

Denne rapport er fremkommet som led i et bachelor projekt. Det er skrevet i samarbejde med institut for Informatik og Matematisk Modellering på Danmarks Tekniske Universitet i Kgs. Lyngby. Projektet har foregået over perioden fra Februar 2011 til Juni 2011 og har haft en ECTS pointbelastning på 20 points.
Vores vejleder på projektet har været Lektor Christian Dam Jensen fra Institut for Informatik og Matematisk Modellering på Danmarks Tekniske Universitet, Lyngby.

Lyngby, Juni 2011

Martin Metz Noel Vang

Indhold

Figurer

This page intentionally left blank

This page intentionally left blank

1 Introduktion

Kryptografi har været praktiseret i mange tusinde år. Helt tilbage til det gamle egypten har fundet rester af at folk også dengang benyttede sig af kode sprog. Det var dog først da computeren for alvor tog sit indtog at kryptografi for alvor blev nyttigt. Med computeren var det muligt at udføre krypterings opgaver, som ville tage meget lang tid eller være umuligt for et menneske at udføre. Computeren udvikler sig hele tiden og bliver bedre og bedre. Det stiller også større krav til de krypteringsmetoder der bliver anvendt. Ting som engang anses for at være umulige at bryde med ren computer kraft, kan idag sagtens brydes. De fleste krypteringsalgoritmer har et solidt grundlag i matematikken og har igennem flere år været udsat for offentlig granskning uden at det har først til at de er blevet brudt. Nogle har dog også måtte lade livet eller ihvertfald midlertidigt indtil en ny revision så dagens lys. Vi lever idag i en digital tidsalder hvor næsten alt kan foregå over en computer. Dette stiller høje krav til den sikkerhed der skal være omkring brugen af computer. Vi har i dag, bankadgang, adgang til det offentlige og en masse private ting på vores computere. som for eksempel breve, emails og billeder og meget mere. Alle disse ting skulle jo helst være sikret og det kan man gøre igennem kryptografi. Men det er ikke bare computere der har brug for denne sikkerhed. Mobiltelefoner er også blevet så smarte idag og så fyldt med funktionalitet og personlige oplysninger at også her er det relevant at tænke på sikkerheden. Men for mange mennesker er det med kryptografi et lidt diffust begreb og langt de fleste ønsker bare t det skal virke uden at tænke for meget over det. Faktisk ville de helst slet ikke tænke på det overhovedet, men bare regne med at ens data er sikkert. Det er vores opgave som ingeniører at få integreret disse løsninger på en effektiv og smart måde. Men det er ikke alle ingeniører der har lige stor erfaring med hvor lang tid de forskellige algoritmer tager, og i hvilken sammenhæng en given konfiguration er mest optimal.

1.1 Målsætning for projektet

Vi har derfor udviklet et program som kan kryptere på en masse forskellige måder og med et stort udbud af indstillinger, man kan justere på. Meningen er så at man kan få en bedre ide om, hvordan en given konfiguration kører. Vi har prøvet at få integreret muligheden for netop at sammenligne de forskellige indstillinger med hinanden. Vores primære mål har været, at når brugerne har brugt programmet har de en bedre praktisk forståelse for, hvad det vil sige at kryptere noget med symmetrisk kryptering i forhold til asymmetrisk. Også omkring nøgle størrelse håber vi at programmet kan bidrage positivt til deres forståelse af hvor lang tid det kan tage hvis man vælger nogle ikke optimale indstillinger. Det kræver derfor også at vi får en god portion funktionalitet ind i vores program, ikke kun krypteringsmæssigt, men i høj grad også den funktionalitet der gør programmet brugbart.

1.2 Rapport struktur

I rapporten har vi bevidst valgt ikke at gå helt ned i dybden med hvordan koden er blevet skrevet men vi har prøvet at fokusere på de overordnede elementer i koden som ikke nødvendigvis behøver at være sprog-specifikke. Vi har bygget rapporten sådan op at der i første sektion er en introduktion til projektet sammen med vores målsætninger. Derefter bliver den kravspecifikation vi har udviklet og arbejdet udfra gennemgået i tredje sektion. Fjerde sektion indeholder de overordnede design mønstre der er blevet brugt samt en kort gennemgang af de tanker som ligger bag vores hovedbestandele. Herefter vil 5. sektion omhandle det kryptografiske grundlag der udgør vores program samt en kort gennemgang af hvilke krypteringsfunktionalitet der tilbydes i programmet. 6 sektion er et afsnit de prøver at gå lidt nærmere vores kode uden at gå i for mange detajler, men dog nok til at man kan få en ide om hvordan nogle af modulerne virker. Vi har også et afsnit om den grafiske brugergrænse vi har implementeret og de værktøjer og ideer der er blevet brugt til udviklingen heraf. Vi slutter af med en lille smule omkring testning af programmet, også selvfølgelig en konklusion.

2 Udviklingsproces

Vi har under udviklings processen måtte justere på vores tidsplan en lille bitte smule, og på vores første kravspecifikation. Dette skyldes blandt andet at det har været lidt svært at vurdere hvor lang tid nogle af del-elementerne burde have taget. Til selve udviklingen af programmet og rapportskrivningen har vi brugt version styrings software, som har gjort det nemmere for os at arbejde på projektet samtidigt og derfor arbejde mere effektivt. Vi har også prøvet at udvikle programmet så det består af små del-moduler som er uafhængige af hinanden så vidt muligt, så man derfor har kunnet udvikle en del af programmet og teste om den giver det rette output afhængigt af input. Dette stemmer også overens med at vi har prøvet at arbejde så objektorienteret som muligt. Fordi vi har udviklet koden på den måde og hele tiden bestræbet os på at have virkende kode i form af små moduler kan det godt siges at vi har benyttet os af spiral modellen under udviklingen af programmet.

3 Kravspecifikation

Vi tog udgangs punkt i projekt beskrivelsen, og lavede en kravspecifikation, som vi mente kunne leve op til de rammer, som projekt beskrivelsen gav os.

3.1 Terminologi

Algoritmer: Man skal kunne vælge en algoritme inde for kategorierne symmetrisk kryptering, asymmetrisk kryptering og envejskryptering. Under symmetrisk, skal man kunne vælge forskellige blok og stream cifre.

Modes: Ved symmetrisk blok kryptering, kan man vælge forskelle indstillinger på, hvordan krypteringen skal foregå. Det kan blandt andet være Cipher-block chaining mode (CBC), hvor man binder alle de krypterede blokke sammen, ved at gøre dem afhængige af den foregående blok.

Paddings: Ved symmetrisk blok kryptering, skal man kunne vælge forskellige paddings indstillinger. Padding er en måde at fylde din sidste blok ud hvis der ikke er nok data til en hel blok.

Parametre: Det skal også være muligt at justere blok størrelse samt nøglestørrelse til den valgte algoritme, inden for de grænser som algoritmen tillader.

Nøgler og initialitionsvektorer: Nøgle er den "hemmelighed" der skal bruges for at genskabe eller ulæselig gøre data. Nogle former for algoritmer benytter sig også af det der kaldes en initialitionsvektor. Den bruges, når en algoritme benytter sig at data fra den tidligere blok til at kryptere den nuværende. På den første blok, findes der ikke nogen tidligere blok og man anvender derfor en initialitionsvektor, der er samme størrelse som blokstørrelsen på den give algoritme. Som input til nøgler og initialitionsvektorer skal det være muligt for programmet at kunne generere dem til brugeren, men det skal også være muligt at taste en streng ind som bliver lavet om til en række bytes som man kan benytte som nøgle. Det samme gælder for initialitionsvektorer for de indstillinger, hvor den er påkrævet. Da man også skal kunne auto-genere en nøgle/initialitionsvektor skal man også kunne gemme og indlæse en tidligere gemt nøgle/initialitionsvektor. Dette gælder både for symmetriske og asymmetriske algoritmer.

Fejlhåndtering: Da det er et program som skal bruges aktivt er det vigtigt for os at programmet er så robust at det ikke kan gå ned og kaste en exception. Vi vil derfor også have at alt input bliver undersøgt for at se om det givne data giver mening. Dette gælder også for de konfigurationer som brugeren kan lave. Findes der en fejl skal brugeren informeres omkring fejlen og så vidt muligt om hvor input fejlen er opstået.

Trådbaseret i forskellige moduler: Da nøgle generering godt kan tage tid afhængigt af de valgte parametre, ønskes denne proces implementeret i med tråde. Det samme gør sig gældende for afviklingen af en krypterings konfiguration, da vi gerne skal kunne køre flere simultant.

Filer: Det skal være muligt at indlæse vilkårlige filer til kryptering. Dette gælder også størrelsen, altså systemet skal kunne håndtere vilkårligt store filer. Det skal dog også være muligt, at bruge en auto generet fil til krypteringen. Og endvidere skal man kunne kryptere en given tekst.

3.2 Minimumskrav

Som minimums krav ville vi have en grafisk brugergrænseflade, som gjorde det nemt at benytte sig af programmet. Vi ønskede også at man skulle kunne arbejde med en række forskellige former for kryptering, blandt andet symmetrisk og asymmetrisk krypterings algoritmer samt envejskrypteringer de såkaldte hash funktioner. Vi ville gerne fokusere på de mest brugte algoritmer, da programmet skal bruges til at give en bedre forståelse om, hvilken betydning valg af algoritmer, og nøgle størrelser vil have for eksempelvis køretiden. Der lægges også stor vægt på at programmet er så platformsuafhængigt som muligt da vi ønsker at programmet ikke skal være bundet til kun en platform. Vi vil derfor udvikle programmet i Java.

3.3 Ønskelige krav i prioriteret rækkefølge

Arbejde med flere objekter: Da vi ønsker at man med programmet kan lege med flere forskellige indstillinger, skal programmet kunne gemme en konfiguration, så det er muligt at have flere forskellige indstillinger klar til at blive kørt og sammenlignet simultant.

Konsol output: Der skal implementeres en konsol, som gør det muligt at give information til brugeren omkring relevante hændelser i programmet, og information omkring køretider og nøglegenerering m.m.

Redigering og information omkring en konfiguration: Det skal også være muligt at redigere i en tidligere gemt konfiguration, så man for eksempel kan skifte nøgle længde eller andet. Det skal også være muligt at se informationer omkring den givne konfiguration, såsom sidste køretid, nøglelængde osv.

Gemme programmets tilstand: At man kan gemme programmets nuværende tilstand (inklusiv gemte konfigurationer, konsol indstillinger, og information omkring kørte konfigurationer) og indlæse den igen. Det betyder at man kan arbejde videre på sine konfigurationer senere.

Simpel guide: Man skal kunne få information omkring alle de forskellige kategorier af algoritmer, modes og padding. Det tænkes at det bare skal være simpel information omkring de forskellige indstillinger og algoritmer.

Flere krypterings former: Vi kunne godt tænke os at have flere muligheder for kryptering. Det kunne blandt andet være noget beskedvalidering (MAC), signering af data, og andre spændende former. Det kunne også være interessant at kigge på ECC-kryptering.

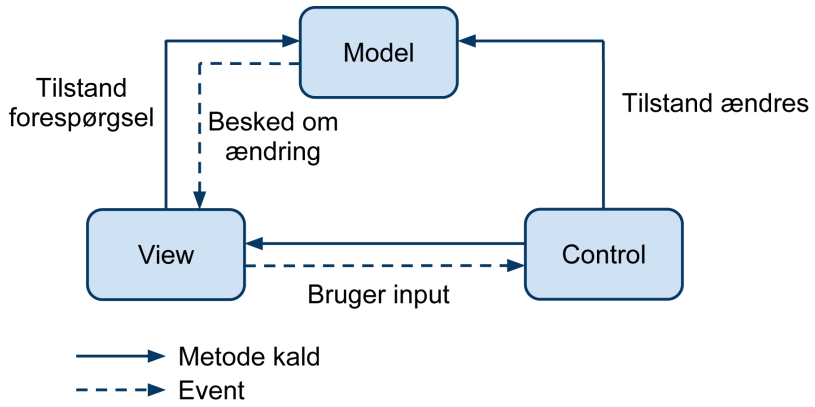
Spil: Et spil, hvor man skulle kunne konkurrere mod andre omkring sin viden om kryptologi. Få point efter hvor rigtig ens svar er, og hvor hurtig man svare.

4 Overordnet programstruktur

Det har været vigtigt for os igennem processen at have struktureret kode, og vi har derfor valgt at benytte os af nogle forskellige design mønstre. Et design mønster er et relevant og rigtig godt hjælpemiddel til at udvikle, og lave arkitekturen af et programmerings projekt. Når man bruger et design mønster kan det hjælpe med at løse nogle af de forskellige problemer der kan opstå, på en velkendt og gennem tænkt måde. På denne måde har man en stuktur og ensartethed i sin implementation, og implementeringen bliver lettere at forstå for en person, som ikke har været med til at udvikle på koden. Det gør også at man nemmere kan finde sine fejl, fordi der er lagt en overordnet ramme, som man arbejder ud fra og derfor nemmere kan have en god ide om, hvor man skal lede efter en given fejl i implementeringen, hvis sådan en skulle opstå.

4.1 Model-View-Control

Model-view-control er ideelt til opbygningen af vores program. Det er fordi det giver mening netop at opdele funktionaliteten som en model, der tager hånd om alle beregninger, opbevaring af data og datastrukturer m.m. Derudover skal vi også have noget, der kan fremvise vores model på en ordenligt måde og der passer vores bruger grænseflade ind. Den vil blive betragtet som vores view. Det er her, hvor brugeren kan udfolde sig i og gennem den kommunikere med den bagved liggende model for at få de ønskede resultater frem. For at binde de to del-elementer sammen bliver der brugt en controller. Controlleren varetager de events som brugeren generer, og sørger for at modellen tager de skridt, der er nødvendige for at imødekomme brugerens forespørgsel [GOF]. Når modellen så har udført sin opgave, og har noget klar til vores view, sender modellen en besked til view'et om, at der er nyt data, som skal vises. Dette giver en skarp opdeling, og gør det muligt nemt helt at skifte view eller have to forskellige implementationer af det. Det gør også at man har en bedre ide om, hvor man skal kigge, hvis man leder efter noget specielt i koden. Ved at bruge dette mønster, kan implementationen blive mere modul baseret. Det gør det også nemmere at arbejdede med flere tråde i modellen for at få en bedre ydeevne. Endnu en fordel ved at arbejde med en modulbaseret implementation, er at man nemt kan modificere et modul, eller vælge at genbruge det [GUI].



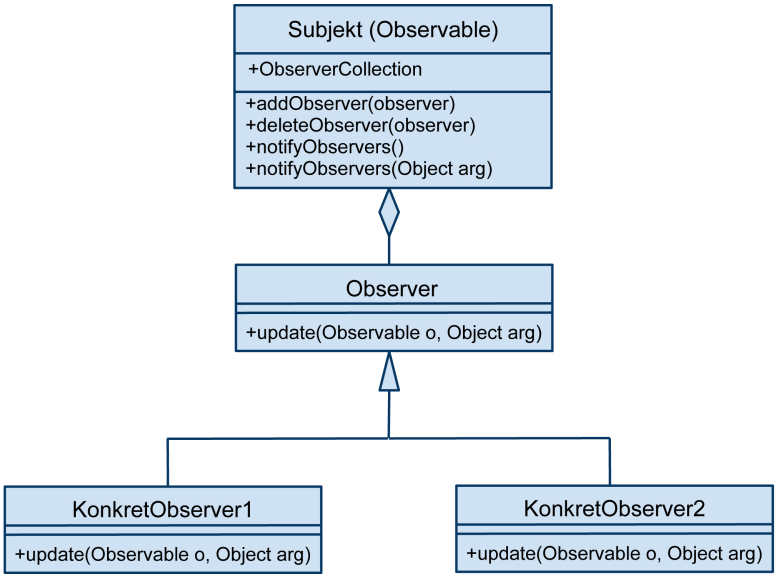
Figur 1: Diagram over Model-View-Control

Som det ses på fig. 1 sker der det at brugeren generer en forespørgsel igennem interaktion med vores view. Den bliver opfanget af controlleren, der kan give programmets grafiske del besked på, hvad den skal opdatere i sit view. Dog fungerer det i vores implementation langt overvejende ved at controlleren fortæller modellen at den skal ændre tilstand. Når modellen har udført sin opgave fortæller modellen vores view, hvor der findes ændringer, og beder view'et om at hente den nye data ud af modellen og fremvise den i viewet.

4.2 Observer

For at kunne kommunikere effektivt imellem de forskellige dele af vores program, har vi valgt at benytte os af et design mønster som kaldes observer pattern.

Observer pattern fungerer på den måde, at man har et objekt der er observable, som også kan kaldes for subjekt, der vedligeholder en liste over, hvilke objekter der er observers på dette objekt. Når der så skal gives en besked sender subjektet en besked ud til alle i den lister over observers.



Figur 2: Struktur i Observer mønster

Som det fremgår af fig. 2 har observer interfacet en kompositions pil til subjektet, der betyder af den afhænger og har et tilhørsforhold til det. Det ses yderligere at de klasser, der skal være konkrete observers har en generalisationspil til det Observer-interface ,som de implementerer. I vores implementation er det vores model, der er subjektet og derfor den der giver besked videre, når der er ændringer i modellens tilstand. Alle de forskellige større moduler i vores view har så implementeret observer interfacet, og har derfor en metode, der bliver kørt når subjektet sender en besked. På denne måde er det nemt at fortælle de respektive dele af view'et, at de skal hente ny data ud fra modellen og gentegne sig selv. Da man kan sende et hvilket som helst objekt med fra subjektet valgte vi at bruge et ID objekt vi har lavet, der fortæller oberserverne hvem modtageren er og på den måde kan vi altså nøjes med kun at opdatere de ting i viewet som rent faktisk har ændret tilstand.

Det betyder at ikke hele vores view skal opdateres, hver gang der er en lille ændring i modellen, da der er blevet taget hånd om, hvornår hver enkel modul skal opdateres.

4.3 Bagved liggende model

Da vi startede med at udvikle programmet var vi allerede klar over at vi ville benytte os af de to førnævnte design mønstre. Det stemte fint overens med vores ide om at starte med at udvikle modellen, som et simpelt konsol styret program, altså hele modellen med simpelt input enten fra konsollen eller fra indtastning inden kompileringen af koden. På den måde var det også helt sikkert at modellen ikke ville have nogen uønskede associationer til vores

view eller controlleren. Det gjorde det også nemmere at programmere bruger grænsefladen fordi alt den bagved liggende krypteringslogik, beregning og opbevaring af data var lavet og testet. Da modellen var færdig og stort set gennem testet, blev der tilføjet et view, også kaldet brugergrænseflade, og diverse relevante kontroller dele til at kommunikere imellem vores view og model. Denne tilgang betyder at de fleste fejl fra modellen kunne rettes og udelukkes inden vi begyndte på view delen og derfor stort set kun skulle fejlfinde i viewet.

4.4 Eventbaseret arkitektur

Da vi har valgt at benytte os af Java's Swing API til at udvikle vores brugergrænseflade er det også relevant lige at nævne at dette API er event baseret. Det er en software arkitektur, der bygger på produktion, opfangning, reaktion og sletning af disse events. Et event kan blive skabt, når man klikker med musen, eller når man trykker en keyboardtast. Det er som regel ikke en god ide at lytte på disse lav niveau events, der bliver skabt, men hellere på et lidt højere niveau. Altså i stedet for at lytte på om museknappen bliver trykket ned, skal man hellere lytte på om for eksempel en knap skifter tilstand fra ikke-trykket til trykket. Tilstandsskift i view komponenterne genererer også events, dog på et lidt højere niveau, da de også fortæller noget, om hvilken komponent der udløste den samt hvilken event der er tale om. De events bliver håndteret af vores control del af programmet, som sørger events'ne bliver omsat til forespørgsler til modellen.

5 Kryptografi

5.1 Valg af bibliotek

Der er et udvalg af krypterings biblioteker på internettet [LIB]. Forudsætningen for valget af hvilket bibliotek, som blev brugt i programmet, var at det skulle opfylde vise krav. Kravene til biblioteket var at det skulle findes til Java da det er vores udviklingssprog, men også at biblioteket skulle indeholde alle de gængse algoritmer.

Det ville være muligt at bruge flere biblioteker, men for at kunne lave programmet så modul baseret på en ukompliceret måde, blev vi enige om kun at bruge et krypterings bibliotek.

Bouncy Castle og SUN's kryptering biblioteker var de to kandidater, som blev overvejet. Valget faldt på Bouncy Castle, da SUN's implementation er underlagt en restriktion fra den amerikanske lovgivning, med hensyn til deres maksimale nøgle størrelser. [RES] Det er dog muligt at benytte sig af en anden licens fil der ophæver de restriktioner. Vi følte også at vi havde det bedre med at implementationen fra Bouncy Castle, da den er open source og at man havde langt større kontrol over hvordan man kunne bruge bibliotekerne.

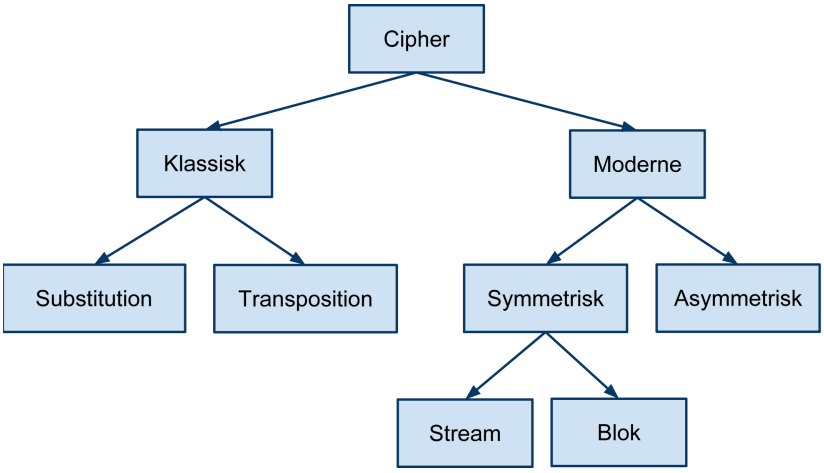
Bouncy Castle kan bruges på to måder, man kan bruge det som provider, hvor der er skrevet en wrapper, så man benytter biblioteket på præcis sammen måde som man ville med JCE. Ulempen det er at man ikke har helt styr på hvad der sker i det underliggende lag, man ved bare at man kalder en funktion, men ikke om den bruger de rigtige parametre. Og hvis algoritmen giver en fejl kan det være svært at gennemskue, hvor den fejl ligger. Den anden måde man kan bruge Bouncy Castle er, hvor man er helt nede og laver de forskellige krypterings klasser og nøgle parameter man skal bruge. Ved at gøre det på denne måde, kommer man til at skrive en del mere kode, men man får en meget bedre forståelse af, hvad ens program gør, hvorfor det gør det og hvordan det gør det.

Vores tilgang til implementeringen er blevet ændret undervejs for at kunne få den mest optimale implementering. Vi startede med at benytte Bouncy Castle igennem JCE, men vi løb ind i nogle problemer og måtte derfor genoverveje tilgangen. Vi følte ikke vi havde nok kontrol over forløbet og kunne ikke umiddelbart rette nogle af de fej,l som biblioteket gav os, når det blev benyttet igennem JCE. Vi valgte derfor at gå helt ned og importere de klasser vi skulle bruge og benytte biblioteket på den måde.

Da det ikke var muligt at finde nogle implementationer af de gamle krypterings algoritmer, blev vi nødsaget til at implementere dem selv. Algoritmerne blev implementeret ud fra en beskrivelse fundet i et kompendium fra et andet kursus på DTU [LRK].

5.2 Implementeret kryptografisk funktionalitet

Programmet skal som tidligere nævnt bruges som et værktøj til at få en bedre fornemmelse for, hvordan forskellige algoritme konfigurationer fungerer. Vi har prøvet at udvælge nogle forskellige former for krypteringsformer, som vi fandt, der kunne være interessante at have med. Konsekvensen af vores valg, om at benytte os af kun et enkelt bibliotek til kryptografien, gjorde at vi selvfølgelig var en smule begrænsede af, hvilke implementationer vi havde adgang til. Vi synes dog stadig, at vi har fået et godt udvalg af krypteringsmuligheder med i programmet. Vi har valgt at fokusere på tre overordnede krypterings kategorier. **Symmetrisk** kryptering, **Asymmetrisk** kryptering og **envejs** kryptering. Vi har valgte også implementere 3 kendte gamle klassiske algoritmer, som man også kan prøve.



Figur 3: Oversigt over cipher struktur

5.3 To-vejs kryptering

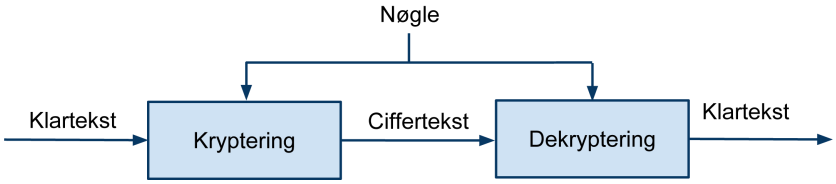
Hoved ideen i symmetrisk og asymmetrisk kryptering er at man har en **'Klartekst' P**, som altså er en tekst, der ikke er modificeret og en **'Ciffertekst' C**, der er den krypterede form af P. Man ønsker så, at C kan udledes ved at benytte sig en krypterings funktion kaldet E kombineret med P således at $\mathbf{C = E(P)}$. Tilsvarende skal der findes en dekrypterings funktion D der kan lave C tilbage til P altså så $\mathbf{P = D(C)}$. Så for at få et fungerende tovejs krypteringssystem skal der gælde at $\mathbf{P = D(E(P))}$. [SIC]

I det beskrevne simplificerede system, kan det godt se ud som at det er dét faktum at algoritmen er hemmelig, der gør at andre ikke kan kryptere og dekryptere en tekst. Det er ikke tilfældet her, og selvom der findes få nøgle løse krypteringsalgoritmer [bla.CEASAR CIPHER], så virker de kun så længe algoritmen er hemmelig. De algoritmer, hvor skaberne ikke har offentlig gjort algoritmen er umiddelbart ikke at betragte som sikre. En tommelfinger

regel siger at en krypteringsalgoritme aldrig må afhænge af hemmeligholdelsen [CRY]. Det betyder at vi er nødt til at have en anden form for sikkerhed, og det er her nøgler kommer ind i billedet.

5.4 Symmetrisk kryptering

Symmetrisk kryptering kaldes også for 'secretkey' kryptering. Det er fordi man i denne form har én nøgle som kan bruges til både kryptering og dekryptering, se fig. 4. Det er derfor vigtigt at holde denne nøgle hemmelig for folk, som ikke skal kunne dekryptere og kryptere beskeder, der er sendt, eller skal sendes med nøglen. Det er heller ikke muligt at være sikker på, hvem der har krypteret og sendt en besked, med denne form for kryptering uden andre hjælpemidler. Det der basalt sker, er at funktionerne D og E er hinandens inverse funktioner, og derfor kan ophæve hinanden med den samme nøgle K. Kryptografisystemet skal derfor kunne tilfredsstille $\mathbf{P = D(K,E(K,P))}$ for at være en gyldig symmetrisk krypterings algoritme.



Figur 4: Symmetrisk: Kryptering og Dekryptering

5.4.1 Klassiske algoritmer

De tre klassiske cifre er blevet valgt, for at give brugeren en forståelse for, hvordan kryptering så ud før de mere avancerede algoritmer blev udviklet. Nogle af deres elementer indgår stadig i dag i de moderne cifre, men som selvstændige krypteringsalgoritmer er de ikke betragtet som sikre.

Shift cipher er en af de enkleste og mest kendte krypterings algoritmer. Algoritmen er en substitutions ciffer. Substitution er en teknik, hvor man udskifter et bogstav med et andet. Altså kunne man for eksempel vælge at i hele teksten bliver A til F. Den teknik er anvendt i denne ciffer og hvert bogstav i klarteksten bliver erstattet med et bogstav nøglens længde længere fremme eller tilbage i alfabetet. Den har i øvrigt rødder tilbage til Julius Caesar's tid hvor den blev kaldt Caesar Cipher. [LRK]

Affine cipher er en mono alfabetisk substitution ciffer, hvor hvert bogstav i alfabetet er knyttet til dens numeriske ækvivalent og derefter krypteret ved hjælp af en simpel matematisk funktion. Og hvert enkelt bogstav krypteres for sig. [LRK]

Vigenere cipher er det man kalder en poly-alfabetisk substitutions ciffer hvilket betyder at man kryptere mere end et bogstav ad gangen ved hjælp af substitution. [LRK]

5.4.2 Blokkrypterings algoritmer

En blok algoritme er en krypterings form hvor man tager et fast defineret udsnit af en tekst og krypterer den. Den mængde data man tager ud afhænger af blokstørrelsen på algoritmen som kan være fast defineret eller variabel inden for et interval. Hvis den sidste blok ikke er komplet bliver man nødt til at benytte sig af et padding skema. Det kommer vi ind på senere.

AES (Advanced Encryption Standard) er en symmetrisk nøgle kryptering standard vedtaget af den amerikanske regering som afløser for DES. AES blev udbudt som en konkurrence hvor flere algoritmer var med, dog var det Rijndael der løb med sejren. Selvom Rijndael tilbyder flere nøglelængder og blok størrelser er dem angivet i tabellen, er de ikke angivet i AES standarden som er udstedt af NIST [NIST-AES] AES cifferen er blevet analyseret indgående og bruges nu verden over og betragtes som sikker.

Understøttede nøglelængder (bit)	Understøttede blokstørrelser (bit)
128, 192 og 256	128

Tabel 1: AES: Nøgle og blokstørrelser

Rijndael er den algoritme som vandt den proces i gang sat af den amerikanske regering for at finde en standard til kryptering. Den benytter sig af substitution, transposition, shift, XOR og addition. Alt efter om der hvilken nøgle der er valgt udfører algoritmen 10,12 eller 14 gentagelses runder.

Understøttede nøglelængder (bit)	Understøttede blokstørrelser (bit)
128, 160, 192, 224 og 256	128, 160, 192, 224 og 256

Tabel 2: Rijndael: Nøgle og blokstørrelser

DES(Data Encryption Standard) blev udviklet af IBM og udvalgt af National Bureau of Standards som en officiel Federal Information Processing Standard (FIPS) for De Forenede Stater i 1976, og som efterfølgende er blevet udbredt til anvendelse I hele verdenen. Den er baseret på en symmetrisk nøgle algoritme, som bruger en 56-bit nøgle. I det bibliotek vi benytter skal man dog give en 64 bit nøgle, som input, men den bruger kun de første 56 bit. Algoritmen er ikke betragtet som sikker i dag grundet nøgle størrelsen er for lille og kan med det rette hardware brydes relativt nemt. [CTAP] Det er grundet designet i algoritmen ikke muligt at udvide den så man kan bruge en større nøgle. Den benytter sig af 16 gentagelses runder.

Understøttede nøglelængder (bit)	Understøttede blokstørrelser (bit)
56	64

Tabel 3: DES: Nøgle og blokstørrelser

TripleDES er et forsøg på at gøre DES algoritmen stærkere. Den fungerer ved at bruge algoritmen 3 gange i træk, hvor den mest gængse er kryptering-dekryptering-kryptering (encryption-decryption-encryption, EDE). Hvor stærk denne form for kryptering er afhænger af nøglen der valgt. Der er normalt 3 muligheder.
1. 3 forskellige nøgler hvilket giver en nøgle på i alt 168 bit, hvor alle 3 nøgler er uafhængige.
2. 2 forskellige nøgler som giver en nøgle på i alt 112 bit. hvor 2 nøgler er uafhængige. K1 og K3 er ens mens K2 er uafhængig af de to andre.
3. 1 nøgle der giver en totalt nøglelængde på 56 bit.

Den tredje mulighed er ikke implementeret i Bouncy Castle og det kan skyldes at når man kun bruger en nøgle svarer det til almindelig DES da, kryptering-dekryptering ophæver hinanden med samme nøgle, men mulighed 1 og 2 er implementeret.

Understøttede nøglelængder (bit)	Understøttede blokstørrelser (bit)
(56), 112, 168	64

Tabel 4: DESede: Nøgle og blokstørrelser

Blowfish er en symmetrisk blok ciffer, designet i 1993 af Bruce Schneier og indgår i en lang række krypterings produkter. Blowfish er en god algoritme at implementere i software, da den er effektiv. Schneier's tanke bag designet af Blowfish var som et alternativ til den aldrende DES og fri for de problemer og begrænsninger, der er forbundet med andre algoritmer. Blowfish algoritmen er offentlig for alle, og kan frit bruges. Den benytter sig af 16 runder.

Understøttede nøglelængder (bit)	Understøttede blokstørrelser (bit)
1-448 bit (Vi arbejder med multiplum af 8)	64

Tabel 5: Blowfish: Nøgle og blokstørrelser

Serpent blev nummer to efter Rijndael, i konkurrencen om at blive Advanced Encryption Standard og en af udviklerne er Lars R. Knudsen fra DTU. Serpent var designet til at give brugerne en høj sikkerhed. Serpent er langsommere, som end eksempelvis Rijndael, da Serpent bruger 32 runder og Rijndael kun bruger 16. Serpent bruger så mange runder for at gøre sikkerheden større [SERPENT]

Understøttede nøglelængder (bit)	Understøttede blokstørrelser (bit)
128, 192 og 256	128

Tabel 6: Serpent: Nøgle og blokstørrelser

Twofish var en af de fem finalister i Advanced Encryption Standard konkurrencen, men blev ikke udvalgt til standardisering. Twofish relateret til den tidligere blok ciffer Blowfish. Twofish er ikke patenteret og er fri at bruge for alle.

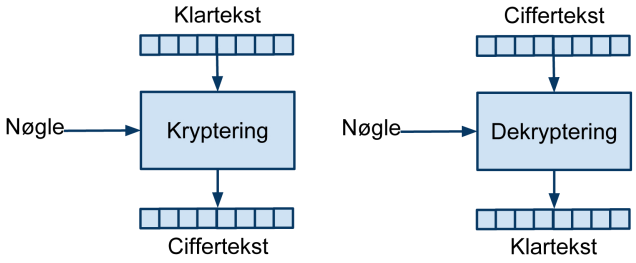
Understøttede nøglelængder (bit)	Understøttede blokstørrelser (bit)
128, 192 og 256	128

Tabel 7: Twofish: Nøgle og blokstørrelser

Rijndael, Serpent og Twofish er taget med fordi de var de tre top placerede i konkurrencen om at blive den nye AES. Blowfish er taget med fordi det er forløberen til Twofish, som iøvrigt også er designet af Bruce Schneier. DES er en ældre og meget anerkendt krypterings algoritme og var den første krypteringsalgoritme til computere. Triple DES er et forsøg på at videreudvikle DES, til at få større nøgler, og derfor er den taget med.

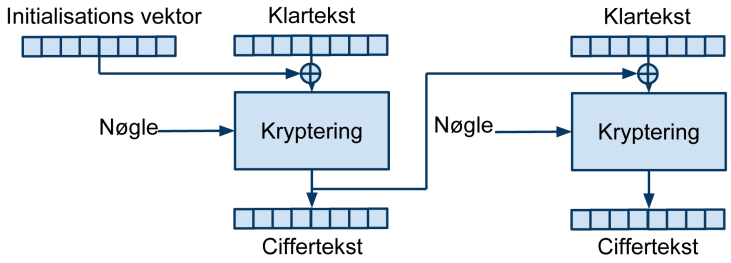
5.4.3 Krypterings indstillinger

Electric codebook (ECB) er den indstilling, som man også kan kalde standard. Når man bruger ECB er de forskellige blokke ikke afhængige af hinanden, og en blok vil altid kryptere til det samme uanset, hvad den foregående blok var [AC]. De betyder også at eventuelle mønstre i en klartekst, også vil træde i cifferteksten. Dette ses tydeligt, hvis man vælger at kryptere et billede, hvor der indgår blokke med samme farve. Ved at se på fig. 5, kan man se, hvordan blokkene ikke er afhængig af hinanden.

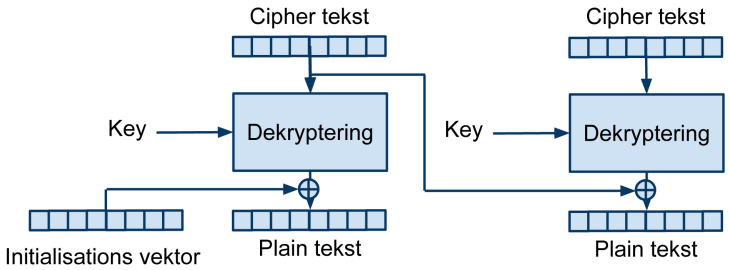


Figur 5: ECB: Kryptering og Dekryptering

Cipher-block chaning (CBC) tager den forgående ciffertekstblok og XOR'er den med den næste klartekst blok, hvor efter blokken bliver krypteret. [AC] Denne indstilling er meget følsom overfor støj, for eksempel, hvis en bit ændre sig, så ændre alle de efterfølgende bit sig, fordi blokkene er afhængige af hinanden, som ses på fig. 6 og 7. Det gør at det også er muligt at benytte CBC til at undersøge integriteten af ens data, ved simpelt at kigge på den sidste blok.

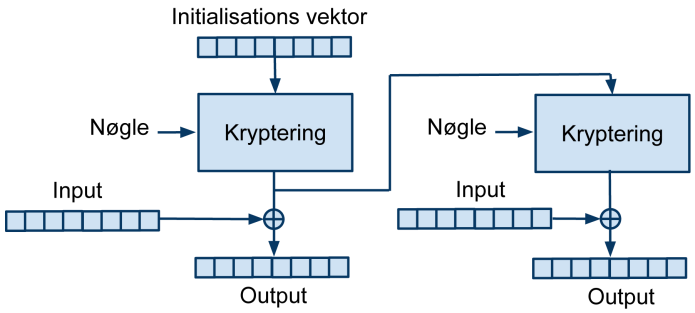


Figur 6: CBC: Kryptering



Figur 7: CBC: Dekryptering

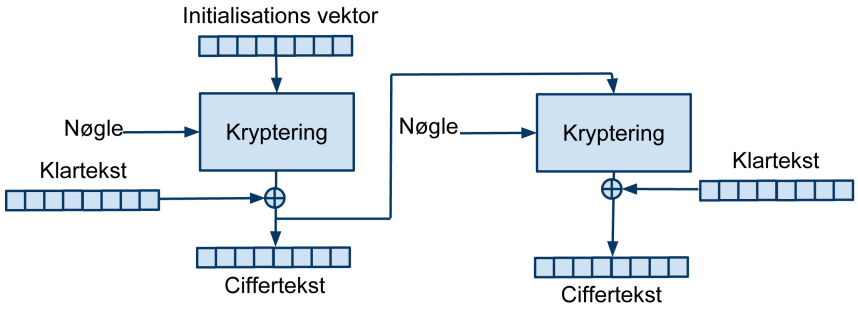
Output feedback (OFB) krypterer en initialisationsvektor med en nøgle, og får en sekvens af bytes med længden n. Det betyder at en blokciffer reelt bliver til en synkron streamciffer. Processen bruger den krypterede sekvens til at gentage processen, som vist på fig. 8, og til at XOR'er den med et inputet af længden n, og på den genererer processen et output. [AC]



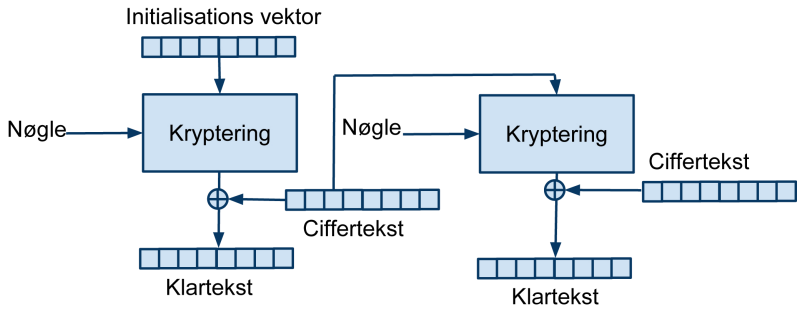
Figur 8: OFB: Kryptering og Dekryptering

Cipher feedback (CFB) er tæt relateret til CBC, den eneste forskel der er på de to indstillinger er, at med CFB kryptere man cifferteksten før man XOR'er den med klarteksten, hvorimod i CBC XOR'er man først [AC]. Dette kan man se ved at sammenligne fig. 9 og 6. Når man dekryptere med indstillingen CFB, udfører man krypterings processen baglængs, som er illustreret på fig. 10.

(OpenPGPCFB)/(PGPCFB) er en variant af CFB. Denne variant skiller sig ud fra CFB, ved at den ikke bruger en initialitionsvektor. Den genererer en tilfældig blok, som bliver krypteret første gang i stedet for initialitionsvektoren. [OPENPGPCFB]

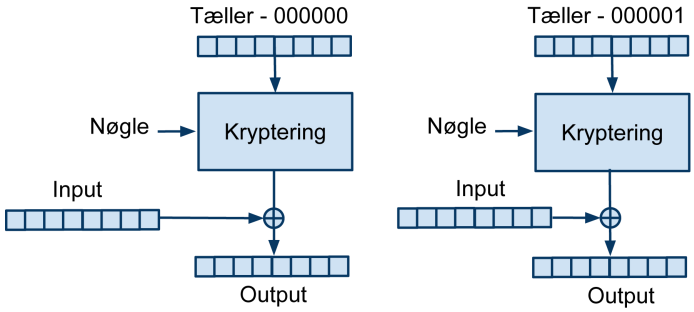


Figur 9: CFB: Kryptering



Figur 10: CFB: Dekryptering

Segmented integer counter (SIC) krypterer en tæller med en nøgle, og får en sekvens af bytes af længden n. Processen XOR'er sekvensen med n input bytes, og på den måde genererer processen et output [AC]. Dette er illustretet på fig. 11, hvor man både kan se, hvordan indstillingen forgår under kryptering og dekryptering.



Figur 11: SIC: Kryptering og Dekryptering

5.4.4 Streamkrypterings algoritmer

En streamkryptering er en anden form for kryptering. Her krypterer man ikke hele blokke men nøjes med enten en bit ad gangen eller nogle gange en byte. [REF merer på]

RC4 er den stream algoritme, som er mest brugt. Den bliver brugt til for eksempel SSL (Secure Sockets Layer) og WEP. RC4 har en nøgle størrelse mellem 40–2048 bits.

RC4 er blevet meget udbredt på grund af dens hastighed og algoritmens enkelthed. Den har dog sine svagheder i at, hvis dens IV ikke er helt tilfældig eller er relaterede, så er den "nem" at bryde. Dette gør sig gældende for WEP's implementation af RC4 som ikke længere er betragtet som sikker.

Grain v1 var den første version af Grain familien. Denne algoritme tager imod en nøgle med en størrelse 80 bits. **Grain-128** er en videre udvikling af Grain

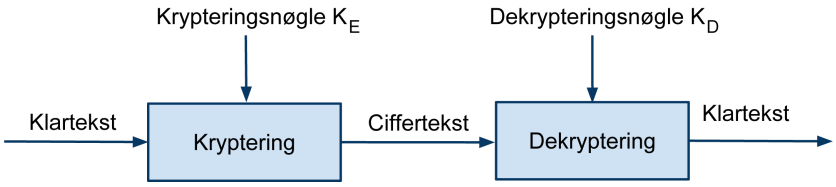
v1, Grain-128 kan tage imod en 128 – bit nøgle og en 96-bit IV. Grain-128 var med i projektet eStream, som fandt sted i årene 2004 til 2008. [GRAIN]

Salsa20 er blevet lavet på en funktion, som er pseudorandom. Salsa20 kan tage imod en nøgle på henholdsvis 128-bit og 256-bit. Salsa20 var med i projektet eStream, som fandt sted i årene 2004 til 2008.

HC256 bruger en 256 bit nøgle og samme størrelse initialitionsvektor. Den er ikke patenteret og det er ikke fundet nogen sikkerhedsfejl i algoritmen endnu. Der findes også en mindre tung variant som også bruger mindre tid. Den kaldes for **HC128** og har tilsvarende en nøgle og en initialitionsvektor på 128 bit. Den var iøvrigt udvalgt til eStream konkurrencen også. [HC128]

5.5 Asymmetrisk kryptering

Asymmetrisk kryptering er også kendt som 'Public Key' kryptering. Det smarte ved asymmetrisk kryptering er at man har 2 nøgler, en public key og en private key. De to nøgler hænger sammen som et par. Hvis man kun har den ene nøgle er det ikke muligt at udlede den anden. Det har den fordel at man kan offentlig gøre sin public key også kan folk kryptere ting med den som kun den der her den tilhørende private key kan dekryptere, se fig. 12. Med dette system er det altså muligt at oprette en hemmelig kontakt uden at skulle dele nøgler først som en potentiel angriber kan få fat i under udvekslingen. Det har dog den ulempe at det er markant langsommere end symmetrisk kryptering, og man bruger det derfor tit til at udveksle en fælles krypteret nøgle til en symmetrisk algoritme. Det skal nævnes, at man i public key systemet stadig har problemet med, hvem der ejer den public key, man har fået. Det er delvist løst ved hjælp af Public Key Infrastructure (PKI) der er et system til at holde styr på nøglerne, men det kræver stadig, at man kan stole på dem, der har udstedt certifikatet [CTAP]. En asymmetrisk kryptering vil skulle opfylde at P = D(Kpriv,E(Kpub,P)), hvor D er dekrypterings mekanismen, E er krypteringsmekanismen og Kpriv og Kpub er henholdsvis private- og public key. [RSA]



Figur 12: Asymmetrisk: Kryptering og Dekryptering

RSA var en af de første avancerede offentlig nøgle algoritme, som kan bruges til både at signere og kryptere med. I teorien har RSA ikke nogen nøgle begrænsning, men når størrelsen af nøgle bliver større, tager det både længere tid at genere en nøgle, og udføre krypterings og dekrypteringsprocessen [SIC]. RSAs blok størrelse afhænger af, hvor stor nøglen er, da den maksimalt kan

kryptere en blok på nøglens størrelse af gangen, hvis input blokken er større end nøglen, kan man starte forfra på nøglen, og kryptere input blokken videre.

ElGamal er baseret på Diffie-Hellman nøgle udveksling, som er en protokol, der kan hjælpe med at skabe en fælles nøgle ved at bruge potensregne reglerne. Ligesom RSA har ElGamal ikke nogen teoretisk maksimal nøgle størrelse.

5.6 En-vejs kryptering

Når man kigger på envejs krypteringer er målsætningen en helt anden. Her er ønsket at man nemt kan gå den ene vej, men ikke tilbage. Det kan blandt andet være meget nyttigt i forbindelse med at sikre integritet af dataene. Dette gøres ved at en arbitrær mængde data kan udtrykkes ved hjælp af et antal bits, hvis længde er afhængig af den valgte algoritme. Dette output kaldes for en hash. En af de vigtigste egenskaber ved en hashfunktion, er at den er kollisionsfri, hvilket reelt betyder at det er stort set umuligt at beregne sig frem til to forskellige klartekster, der giver den samme funktionsværdi, altså hash.

GOST3411 er en hashfunktion, der er baseret på GOST blokcifferen. Den er første gang blevet defineret i den Russiske National Standard. Hvis inputtet ikke er stort nok udfylder algoritmen ud med nuller. Outputtet, som GOST3411 giver, har en størrelse på 256 bit.

MD2/4/5 er algoritmer, som tilhører familien MD. MD betyder MessageDigest og de er alle sammen udviklet af Ron Rivest, som også var med til at udvikle RSA. Der findes nogle forskellige varianter og vi starter med at kigge på MD2. MD2 er fra 1989 og har et output på 128 bit. Den er optimeret til 8 bit systemer, men betragtes ikke længere som sikker i dag, da man har fundet fejl i dens komprimeringsalgoritme, der gør det muligt at finde kollisioner [MD2COL]. MD4 er udviklet tre år senere og har også et output på 128 bit. Algoritmen er ikke betragtet som sikker eftersom et angreb på algoritmen i 1995 viste alvorlige fejl i den der gjorde at det var muligt at finde kollisioner. MD5 er også udviklet i 1991 med et output på 128 bit. Den er ikke længere betragtet som sikker, men har holdt længere end de tidligere versioner. Den er stadig brugt til mange ting i dag, blandt andet til at hashe passwords i mange databaser, men også til at sørge for dataintegritet i diverse programmer, hvor man bruger den til en fil checksum.

RipeMD128/160/256/320 er en familie af hash-funktioner. Original udgaven er på 128 bit men en forbedret udgave på 160 bit findes. Der er fundet en kollision i 128-bit udgaven, som dog ikke er at finde i 160-bit udgaven. Begge disse er forsøgt forbedret med henholdsvis en udgave på 256-bit og 320-bit. Det er dog udelukkende for at formindske risikoen for kollisioner.

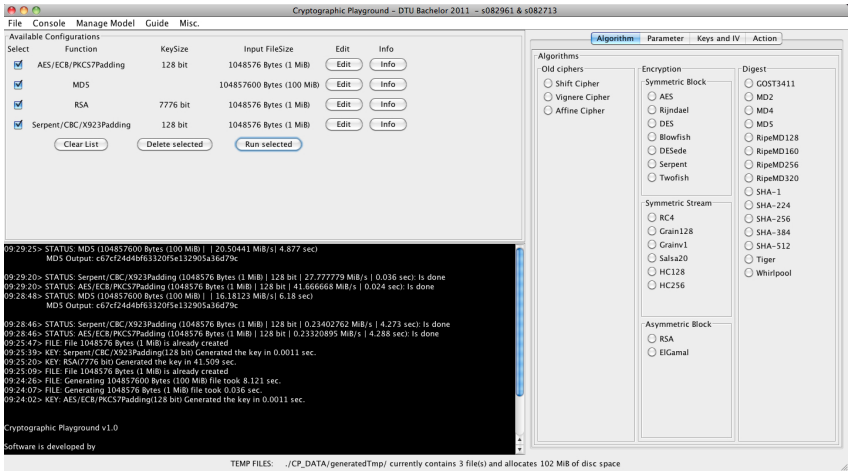
SHA-1/224/256/384/512 er ligesom RIPE en familie af hash-funktioner. SHA står for Secure Hashing Algoritm. SHA-0 blev udviklet i 1993, og havde en nøgle størrelse på 160-bits. Den blev meget hurtig taget af markedet igen,

da man fandt nogle væsentlige fejl i algoritmen. SHA-1 minder meget om SHA-0, dog er SHA-0s fejl rettet. Begge algoritmer er blevet udviklet af NSA. SHA-1 blev udviklet for at være en del af Digital Signature Algorithm. SHA familien blev senere forbedret og fik navnet SHA-2. Denne algoritme har to blok størrelser på henholdsvis 512 bits, som har output størrelse på 224/256 bits og 1024 bits, som har en output størrelse på 384/512 bits. I øjeblikket er NIST i gang med at holde en konkurrence, hvor de skal finde den nye SHA-3. [NIST-SHA3]

Tiger er en algoritme udviklet i 1996 af de to folk, som sammen med Lars Knudsen udviklede den tidligere nævnte Serpent algoritme. Tiger er optimeret til at køre på 64-bit arkitektur, og har et output på 192 bit. Der findes ikke gennemførte angreb på algoritmen med undtagelse af et mindre angreb. Den betragtes stadig som sikker.

Whirlpool er udviklet i år 2000 og bygger på en modificeret udgave af AES. Den har et output på 512 bit, og er ikke patenteret, og man kan derfor frit benytte algoritmen.

6 Programstruktur



Figur 13: Hele programmet som det ser ud på en MacBook.

Programmet er bygget op omkring et data objekt, som indeholder alt data, der skal bruges til at udføre en krypteringsproces. Dette data objekt er ikke et objekt, som er blevet gemt i modellen, men derimod et objekt, der bliver sendt med, hver gang der bliver oprettet et nyt element (for eksempel panel, nøglegenerering m.m), eller der skal laves noget om i data objektet.
Da det var et ønskeligt krav, at kunne sammenligne data objekter, er det derfor muligt at gemme data objekterne i en liste, hvor man også kan tage et data objekt ud af listen og arbejde med det.

Et af de andre ønskelige krav var, at man skulle kunne gemme og indlæse nøgler og programmets tilstand. Dette burde ikke være et problem, da man i Java har muligheden for at skrive objekter ud direkte. Denne metode kan dog kun skrive objekter til en fil, hvis de har implementeret Javas interface Serializable. Det gav os nogle problemer i forhold til vores bibliotek da forskellige dele af klasserne ikke er serializable. Det betyder at man ikke kan gemme Bouncy castle's objekter, og i vores tilfælde kunne vi ikke skrive parametre og engines til en fil. Det gjorde at vi måtte lave vores datastruktur lidt om fordi, at i vores tidlige design var disse nemlig en del af vores data objekter, som bliver gemt når modellen skal gemmes til en fil. Begge problemer bliver løst ved at data objektet indeholder byggestenene til både parametre og engine, også når en krypteringsproces skal udføres, så de bliver genereret, så vi ikke længere gemmer objekter der indeholder klasser fra Bouncy Castle.

I programmets implementering er der blevet udviklet en nøgle og initialitionsvektor generator, som genererer alle byggestene til de parametre, som skal bruges til en krypteringsproces. Der er som udgangspunkt brugt Bouncy Castle's implementering og men vi har også været nødsaget til selv at implementere nogle algoritmer nøgle algoritmer selv. På stort set alle ciffers var det muligt at hive nøgle bytes ud af nøgleobjektet og derfor kunne vi gemme

dem. Det var ikke gældende for ElGamal og RSA. Dem blev vi nødt til selv at implementere, for at have helt kontrol over dem. Algoritmerne blev fundet i tidligere nævnte kompendium [LRK]

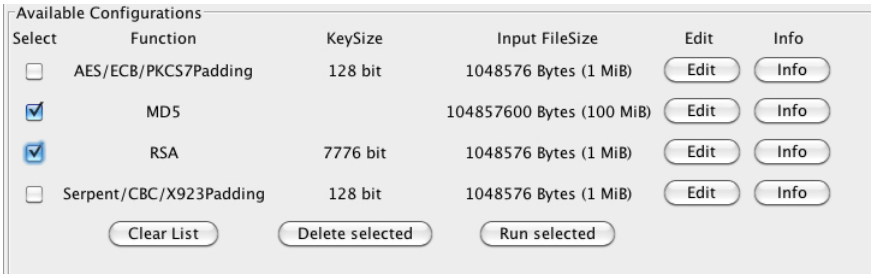
En af vores intentioner med programmet er, at man nemt skal kunne videreudvikle det, ved for eksempel at lægge flere algoritmer ind i programmet eller ændre de eksisterende. Derfor bliver alle algoritmerne specificeret i en initialiserings fil, som programmet indlæser, hver gang det starter. På den måde kan man let ændre, tilføje eller fjerne algoritmer. Disse algoritmer bliver lagt i en liste i programmet og vist i programmets grafiske del. Hvis man tilføjer en algoritme, som ikke er implementeret i programmet, vil algoritmen ikke kunne eksekveres, men programmet vil ikke gå ned. Det er dog lidt mere arbejde en bare at ændre i initialitions filen, men alt det grafiske bliver generet ud fra den, og det er simpelt at tilføje algoritmen i programmet også.

Figur 14: Algoritme Panelet

Ligesom programmet henter information omkring algoritmeparametre fra en fil, henter programmet også al information omkring algoritmerne fra tekst filer, og lægger dem ind i guiden. Herved kan brugerne læse omkring de forskellige algoritmer og danne sig et billede af, hvordan algoritmerne hænger sammen.

Krypteringsprocessen er det bærende element i programmet. Den tager hånd om oprettelse af krypterings-engine og -parametre, for derpå at kryptere den givne fil eller tekststreng.

Processen er blevet lavet på en sådan måde, at det er ligegyldigt hvor stor filen er. Programmet henter en mængde data ind fra en fil svarende til den bufferstørrelse, som er defineret i programmet. Til symmetrisk og digest krypteringsalgoritmer har vi defineret en værdi på 2000 bytes. Dette tal er valgt ud fra testning. Hvis værdien forhøjes sker der ikke nogen markant ændring i hastigheden, og hvis værdien mindskes falder hastigheden væsentligt. RSA og ElGamal har ikke nogen fast defineret bufferstørrelse, da bufferens størrelse skal være den maximale størrelse som krypterings-engine kan håndtere. Ellers giver Bouncy Castle biblioteket en fejl. Denne værdi bliver fastsat automatisk af, efter hvor stor nøglen er. Det skal dog også lige nævnes at med vores implementering med buffer er det muligt at kryptere større filer en nøglen. Det fungere i praksis ved at man kryptere blokke og lægger dem ved siden af hinanden i filen.



Figur 15: Liste over objekter

I implementeringen har vi lagt stor vægt på at bruge super klasser. På den måde behøver man ikke, at lave individuelle metoder til alle funktioner i programmet, da man kan sammentrække mange af metoderne. Eksempelvist når der skal laves en krypterings-engine, hvor en speciel metode returnerer krypterings-engines superklasse. Ved at gøre dette, ender man ud med at have fire metoder i stedet for 33. Én for hver algoritme. Superklasserne hjælper med til at holde en modulbaseret skruktur, ved at opdele algoritmerne i kategorier såsom Symmetriske, Asymmetrisk, Blok, Stream og digest.
Selve programmet er delt op i tre dele; model, view og kontrol. Modellen er delt yderligere op så vi adskiller nøgle, initialiseringsvektor, diverse krypteringsengine, parametre, filhåndtering, selve det data som bliver gemt og de metoder som styrer hele programmet. Ved at opdele modellen på denne måde, gør man det nemt og overskueligt at fjerne og tilføje moduler til modellen.

View delen er også lavet modulbaseret. Ved at lave programmet således kan man genbruge diverse paneler, så man ikke behøver at have tusinde linjer af den samme kode.

Designet er blevet udviklet med henblik på at gøre programmet brugervenligt for at brugeren kan forstå, hvad meningen med programmet er. Programmets grafiske del er opdelt i fire sektioner;
- En liste over hvilke dataobjekter brugeren har oprettet
- En konsol hvor brugeren kan læse en log, som forklarer hvad der sker i programmet

- En sektion hvor brugeren kan oprette data objekter
- En sektion, hvor brugeren kan ændre i data objekterne. Se fig13 for hele programmet.

Når brugeren vælger at ændre indstillinger i programmet skal den grafiske del opdateres. For at hele programmet ikke skal opdateres, hver gang der sker en ændring, er der inkorporeret en metode, som kun opdaterer de relevante panels, som har en relation til det panel, som brugeren har ændre indstillinger i.

Ligesom både modellen og view er modulbaseret er kontrolleren også modulbaseret. Kontrolleren lægger sig meget tæt op af den måde view er opdelt. Stort set hvert view-modul har en kontroller tilknyttet, som varetager de events brugeren opretter i viewet, og som videregiver ordre til viewet og modellen.

7 Grafisk Brugergrænseflade

7.1 Java Swing

Når man skal lave en brugergrænseflade i Java, har man tre muligheder. Swing er Java's primære Graphical user interface(GUI) værktøj, som bliver brugt til at lave den grafiske brugerflade til Java programmer. Swing er en form for overbygning til AWT (abstract windows toolkit), som var det første grafiske modelleringsværktøj til Java. AWT er et GUI bibliotek til programmer og applets, og er en grafisk måde at fortolke den underliggende kode. Swing har implementeret alle AWT's komponenter men bruger til forskel fra AWT ikke peers. Eksempelvis skal man have installeret Java's engine ved brug af Swing, for at vise dennes komponenter, hvorimod AWT ikke altid havde brug for Java's virtuelle maskine, da de grafiske elementer var indbygget i webbrowseren.

Swing har et større spektrum af komponenter og er langt mere avanceret end AWT, i form af bl.a. fanebladspaneler, scroll paneler og træer. AWT er også kendt for at være notorisk svært at programmere i. Derudover tilbyder Swing en styresystemsbaseret [GUI] program tema, som betyder at udseendet og følelsen ved at bruge programmet sammenspiller med det benyttede styresystem. På den måde kan man på en Mac computer få samme udseende som det skin man bruger, hvor man med samme program på en Windows PC vil have et Windows tema i stedet.

Fordelen ved swing ud over ovennævnte features er, at der er mulighed for at bruge meddelelses bokse og ikoner. Derudover er swing meget robust da den løbende opdateres og vedligeholdes af Sun. [AWT/SWING] En mindre ulempe er at komponenterne generelt er langsommere end AWT da Java gentegner dem med Java's egen grafiske API i stedet for computerens, som for eksempel directX på en Windows computer. Af alternativer til swing kan nævnes AWT og SWT. AWT er beskrevet ovenfor, mens SWT (standard widget toolkit) er stort set det samme som swing. Dog minder implementeringen i højere grad om AWT. SWT er lavet af IBM og konkurrerer dermed med Sun. Swing fokuserer dog mere på følelsen og styresystemets udseende.

Grunden til, at vi har valgt swing er, at den i forhold til AWT har flere komponenter og da vores underliggende kode er skrevet i Java, skal der alligevel installeres en virtuel maskine (JVM) på computeren for at køre programmet. Swing er blevet valgt frem for SWT da vi har arbejdet med swing før. Der er ellers ikke de store forskelle på SWT og Swing.

7.2 Komponenter

Nedenfor har vi taget valgt et mindre udsnit af de komponenter vi har brugt i programmet, og de vil blive beskrevet og forklaret. Vi har valgt ikke at fokusere på de gængse swing komponenter såsom JLabel, JButton m.m. I

stedet har vi lagt større vægt på de mere avancerede komponenter og dem man ikke bruger så tit.

JTabbedPane er et af de vigtigste container blev der brugt i programmet. Et JtabbedPane er en komponent som gør det muligt at skifte mellem en gruppe komponenter ved at klikke på et tab med et ikon eller tekst på. Denne komponent gør det mere overskueligt at navigere rundt mellem de relevante parametre, som man skal give programmet før det kan kryptere eller dekryptere data.

JSlider bruger vi til at definere diverse parameter i programmet. En JSlider er en komponent, hvor man kan vælge en værdi grafisk, ved at flytte rundt på slideren. JSlideren har sine fordele ved at man kan give den en parameter, som fortæller den, hvilke værdier den skal vise, og hvilke spring den skal tage. Eksempelvis har AES tre forskellige nøglestørrelser, hvor man med en JSlider kan sætte intervallerne, så man ikke ender med en fejl i nøglestørrelsen.

JProgressBar bliver brugt til at kunne sammenligne de forskellige algoritmer som programmet kan køre. En JProgressBar er en komponent, som grafisk viser, hvor langt en underliggende proces er kommet. Denne underliggende proces er en Swing worker. Kort fortalt er en swing worker en trådbaseret klasse, som laver en opgave og fortæller JProgressBar hvor langt den er nået. Swing worker bliver forklaret bedre i næste afsnit.

JScrollPane er tilføjet til diverse paneler for at undgå problemer med ikke at kunne vise hele panelerne fuldt ud, hvis skærmen eller vinduet er for småt. Denne komponent giver mulighed for at scrolle i et panel, hvis panelet er for stort til viewet.

JTree bliver brugt i Guide sektionen, så det er nem at navigere rundt, og til at give en forståelse om, hvordan de forskellige algoritmer hører sammen. Et JTree er en måde at kunne vise, hvordan hierarkiet er fordelt. I dette tilfælde er et JTree rigtig brugbart, da det ikke kun viser hierarkiet, men også bruges til at videregive informationen som skal vises i guiden.

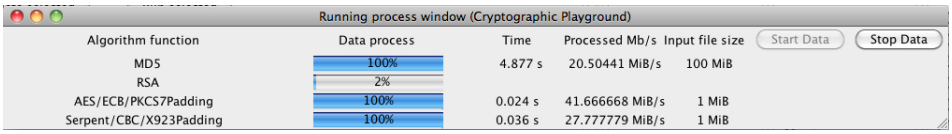
7.3 Tråde og Swing-worker

Når man bruger en computer, tager man det ofte for givet, at den ikke fryser, og man kan bruge flere programmer på én gang, eller at et program kan lave flere ting af gangen. For at kunne opfylde disse krav bliver programmer udviklet til at køre i tråde.

Tråde er en måde at benytte computerens ressourcer på, ved at uddelegere trådene på flere kerner, hvis processoren indeholder flere kerner. Har man kun en enkelkerne computer, kan man stadig godt bruge tråde, da operativsystemet skifter mellem de aktive tråde på kernen. I programmet har de regnekrævende klasser nedarvet javas Thread klasse, så de kan køre parallelt med programmets grafiske del. På den måde kan programmet stadig bruges, når der for eksempel bliver genereret nøgler eller der bliver kørt krypteringsprocesser. Krypteringsprocesser bliver eksekveret i en swing worker, som er

en underklasse af javas Thread klasse.

Swing worker er en tråd, som har nogle få ekstra metoder, som er designet til at kunne arbejde sammen med de forskellige swing komponenter. Swing worker klassen er blevet implementeret af Sun og tilføjet til Swing biblioteket. Swing workeren bliver brugt i stedet for en tråd, da den kan videregive information til JProgressBar, så brugeren kan se hvor langt processen er kommet. se fig. 16



Figur 16: Kørende processer.

7.4 Fejlhåndtering

I implementeringen har vi lagt stor fokus på fejlhåndtering, da meningen med dette program er at stå med et færdigt produkt, som vores vejleder kan bruge til sin undervisning. For at kunne lokalisere og undgå "alle" tænkelige fejl, er hver metode blev analyseret, for hvilke fejl der kunne opstå, og derefter taget hånd om problemerne. Der er blevet brugt to metoder for at undgå at programmet terminerer før tid. Den ene metode er en try catch, som indkapsler en algoritme. Hvis den giver en fejl, bliver catch'en eksekveret, hvorpå der bliver vist en fejlmeddelelse eller taget hånd om det på anden vis. Den anden metode er et sanitets filter, som kontrollerer om alle parametrene stemmer overens med hinanden, som for eksempel at nøglen er ligeså lang som brugeren har bedt om, før parametrene skal bruges til krypteringsprocessen. Alt input bliver kørt igennem dette filter før modellen får lov til at arbejde på data objektet. Som beskrevet før har vi lagt mange timer i fejlhåndtering og gennemtænkning af, hvordan diverse fejl kunne undgås og håndteres på den smarteste måde uden at være til gene for brugeren.

8 Test

8.1 Funktionel test

Da vores implementation skal benyttes af andre studerende er det vigtigt at programmet opfører sig hensigtsmæssigt og ikke går ned. Det er derfor vigtigt at de forskellige dele af programmet er testet for mulige fejl. Der er ligeledes taget hånd om tilfældet hvor brugeren kommer med input som programmet ikke kan tolke. Vi har ikke lavet JUnit testning men har valgt at fokusere på at teste det manuelt, altså en form af funktionel testning. De forskellige moduler er blevet testet på mange forskellige måder og med meget forskelligt input for at se om programmet reagerer som vi ønsker det skal. På denne måde kan man teste, om man har implementeret sine teoretiske ideer rigtigt. Igennem hele implementeringen er der blevet lavet funktionelle test på hver enkel metode, og rettet de fejl der måtte opstå. Ved at lave test løbende, gør man det nemmere for sig selv. For hvis man finder en fejl når man tester, så ved man at fejlen ligger i den metode man lige har implementeret, og så behøver man ikke at lede hele programmet igennem for en lille fejl. Vi har også måtte teste vores krypteringsdel grundigt for at finde de fejl der har kunnet opstå. Ved at kryptere en fil og dekryptere filen igen, har vi kunne undersøge om programmet gjorde det rigtigt. Vi testede derfor output-filerne under udviklingen og til igen til sidst, for at se om de efter en tur igennem programmet var 100 procent identiske med da vi startede. Det gjorde vi ved at køre dem igennem en hash funktion og derefter sammenligne værdierne. På den måde har vores eget program har været med til at teste sig selv. Det skal dog nævnes at vores hash-funktioner er blevet testet ved at sammenligne outputtet fra andre kommercielle checksums programmer. Hvis et krypteringsoutput ikke var som forventet, har vi især benyttet os flittigt af HEX editorer. Hex editorer undersøger det output, som programmet giver, for at finde ud af hvor i filen, og hvilke bytes der eventuelt er malplaceret eller forkert udregnet.

8.2 Performance test

Performance er vigtig i et program. Både i forhold til de ting programmet beregner, men også i høj grad det grafiske. Brugeren vil blive irriteret hvis programmet hakker og virker en smule tungt. Da vores program ikke skal genereres og gentegnes mange gange i sekundet er det begrænset hvor mange grafiske ressourcer brugergrænsefladen benytter. Dog har vi alligevel valgt at optimere så vidt muligt på, hvordan og hvornår de enkelte paneler bliver gentegnet, da der alligevel kan være en del performance at hente her. Det har betydet at vi har fået en betydeligt bedre performance i forhold til starten, hvor vi bare gentegnede hele programmet, inklusiv elementer som ikke var aktive på skærmen, ved at optimere på disse parametre.

Det er kun modellen, som skal bruge mange ressourcer til diverse opgaver. For at få den optimale performance, er alle de regnetunge algoritmer i modellen blevet uddelegeret til deres egne tråde. For eksempel når programmet skal

lave en fil på en GiB data eller kryptere en stor fil, kan brugeren stadig bruge programmet, og derved spare tid i sidste ende.

9 Konklusion

I vores projekt har det vist sig at tidsplanen ikke kunne overholdes. Dette skyldes blandt andet at nogle ting har taget længere tid end, hvad vi havde vurderet fra start af. Vi stødte også ind i nogle problemer med små fejl i det bibliotek, som vi benyttede os af, som gjorde at vi måtte finde nye løsninger til nogle af vores problemer. Selvom det har været en stor hjælp, at have et bibliotek med krypterings algoritmerne i, har det også betydet at vi har brugt væsentlig mere tid end vi havde regnet med på at forstå, hvordan biblioteket fungerede. Vi startede med bare at benytte Bouncy Castle, som provider igennem et JCE wrapper interface. Det er smart, fordi det er meget simpelt at bruge det på denne måde, men man mister også en del føling og kontrol med, hvad der sker. Og da vi gentagne gange sad med fejl, der var meget svære at løse, valgte vi at omskrive koden til at benytte biblioteket direkte, og selv hive de klasser ud vi skulle bruge. Det gør at vi har meget mere kontrol over input til klasserne, men har også den konsekvens at man skal sætte sig meget mere ind i, hvordan biblioteket virker. Dette har taget en del ekstra tid. Et andet problem vi havde med biblioteket var at mange af klasserne ikke var gjort serializeable, som blev til et problem, når vi skulle gemme vores model, da den indeholdte objekter, som har Bouncy Castle objekter i sig. Hverken engines eller nøgler har det været muligt at skrive ud som en stream. Vi har derfor måttet hive de parametre ud som skal bruges for at generere de forskellige objekter fra Bouncy Castle. Det lykkedes fint for os med stortset alle algoritmer undtagen de asymmetriske. Det var ikke muligt at få noget ud, som vi kunne brug,e hvis vi skulle genskabe nøglerne i programmet. Vi blev derfor nødt til at implementere nøgle genereringen selv for ElGamal og RSA. Dette var også medvirkende til at tidsplanen blev forskudt. Så desværre nåede vi ikke alle vores ønskelige krav.

Vi fik lavet vores view til vores guide samt indlæsnings funktion til at tage imod informationen der skulle vises, men grundet tidspres fik vi ikke tastet informationen ind. Derfor er denne menu deaktiveret. Vi blev også nødt til at droppe at implementere flere krypteringsformer. Vi ville gerne have haft signering med samt MAC's. Elliptic curve cryptography (ECC) havde også været spændende at få med, men desværre kunne det ikke lade sig gøre grundet tidsmangel. Et sidste ønske var at få implementeret en form for quiz/spil med relevante spørgsmål omkring kryptografien, dette blev dog lagt på hylden, da vi så at tiden var knap.

Noget som også fyldte en del i projektet ud over de allerede nævnte ting, var fejlhåndteringen. Når programmet skal bruges i praksis vidste vi, hvor vigtigt det var at programmet var smart nok til at håndtere de input, som en bruger kan give. En bruger kan godt komme til at benytte programmet forkert, og derfor skal vi kunne tage hånd om det. Vi blev selv tilfredse med resultatet,

og syntes vi har testet programmet grundigt. Både for "bruger"fejl, men også for funktionalitets fejl.

Det har været et spændende projekt at arbejde med, og vi føler at vi har fået en rigtig god fornemmelse for, hvor omfattende det kan være at skulle udvikle så stort et projekt.

10 Litteratur

[GOF] **Book:** Design Patterns: Elements of Reusable Object-Oriented Software **Author(s):** Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides

[AC] **Book:** Applied Cryptography (2nd edition) **Author(s):** Bruce Schneier

[SIC] **Book:** Security in Computing (4th edition) **Author(s):** Charles P. Pfleeger, Shari Lawrence Pfleeger

[CTAP] **Book:** Cryptography - Theory and practice (3rd edition) **Author(s):** Douglas R. Stinson

[UML] **Book:** UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition) **Author(s):** Martin Fowler

[GUI] **Book:** Introduction to GUI with Java Swing **Author(s):** Paul Fischer

[BJP] **Book:** Building Java Programs **Author(s):** Stuart Reges, Marty Stepp

[LRK] **Pdf:** Cryptography 1 **Author(s):** Lars R. Knudsen (DTU)

[NIST-AES] **Pdf:** ADVANCED ENCRYPTION STANDARD, `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf` **Author(s):** NIST

[BC] **Website** `http://www.bouncycastle.org/specifications.html`

[BCDOC] **Website** `http://www.bouncycastle.org/docs/docs1.6/index.html`

[RES] **Website** `http://download.java.net/jdk7/docs/technotes/guides/security/SunProviders.html#importlimits`

[LIB] **Website** `http://www.homeport.org/~adam/crypto/`

[CRY] **Website** `http://www.infosyssec.com/infosyssec/cryptintro.htm`

[SERPENT] **Website** `http://en.kryptotel.net/serpent.html`

[GRAIN] **Pdf:** Grain - A Stream Cipher for Constrained Environments, `http://www.ecrypt.eu.org/stream/ciphers/grain/grain.pdf` **Author(s):** Martin Hell, Thomas Johansson and Willi Meier

[HC128] **Pdf:** The Stream Cipher HC-128, `http://www.ecrypt.eu.org/stream/p3ciphers/hc/hc128_p3.pdf` **Author(s):** Hongjun Wu

[RSA] **Pdf:** A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, `http://people.csail.mit.edu/rivest/Rsapaper.pdf` **Author(s):** R.L. Rivest, A. Shamir, and L. Adleman

[MD2COL] **Pdf:** An improved preimage attack on MD2, `http://www1.spms.ntu.edu.sg/~guojian/SHA/089-Preimage-MD2.pdf` **Author(s):** Søren S. Thomsen, DTU.

[NIST-SHA3] **Pdf:** Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA–3) Family, `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf` **Author(s):** NIST

[OPENPGPCFB] **Pdf:** An Attack on CFB Mode Encryption As Used By OpenPGP, `http://eprint.iacr.org/2005/033.pdf` **Author(s):** Serge Mister and Robert Zuccherato

[AWT/SWING] **Website** `http://edn.embarcadero.com/article/26970`

A Tidsplan

A.1 1. Februar til 1. Marts

Udvikling af kravspecifikation og indsamling af baggrundsmateriale.

A.2 1. Marts til 15. Marts

Undersøgelse og valg af kryptografi-biblioteker

A.3 15. Marts til 15. April

Implementation af model til programmet

A.4 15. April til 19. April

Testning af model

A.5 19. April til 15. Maj

Udvikling af den grafiske brugergrænseflade

A.6 15. Maj til 19. Maj

Testning af den grafiske brugergrænseflade

A.7 19. Maj til 12. Juni

Færdig med første udkast til rapporten

A.8 25. Juni

Færdig med endelig version af rapport.

A.9 25. Juni til 27. Juni

Printning og klargøring til aflevering.

B Bruger guide

B.1 For at kryptere data

For at starte en krypterings-proces, skal man først vælge, hvilke algoritme man vil kryptere med. Dette gøre i det første faneblad i højre side af programmet.
Hvis man vælger en algoritme, som har brug for diverse parametre, vælger man parametrene i andet faneblad.
Nøgle generering eller indlæsning af nøgler kan foretages i tredje faneblad, hvis det er nødvendigt.
Valg af hvilken fil der skal indgå i krypteringsprocessen vælges i fjerde faneblad.
Når alle de ovenstående punkter er udfyldt korrekt, kan data blive krypteret ved at trykke på RUN knappen.

B.2 For at gemme krypterings-proces indstillinger

For at starte gemme krypterings-proces indstillinger, skal man først vælge, hvilke algoritme man vil kryptere med. Dette gøre i det første faneblad i højre side af programmet.
Hvis man vælger en algoritme, som har brug for diverse parametre, vælger man parametrene i andet faneblad.
Nøgle generering eller indlæsning af nøgler kan foretages i tredje faneblad, hvis det er nødvendigt.
Valg af hvilken fil der skal indgå i krypteringsprocessen vælges i fjerde faneblad.
Når alle de ovenstående punkter er udfyldt korrekt, kan data blive krypteret ved at trykke på Tilføj til list knappen.

B.3 For at ændre en eksisterende krypterings-proces

Find den proces du vil ændre i listen, og tryk på knappen "edit".
Et vindue kommer frem, her kan du ændre indstillingerne for den valgte krypterings-proces, ligesom da du oprettede elementet.
Luk vinduet ned og ændringerne er gemt.

B.4 Kører krypterings-processer fra listen

Vælg de krypterings-processer, der skal køres, ved at sætte et flueben ved dem.
Tryk på "Run Selected".
Der kommer et nyt vindue med processerne, tryk på start.

B.5 Slet krypterings-processer fra listen

Vælg de krypterings-processer, der skal fjernes fra listen, ved at sætte et flueben ved dem.
Tryk på "Delete Selected".
Der kommer et nyt vindue med processerne, tryk på start.

B.6 Find information om algoritmer

Vælg guide i menu baren, og tryk på "start tutorial".
Der kommer et vindue op, hvor man kan finde information omkring algoritmerne.

C Source code

C.1 Model Source Code

Sourcecode not included in printed report. Its on the attached CDROM.
It is also included in the electronic version of the report.

C.2 View Source Code

Sourcecode not included in printed report. Its on the attached CDROM.
It is also included in the electronic version of the report.

C.3 Controller Source Code

Sourcecode not included in printed report. Its on the attached CDROM.
It is also included in the electronic version of the report.

C.4 Misc Source Code

Sourcecode not included in printed report. Its on the attached CDROM.
It is also included in the electronic version of the report.