Digitale Signaturer

på

Ressourcebegrænsede enheder

Martin Hamann c971662
Jesper Stocholm Pedersen c960401

Polyteknisk Midtvejsprojekt
Under vejledning af Jørn Møller Jensen
Institut For Matematik
Danmarks Tekniske Universitet
11. januar 2001

Indhold

Figurer

Resumé

Udbredelsen af Internettet har medført et ønske om, at anvende det til elektronisk handel, samt kommunikation mellem borgere og offentlige myndigheder. Dette, kombineret med øget anvendelse af mobile enheder, såsom mobiltelefoner og PDA'ere, har styrket behovet for metoder til sikker identificering af brugere på Internettet.

Digitale signaturer spiller en meget stor rolle i opfyldelsen af dette behov. Den matematiske teori er ikke ny, men digitale signaturer har længe været forbeholdt lukkede, proprietære systemer som f.eks. home-banking. Denne rapport fokuserer specielt på digitale signaturer på ressourcebegrænsede enheder.

Kapitel 1

Forord

Nærværende rapport omhandler samfundsmæssige-, matematiske og implementeringsmæssige aspekter af digitale signaturer, med fokus på ressourcebegrænsede enheder. Rapporten dokumenterer tillige udviklingen af en komplet infrastruktur med klient og server.

Efter indledningen i det følgende kapitel gennemgår vi i kapitel 3 de umiddelbare fordele ved digitale signaturer. Herefter fortæller vi om de politiske overvejelser, der har været gjort omkring digitale signaturer, og slutteligt ser vi på de juridiske forhold, herunder den nye lovgivning om digitale signaturer.

Kapitel 4 gennemgår de tekniske overvejelser, vi skal igennem, før vi praktisk kan implementere digitale signaturer. Vi lægger vægt på at forklare opbygningen af selve strukturen og motiverer vores valg af teknologi og matematik.

I kapitel 5 ser vi indledningsvist på udvalgte dele af den matematiske gruppeteori, for efterfølgende at gennemgå matematikken bag digitale signaturer baseret på elliptiske kurver.

Kapitel 6 dokumenterer den applikation, vi har udviklet for at demonstrere de behandlede begreber. Vi beskriver de valg, vi løbende har foretaget i udviklingsprocessen og deres betydning for applikationens endelige udformning. Endelig fokuserer vi på nogle vigtige tekniske detaljer.

Herefter samler vi op på de ovennævnte kapitler, og perspektiverer vores arbejde. Vi ser på, hvad det er lykkedes os at gennemføre, og hvordan vores projekt senere kan videreføres.

Bagerst forefindes appendices indeholdende *Lov om elektroniske signaturer*,

en oversigt over Javaklasser og endeligt kildekoden til vores applikation.

Vi anbefaler, at kapitlerne i rapporten læses i ovennævnte rækkefølge. For læsere med særlig interesse i enkelte emner, er det dog muligt at dykke ned i de relaterede afsnit og læse dem særskilt. Dette forudsætter dog et grundlæggende kendskab til emnet, herunder specielt terminologien.

Læserens forudsætninger antages at omfatte forståelse for grundlæggende talteoretiske begreber, eksempelvis opnået ved at følge et universitetsfag i kodningsteori, algebra eller kryptografi.

Rapporten er baseret på et Polyteknisk Midtvejsprojekt på Institut for Matematik på Danmarks Tekniske Universitet, under vejledning af Jørn Møller Jensen. Til rapporten er knyttet en hjemmeside på adressen

`http://stocholm.dk/pmp`

Lyngby, 11. januar 2001

Martin Hamann

Jesper Stocholm Pedersen

Kapitel 2

Indledning

"En forretningsmand sidder i Kastrup Lufthavn, da han møder en god bekendt. Samtalen falder på aktiekøb, og hun anbefaler ham, at se nærmere på en bestemt aktie, men råder ham til at handle hurtigt. Forretningsmanden tager hendes råd til efterretning, og går straks hen til en offentlig Internetterminal. Han logger ind på Fondsbørsens hjemmeside for at se nærmere på aktierne. Han beslutter sig for at købe et antal aktier og indtaster detaljer om handlen. Inden handlen gennemføres, bliver han bedt om at indsætte sit Smart Card i terminalen. Han indsætter kortet og indtaster sin PIN-kode. Kort efter bekræftes handlen og han går ud til sit fly ..."

Ovenstående er et tænkt men ikke urealistisk eksempel. For at denne situation kan blive en realitet, er der en række krav til kommunikationen mellem forretningsmanden og fondsbørsen.

For det først skal begge parter være sikre på hinandens identitet. For det andet skal begge parter have sikkerhed for, at ingen har ændret i deres kommunikation undervejs. For det tredje må ingen af parterne senere have mulighed for at nægte, at handlen har fundet sted.

Disse tre krav kan alle opfyldes ved anvendelse af digitale signaturer, og beskrivelsen af dette og de omkringliggende problemstillinger er kernen i denne rapport.

Et centralt aspekt i vores eksempel er, at handlen kan gennemføres uafhængigt af, hvor forretningsmanden befinder sig. Han er således ikke afhængig af at være hjemme på sit kontor ved sin stationære PC, men han kan udføre handlen via enhver computer med Internetadgang. Denne uafhængighed indebærer et krav om mobilitet, dvs. at han skal kunne identificere sig overfor

Fondsbørsen, uanset hvilken PC han sidder ved. Han skal altså kunne opbevare beviset for sin identitet på f.eks. sin mobiltelefon, en PDA eller et Smart Card, enheder som vi under ét betegner *ressourcebegrænsede enheder*.

Anvendelsen af de ressourcebegrænsede enheder til opbevaring af bevis for identitet indebærer nogle åbenbare fordele for forretningsmanden. Han vil ikke længere være bundet af én PC, hvor der er installeret software fra hans bank. Han kan nøjes med at medtage et Smart Card med beviset for sin identitet.

Desværre medfører anvendelsen af Smart Card'et nogle udfordringer i implementeringsfasen. Ressourcebegrænsede enheder er i sagens natur begrænsede i forhold til stationære PC'er, og dette betyder, at man grundigt skal overveje, hvilken programmeringsteknik man vil anvende, hvilke algoritmer der skal implementeres og hvilken matematisk teknologi, der skal ligge til grund for applikationen.

Vi arbejder udfra følgende målsætning. Vi vil gennemgå fordele og ulemper for brugerne af digitale signaturer. Vi vil gennemgå de indledende overvejelser, der ligger forud for en implementering af digitale signaturer. Vi vil analysere den matematiske teori og implementere den i praksis.

Kapitel 3

Digitale signaturer

Vi giver i dette kapitel et overblik over digitale signaturer og de forskellige forudsætninger, krav og forventninger til teknologien. Vi belyser de umiddelbare fordele ved indførsel af digitale signaturer og beskriver de overordnede overvejelser, der ligger til grund for overhovedet at påtænke at indføre digitale signaturer. Slutteligt gennemgår vi nogle af de juridiske aspekter, herunder den nye lovgivning på området.

3.1 Definitioner

En digital signatur er en digital ækvivalent til den almindelige håndskrevne underskrift. En digital signatur er unikt knyttet til én person, og kan kun genereres af netop denne person. Derudover kan man ud fra en digital signatur bestemme identiteten af underskriveren, og ændringer i det signerede dokument kan opdages.

De informationer, der bruges til verificering af en signatur, benævnes *et certifikat*. Certifikater udstedes og administreres af et *Nøglecenter*[1]. Vi vender tilbage til dette i kapitel 4.

[1]Den engelske betegnelse er *Certificate Authority (CA)*, og som dansk oversættelse ses *Nøglecenter*, *Nøgleudsteder* eller *Certificeringscenter*. Vi bruger betegnelsen *Nøglecenter*.

3.2 Fordele ved digitale signaturer

Der er flere fordele forbundet med brugen af digitale signaturer fremfor håndskrevne signaturer. Fordelene omfatter en effektivisering af arbejdsgange i det offentlige og i private virksomheder, sikker identifikation af køber og sælger ved elektronisk handel, sikring af kommunikation imellem borgerne, etc.

En stor del af arbejdet i amter, kommuner og private virksomheder, består i håndtering af kommunikation og overførsel af papirer mellem forskellige parter. En lang række af disse arbejdsgange kan effektiviseres eller helt overflødiggøres ved indførsel af digitale signaturer. Langt de fleste dokumenter findes allerede lagret på digital form og digitale signaturer gør det muligt at behandle disse dokumenter med samme sikkerhed som fysiske dokumenter og samtidig frigive økonomisk kostbare ressourcer.

For borgerne er der også fordele ved udbredt brug af digitale signaturer. I dag tilbyder mange offentlige myndigheder information via deres hjemmesider på Internettet, men der er sjældent tale om decideret tovejskommunikation. Med digitale signaturer er det muligt for såvel borgerne som de offentlige myndigheder, at være sikre på identiteten af modparten. Dette åbner mulighed for en række fleksible tjenester, hvor der kan indgås bindende aftaler. Borgerne får mulighed for at kommunikere med det offentlige via Internettet døgnet rundt og uanset hvor de befinder sig. Alene at kunne undgå personligt fremmøde ved den offentlige myndighed vil således være en fordel for forbrugerne. De offentlige myndighed kan indføre en højere grad af selvbetjening, og det er dermed muligt at skære ned på behandlingstiden for administrative sager. Dette kan f.eks. være i forbindelse med ændring af selvangivelse, ændring af skatteforhold, flyttemeddelelser, medicinbestilling eller stemmeafgivelse.

Digitale signaturer finder som nævnt anvendelse ved udveksling af oplysninger over usikre netværk, som eksempelvis Internettet. Et vigtigt eksempel herpå er elektronisk handel. I dag foregår handel via Internettet typisk ved, at man indtaster navn, adresse og kreditkort-oplysninger på en hjemmeside, hvorefter man får lov til at handle. Problemet i dette er, at virksomheden ikke har nogen sikkerhed for, at de angivne oplysninger er rigtige. Det er tvivlsomt, om virksomheden overhovedet opdager, hvis adressen ikke eksisterer. Forbrugeren kan også indtaste falske kreditkortnumre, blot de opfylder nogle krav til antallet af cifre og eventuelle kontrolcifre. Drejer købet sig om on-line tjenesteydelser, software eller lignende, vil virksomheden ikke have mulighed for at undersøge, hvem der har afgivet de falske oplysninger. Selv hvis der er tale om fysiske varer, der bliver leveret til en adresse, opstår der problemer,

hvis forbrugeren nægter at have bestilt de pågældende varer. Digitale signaturer medfører *uafviselighed*, idet underskriveren af en digital signatur ikke senere kan undsige sig at have lavet signaturen.

Elektronisk handel omfatter dels handel mellem virksomheder og forbrugere, men i stadig stigende grad også handel mellem virksomheder. I takt med den stigende globalisering er behovet for at kunne kommunikere med virksomheder i andre dele af verden steget. En virksomheds primære marked er ikke længere begrænset til de geografisk nærmest beliggende virksomheder, men handelspartnerne kan være fordelt over hele verden. Hvis et værft eksempelvis i Polen skal sælge skibe til et rederi i Panama, vil det være optimalt, hvis parterne ikke behøver mødes, hver gang de skal tilslutte sig en revideret kontrakt. Digitale signaturer kan således også vise sig en stor fordel for virksomheder, der ikke har nogen direkte forbrugerkontakt.

Kommunikation borgerne imellem er et ofte overset område for digitale signaturer. De fleste brugere af e-mail har ikke behov for at kunne signere eller kryptere deres e-mail, da de fleste bruger e-mail til korte uformelle beskeder. Men i takt med at den private anvendelse af Internettet bliver mere seriøs, vil den almindelige bruger have et stigende behov for at signere eller kryptere e-mail.

3.3 Politiske overvejelser

Selvom der umiddelbart er store fordele ved indførslen af digitale signaturer, er der også store bekymringer for konsekvenserne, og overvejelser om hvorvidt det er hensigtsmæssigt.

Et væsentligt punkt i debatten er problemstillingen omkring overvågning af borgerne. Debatten har været meget følelsesladet, da mange frygtede, at en statslig udstedt digital signatur ville indeholde bagdøre, der ville give politiet eller andre myndigheder mulighed for at overvåge borgernes kommunikation.

Forskningsministeriet og IT-sikkerhedsrådet har afholdt høringer om kryptering og digitale signaturer, og et af debatpunkterne har været hvilken slags software, der skal bruges til implementering af digitale signaturer i samfundet. Debatten har blandt andet fokuseret på, hvorvidt den anvendte software

skal være *Open Source*[2] eller proprietær software. En løsning baseret på Open Source software vil gøre diskussionen om bagdøre irrelevant, da programmets kildekode bliver fuldt offentlig. På den anden side har det været fremhævet, at en tvungen Open Source løsning vil afholde de store softwarefirmaer fra at gå ind på markedet. Dette vil medføre, at implementeringen af digitale signaturer bliver udskudt i forhold til regeringens oprindelige ønske. Indtil videre har man ladet markedet vælge hvorvidt løsningerne skal baseres på Open Source software.

En anden del af debatten er problematikken omkring tilliden til en digital signatur. Vi opfatter vores egen underskrift, som den ultimative validering af, at det vitterligt er os, der har underskrevet et givet dokument, og dermed har været fysisk tilstede. Denne tryghed og tillid vi har til underskriften, er svær at overføre til den digitale signatur. Ofte vil vi ikke være fysisk tilstede, ved f.eks. en offentlig myndighed, når vi underskriver digitalt. Dette er netop en af de vigtigste muligheder, som digitale signaturer åbner op for.

For at kunne sidestille håndskrevne underskrifter med digitale signaturer, er det vigtigt at få overført følgende karakteristika fra den håndskrevne underskrift:

1. Den er unikt knyttet til underskriveren
2. Det er muligt at fastslå identiteten af underskriveren udfra underskriften
3. Ejeren af underskriften er den eneste, der kan lave netop denne underskrift
4. Den beviser, at underskriveren har haft det fysiske dokument i hænde.

Ved signering af digitale dokumenter får vi endvidere brug for følgende egenskab:

5. En underskrift kan knyttes til et dokument, så ændringer i dette kan opdages.

En digital signatur skal omfatte alle ovenstående karakteristika. De første 4 vil sidestille den med den håndskrevne underskrift, mens den 5. vil tilføje en

[2]Open Source betyder "fri kildekode"og Open Source software dækker over software hvor kildekoden til programmet bliver offentliggjort. Dette bremser ofte den økonomiske vinding ved softwareudviklingen, men til gengæld kan brugerne få større tillid til at softwaren ikke indeholder uønskede elementer til overvågning etc.

ny men nødvendig egenskab. Problemet ved udveksling af digitale dokumenter, er at alle kan ændre i de bits, der definerer dokumentet. Netop derfor er punkt 5 en absolut nødvendighed ved digitale signaturer.

Den sidste politiske overvejelse, vi kommer ind på, er diskussionen om, hvem der skal opbygge den tekniske infrastruktur.

En mulighed er at lade staten opbygge infrastrukturen. Derved sikres det, at infrastrukturen kommer på det ønskede tidspunkt, at den bliver landsdækkende og at den opfylder de ønsker staten måtte have. Desværre betyder det også, at staten skal betale de ganske betragtelige etableringsomkostninger. Dette er den løsning, den finske regering har valgt.

Alternativet er at lade markedskræfterne styre udviklingen, som det er sket ved udvikling af de første digitale signatursystemer, som kendes fra homebanking systemer. Her er det bankerne, der har betalt for opbygningen af infrastrukturen, der dog derfor begrænser sig til bankens egne kunder. Herved betaler staten ikke penge for det, men kan til gengæld heller ikke styre udviklingen.

I Danmark har regeringen valgt at lade det være op til markedet at opbygge infrastrukturen. Fra statens side er der opsat nogle krav, der skal opfyldes af virksomheder, såfremt de vil påtage sig at opbygge og drive denne infrastruktur. Det er Forskningsministeriets håb, at konkurrencen på markedet vil sikre, at det sker i løbet af relativ kort tid. Der er da også allerede store firmaer, som KMD, Post Danmark og Tele Danmark, der har meldt sig på markedet for udstedelse af certifikater.

Disse emner er yderligere uddybet i Forskningsministeriets rapport [7] om praktisk brug af kryptering og digitale signaturer.

3.4 Juridiske overvejelser

Som det fremgår af diskussionen ovenfor, er det juridiske aspekt af digitale signaturer meget vigtigt. I det følgende ser vi nærmere på lovgivningen på området. *Lovgivning om Elektroniske signaturer* [12], trådte i kraft 1. oktober 2000, og er gengivet i bilag A. Loven omhandler *elektroniske* fremfor *digitale* signaturer, hvilket skyldes et ønske om at gøre loven uafhængig af hvilken teknologi, der bruges til selve implementeringen.

Loven definerer krav til et nøglecenter og til de procedurer den varetager. Vi opridser de vigtigste elementer af lovgivningen.

- ☐ Definition af et *kvalificeret certifikat*.
- ☐ Krav til underskrivning af elektroniske dokumenter.
- ☐ Krav til et nøglecenter.
- ☐ Ansvarsplacering mellem et nøglecenter og forbruger.

Den første del af loven definerer et kvalificeret certifikat. Hovedpunkterne i denne definition går på at føre de i afsnit 3.3 nævnte karakteristika for håndskrevne underskrifter over til certifikater. En række af kravene til kvalificerede certifikater er ikke direkte relateret til sikkerheden bag digitale signaturer, men specificerer f.eks. gyldighedsperiode og beløbsbegrænsninger ved anvendelse [12] §3-4. Det er desuden værd at bemærke, at et kvalificeret certifikat kun kan udstedes til en person, og ikke til en virksomhed.

Dernæst beskriver loven en række krav til et nøglecenter. Denne del af loven er skrevet så teknologineutralt som muligt og der bruges meget overordnede vendinger. Bl.a. skal et nøglecenter *anvende betryggende [...] procedurer, anvende pålidelige [...] produkter* og *beskæftige personale med [...] ekspertise* [12] §5, stk. 1-3.

Som før nævnt er der ingen krav om, at et nøglecenter skal offentliggøre kildekoden til den bagvedliggende software, med andre ord behøver nøglecentret ikke at bruge Open Source software. Der er derimod krav om, at nøglecentret skal offentliggøre oversigter over hvilke interne procedurer den følger, og hvordan den formelt overholder kravene i lovgivningen.

En vigtig del af lovforslaget om elektroniske signaturer er placering af ansvar. Det er vigtigt, at der ikke kan drages tvivl om dette i tilfælde af uoverensstemmelser. Derfor bruges en stor del af lovforslaget på at definere, i hvilke tilfælde brugeren er ansvarlig for fejl, og i hvilke tilfælde ansvaret falder på nøglecentret.

En bruger gøres ansvarlig for fejl, hvis hendes private nøgle kompromitteres, og da kun hvis hun samtidigt undlader at tilbagetrække sit certifikat. Fra det øjeblik certifikatet bliver trukket tilbage, ligger ansvaret hos nøglecentret. Disse regler er de samme, som en bruger er beskyttet af, hvis der

betales med et betalingskort. Brugeren er altså beskyttet af *Lov om visse betalingsmidler*, se [13] §11, stk. 2-6.

Nøglecentret er primært ansvarlig, hvis der er fejl i de certifikater, der udstedes til brugerne. Nøglecentret er også ansvarlig, hvis den tilbagetrækningsservice, som de jf. loven skal tilbyde, ikke fungerer, eller hvis oplysningerne på certifikatet ikke er korrekte.

Nøglecentret er ikke ansvarlig for køb med og anvendelse af certifikatet, hvis denne anvendelse falder udenfor certifikatets anvendelsesområde. Det er ikke i loven specificeret, hvem der bærer dette ansvar. Det er dog tænkeligt, at det vil være personen eller virksomheden, der fejlagtigt har accepteret, at certifikatet anvendes udenfor det tilsigtede område, der bærer ansvaret. Igen er dette analogt til [13], hvor det er butikken, der har accepteret handel med et udløbet eller spærret kreditkort, der har det økonomiske ansvar.

Desuden er der en række krav i lovforslaget, der søger at sikre, at det enkelte nøglecenter vitterligt har mulighed for, at bære denne ansvarsbyrde. Der stilles bl.a. krav til, at nøglecentret "*[...] til stadighed [bør] have tilstrækkelige økonomiske ressourcer til at drive virksomhed i overensstemmelse med bestemmelserne i denne lov, herunder til at opfylde erstatningsmæssige forpligtelser i henhold til loven [...]*" [12] §5, stk. 5. Med andre ord vil man sikre sig, at et nøglecenter ikke kan komme i likviditetsproblemer i tilfælde af erstatningsansvar.

3.5 Opsamling

Som vi har set i dette kapitel, er vejen til at implementere digitale signaturer i Danmark meget lang. Der er en lang række fordele ved indførsel af digitale signaturer, men der er også store udfordringer i at indføre dem med succes. Disse udfordringer inkluderer ikke alene det juridiske aspekt. Lige så vigtigt er også, at aspektet med forbrugerens rolle undersøges, dvs. spørgsmålet om hvorvidt forbrugeren i det hele taget *ønsker* at anvende digitale signaturer.

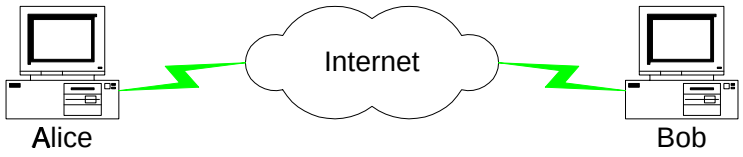
Kapitel 4

Digitale signaturer i praksis

Fra at beskrive digitale signaturer i et større perspektiv i foregående kapitel, fokuserer vi nu på, hvordan vi realiserer tankerne i praksis. I dette kapitel introducerer vi den nødvendige terminologi, og vi beskriver, hvordan digitale signaturer fungerer i praksis.

4.1 Alice og Bob

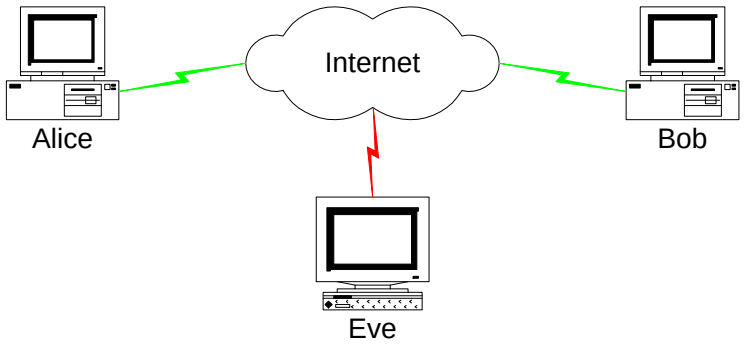
Vi forestiller os to parter, der kommunikerer over et usikkert netværk, eksempelvis Internettet. Den initierende part, Alice, sender beskeder til den modtagende part, Bob. Dette er illustreret på figur 4.1.



Figur 4.1: Alice og Bob

Desuden findes en modarbejdende tredje part, Eve[1], som forsøger at opsnappe kommunikationen mellem Alice og Bob. Eve prøver enten at ændre i kommunikationen eller udgive sig for at være den ene af de to parter. Sammenhængen er vist i figur 4.2.

[1] I litteraturen benævnes modparten ofte Charlie, Eve eller Mallory. Charlie synes brugt, for at holde den alfabetiske rækkefølge påbegyndt med navne som Alice og Bob, mens Eve og Mallory bruges udfra en mere snedig betragtning, idet Eve er afledt af *Eavesdropper* (Aflytter) og Mallory af *Malicious* (Ondskabsfuld). Vores valg af Eve er arbitrært.



Figur 4.2: Alice, Bob og Eve

Når Alice sender en besked til Bob med en digital signatur, siger vi at Alice *signerer* beskeden. Dette gør hun ved hjælp af sin *private nøgle*, som kun hun er i besiddelse af. Da Alice er den eneste, der har dén private nøgle, er hun den eneste, der kan lave en signatur, der beviseligt kommer fra hende. Signaturen påhæftes[2] beskeden inden den sendes til Bob.

Når Bob modtager beskeden med den påhæftede digitale signatur, kan han undersøge om signaturen virkelig er lavet af Alice og dermed om beskeden kommer fra Alice. Vi siger, at Bob *verificerer* signaturen. Til dette bruger Bob Alice's *offentlige nøgle*, som alle har adgang til. Alice's private og offentlige nøgler passer sammen matematisk, således at der til hver offentlig nøgle hører netop én privat nøgle og omvendt. Alice kan vælge at offentliggøre sin offentlige nøgle på sin hjemmeside eller hun kan sende den med, når hun sender en besked. Hun kan også vælge at offentliggøre nøglen via en offentlig nøgleserver eller nøglecenter. Dette vender vi tilbage til i næste afsnit.

Eve kan, på samme vis som Bob, verificere, at det er Alice der har sendt beskeden, men Eve kan ikke lave en signatur, der ser ud, som om den kommer fra Alice, da Eve ikke har Alice's private nøgle. Da signaturen er dannet udfra den besked Alice sendte, kan Eve heller ikke ændre i beskeden, uden af Bob vil opdage det, når han forsøger at verificere.
Når der bliver involveret flere parter, bliver systemet hurtigt uoverskueligt, og vi ser derfor på hvordan vi kan formalisere strukturen i systemet. En sådan

[2] Teknisk er der flere måder at gøre det på. Alice kan vælge blot at skrive signaturen lige efter beskeden (påhæfte den), eller signaturen kan sendes med i en seperat fil (vedhæfte den).

struktur er nærmere beskrevet i følgende afsnit.

4.2 Public Key Infrastructure

En *Public Key Infrastructure (PKI)* er en formel beskrivelse af de førnævnte parter, deres indbyrdes forhold og procedurer for signering og verificering.

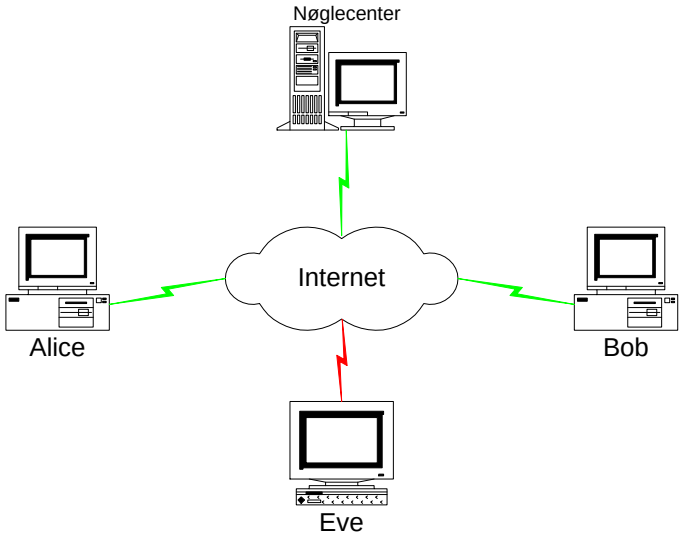
Da Alice og Bob ikke som udgangspunkt kan stole på hinanden, indfører vi en ny part, kaldet et *Nøglecenter*. Et nøglecenter vil i praksis ofte være enten globalt eller nationalt. *VeriSign* i USA er et eksempel på et globalt nøglecenter, hvorimod den danske virksomhed *KMD* sandsynligvis vil rette sig imod det danske marked. Et nøglecenter har til formål at garantere, at de enkelte parter, er dem de giver sig ud for at være. Nøglecentret skal derfor som udgangspunkt have sikkerhed for Alice's identitet. Dette opnås ved at Alice møder op hos nøglecentret med eksempelvis kørekort, pas eller lignende.

Nøglecentret kan nu lave den private nøgle og tilhørende offentlige nøgle. Den private nøgle videregives til Alice. Den offentlige nøgle lagres i form af et *certifikat*, som udover nøglen indeholder information om Alice's identitet[3]. Nøglecentret signerer certifikatet med sin egen private nøgle. Da nøglecentret netop har sikret sig bevis for Alice's identitet, kan nøglecentret garantere, at kun Alice har fået udleveret den private nøgle. Hvis den private nøgle bliver kompromitteret, er Alice forpligtet til at kontakte nøglecentret, som så trækker certifikatet tilbage. Dette vender vi tilbage til senere.

Når Bob modtager en signeret besked fra Alice, kan han verificere, at Alice virkeligt er afsenderen. Bob tager kontakt til nøglecentret og beder om at få udleveret Alice's certifikat. Da kommunikationen mellem nøglecentret og Bob foregår over det usikre netværk, kan Bob ikke være sikker på autenciteten af det modtagne certifikat. Når Bob modtager certifikatet, forsøger han derfor først at verificere, at certifikatet kommer fra nøglecentret. Vi antager, at nøglecentrets eget certifikat, er indlagt i den software som Bob bruger til verificering. Bob har derfor mulighed for at verificere, at Alice's certifikat er underskrevet af nøglecentret. Lykkes denne verificering, har Bob nu Alice's offentlige nøgle og kan undersøge hvorvidt beskeden er signeret af Alice. Denne PKI er illustreret på figur 4.3.

Vi har ikke opnået explicit tillid til Alice ved indførslen af nøglecentret. I

[3]Certifikatet indeholder også et unikt ID felt samt teknisk information om det brugte signeringssystem. Dette vil vi dog ikke komme nærmere ind på her.



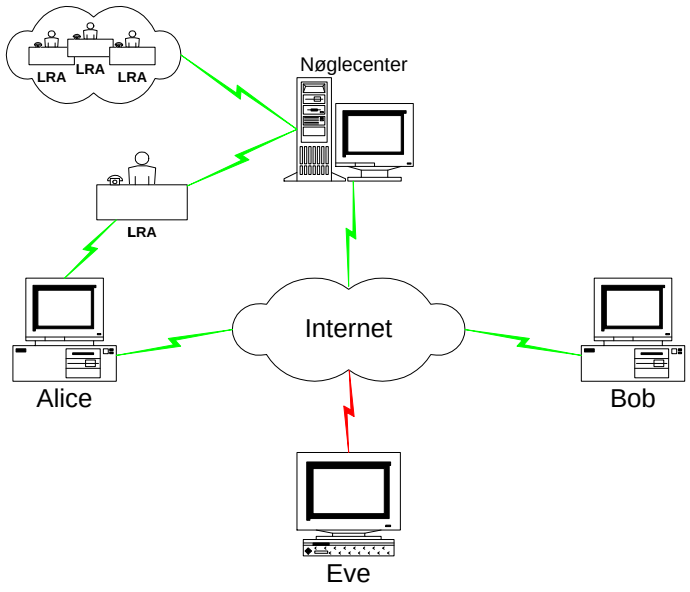
Figur 4.3: En simpel PKI med ét nøglecenter

stedet har vi flyttet tillidsproblemet fra at ligge imellem Alice og Bob til at ligge imellem Bob og nøglecentret. I den her beskrevne PKI, antager vi imidlertid at vi har fuld tillid til nøglecentret. At dette ikke altid er tilfældet, og indebærer yderligere komplikationer, vender vi tilbage i afsnit 4.3.

Som nævnt skal nøglecentret have bevis for identiteten af Alice, før et certifikat kan udstedes. Dette kræver, at personen møder personligt op hos nøglecentret. Det personlige fremmøde er kun nødvendigt, når certifikatet udstedes, men er alligevel uhensigtsmæssigt. Vi vil gerne have PKI'en til at fungere uanset fysiske afstande, og får derfor brug for en lokal repræsentant for nøglecentret.

Vi indfører derfor en såkaldt *Local Registration Authority (LRA)*. Alice kan nu gå til LRA'en for at få udstedt sit certifikat, der efterfølgende bliver sendt til nøglecentret. En LRA kan f.eks. befinde sig i en virksomhed, på et kommunekontor eller lignende. Vi antager, at LRA'en og nøglecentret har deres egne kommunikationskanaler, eller på forhånd har udvekslet nøgler, og således kan udveksle informationer over en sikker kanal. Et nøglecenter kan have et helt netværk af LRA'er og det får vores PKI til at se ud som på figur 4.4.

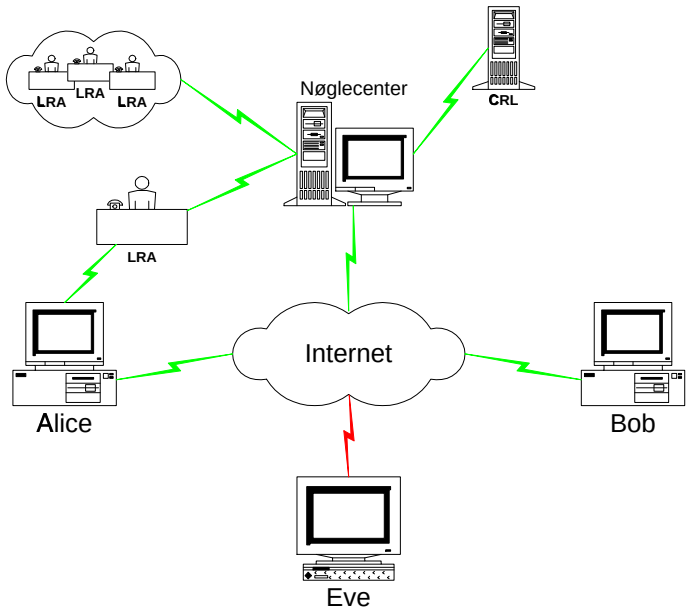
Når Bob modtager en signeret besked, kan han vælge enten at hente certifikatet hos nøglecentret, som tidligere beskrevet, eller bruge en lokal kopi,



Figur 4.4: En PKI med ét nøglecenter og flere LRA

som han tidligere har hentet. Vælger Bob det første, vil nøglecentret kontrollere, at certifikatet ikke er udløbet eller trukket tilbage, inden det sendes til Bob. Vælger Bob at bruge en lokal kopi, skal han selv forespørge nøglecentret, om certifikatet er gyldigt. Hvis certifikatet ikke er gyldigt, skal signaturen afvises. Vi har derfor brug for en liste over de tilbagetrukne certifikater, en såkaldt *Certificate Revocation List (CRL)*. Denne liste kan enten ligge hos nøglecentret eller på en seperat server. I begge tilfælde tager Bob kontakt til nøglecentret, som så slår certifikatet op i CRL'en. Vores PKI ser nu ud som på figur 4.5. Vi bemærker, at vi aldrig kan undgå, at skulle kontakte nøglecentret, men vi kan undgå, at skulle downloade hele certifikatet hver gang.

Vi mangler nu kun én ting, for at vores PKI er komplet, nemlig en mulighed for at kunne tidsstemple en besked. Dette er specielt vigtigt i tilfælde af financielle transaktioner, hvor det er essentielt, at vi efterfølgende kan fastslå, hvornår en givet aftale eller kontrakt er indgået. Denne service foretages af en *Time Server*. Igen kan denne service leveres af nøglecentret, eventuelt baseret på en seperat server. Ved signering af en besked, tager Alice kontakt til en Time Server, som tilføjer beskeden et tidsstempel. Da dette tidsstempel er signeret af Time Serveren, kan Bob, med de nødvendige nøgler, verificere at tidsstemplet i beskeden er korrekt. Den samlede PKI med Alice, Bob, Eve,



Figur 4.5: En PKI med Nøglecenter, LRA og én CRL

Nøglecenter, LRA, CRL og endelig Time Server ses i figur 4.6.

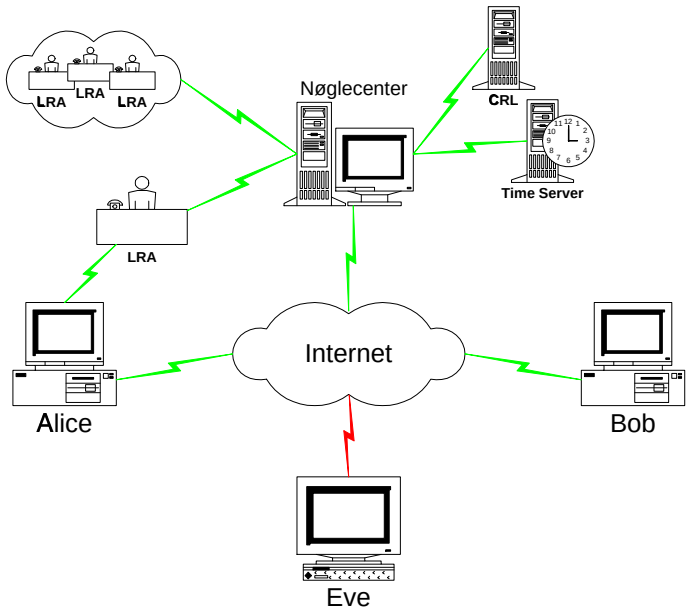
Vi har beskrevet en komplet PKI til digitale signaturer. Vi har sikret os, at certifikater bliver signeret af et nøglecenter, at beskeder kan tidsstemples, samt at certifikaternes gyldighed bliver verificeret.

Hvis PKI'en skal skaleres, til f.eks. internationalt plan, opstår der visse komplikationer, som vi belyser herunder.

4.3 Problemer med PKI

Den ovenfor beskrevne PKI er meget forenklet i forhold til en virkelig implementation. Eksempelvis vil der ofte være flere end blot ét nøglecenter. Dette giver umiddelbart et problem, da Bob og Alice ikke nødvendigvis stoler på det samme nøglecenter - specielt ikke, hvis Alice og Bob bor i forskellige lande. Bob kan måske ikke verificere Alice's besked, da han ikke har tillid til hendes nøglecenter. Der er grundlæggende to muligheder for at løse dette; *krydscertificering* og en *hierarkisk løsning*.

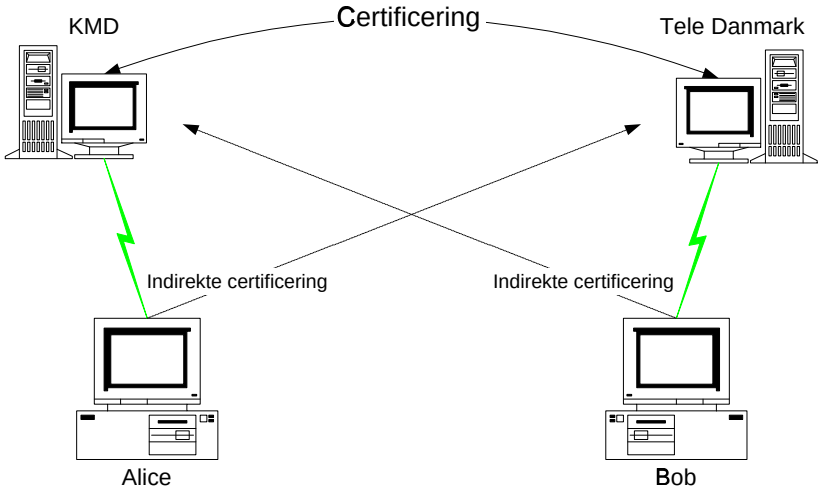
Krydscertificering indebærer, at hvert enkelt nøglecenter certificerer de andre



Figur 4.6: En PKI med Nøglecenter, LRA, CRL og Time Server

nøglecentre. Med denne løsning, stoler Bob indirekte på Alice's nøglecenter, fordi hans eget nøglecenter stoler på Alice's nøglecenter. Nu kan Bob verificere beskeden fra Alice. I Danmark forventer man, at denne model bliver en realitet. Det er vurderingen, fra bla. IT-Sikkerhedsrådet, at vi i Danmark ender med 3-4 nøglecentre, og det er nærliggende, at de verificerer hinanden. Som beskrevet i afsnittet om lovgivningen i kapitel 3, skal alle nøglecentre i Danmark overholde de samme retningslinier, og det bør ikke være et problem at verificere andre nøglecentre, der overholder disse krav. Sammenhængen er vist på figur 4.7

En anden løsningsmodel er en hierarkisk løsning, hvor ét nøglecenter certificerer alle andre nøglecentre. Et *hoved-nøglecenter* får dermed samme status for de andre nøglecentre, som de har for en almindelig bruger som Alice eller Bob. Den amerikanske virksomhed *VeriSign* har en de facto position på verdensmarkedet som hoved-nøglecenter, hvor VeriSign udsteder certifikater til virksomheder og personer i hele verden. I Europa prøver den belgiske virksomhed *GlobalSign*, at få en tilsvarende status. I Danmark har Tele Danmark valgt at lade sig certificere af GlobalSign, hvilket betyder, at Tele Danmarks certifikater kan bruges af alle brugere, der også har GlobalSign som nøglecenter. Principppet i hierarkisk struktur ses af figur 4.8



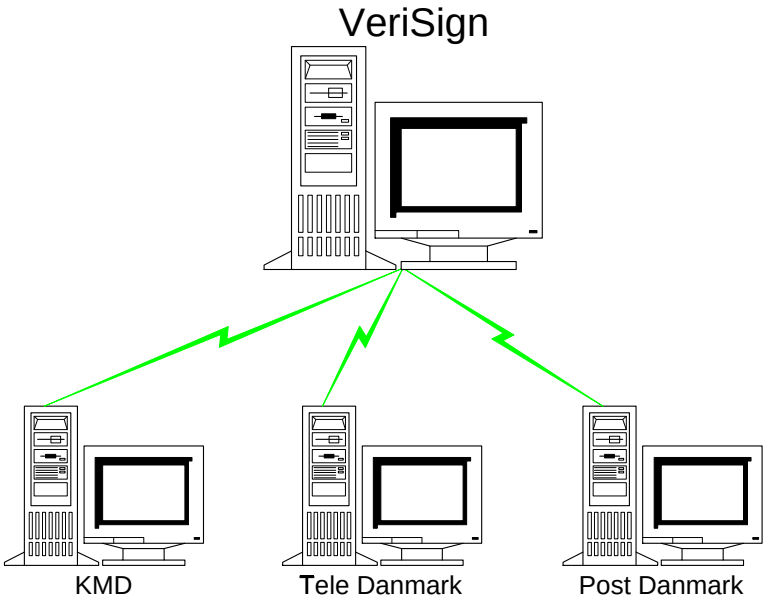
Figur 4.7: Krydscertificering

Problemerne er dog ikke ovre med en hierarkisk opbygning af vores PKI. I takt med at hierarkiet udbygges, mister man tilliden. Hvis Alice er certificeret af British Telecom, der er certificeret af VeriSign, der har certificeret Post Danmark, der har certificeret Bob, er det ikke uden videre givet, at Bob har tillid til, at Alice's nøglecenter har certificeret hende korrekt.

4.4 Enheder med begrænsede ressourcer

Vores PKI er defineret så bredt, at den nemt kan skaleres, hvis den skal implementeres i en virksomhed, et helt land, eller på Internettet generelt. Vores PKI er heller ikke indskrænket hvad angår den anvendte teknologi, men kan beskrive inhomogene miljøer, hvor klienterne omfatter stationære computere, håndholdte computere, mobiltelefoner eller *Smart Cards*. I denne rapport fokuserer vi på det mobile aspekt, hvor PKI'en anvendes uafhængigt af en stationær computer.

Dette indebærer nogle udfordringer til teknologien bag vores PKI. De mest almindelige algoritmer til digitale signaturer, det vil sige DSA, ElGamal (se kapitel 5) og RSA, stiller store krav til computerens regnekraft og hukommelse. Dét er et problem, da en mobil enhed har markant mindre regnekraft og hukommelse end en stationær computer. Dette er specielt tilfældet, hvis enheden er en mobiltelefon eller et Smart Card, hvor alle regneoperationer fo-



Figur 4.8: Hierarkisk PKI

regår i en chip på et plastikkort. I det følgende vil vi fokusere på Smart Cards.

Anvendelse af et Smart Card har flere fordele i forhold til anvendelse af disketter eller magnetstribekort til opbevaring af nøglerne. Et Smart Card er *tamper resistant*, hvilket betyder, at data i specifikke områder på kortet ikke kan fjernes igen, og indholdet kan ikke læses uden at ødelægge disse data. Kortet er desuden modstandsdygtig overfor en lang række af fysiske angreb, såsom temperatursvingninger, magnetfelter, mekanisk adskillelse etc. Derfor er det et velegnet lagringsmedium til nøglerne, da de ellers bliver kompromitterede, hvis kortet tabes, eller på anden måde kommer uden for ejerens kontrol.

Funktionaliteten i et Smart Card er begrænset på mange områder set i forhold til andre enheder. Dette giver sig bedst udtryk i, at alt, det vil sige processor, arbejdshukommelse og lager, er samlet i én chip. Når chippen samtidig er maksimalt 25x25 mm med en tykkelse på 0,5 mm, er det klart, at kun en begrænset mængde funktioner kan implementeres[4]. En yderligere begrænsning er hastigheden, hvormed data læses fra og skrives til chipkortet. Det er derfor vigtigt, at datatrafikken til og fra Smart Card'et minimeres.

[4]Disse mål stammer fra Smart Card standarden ISO 7816

Med disse begrænsninger in mente er det vigtigt, at den anvendte algoritme opfylder følgende krav:

- ☐ Beregningsarbejdet ved signering og verificering skal være minimalt.
- ☐ Den nødvendige arbejdshukommelse under beregningsarbejdet skal være minimal.
- ☐ Den nødvendige lagerplads, typisk til den private nøgle, skal være minimal.
- ☐ Størrelsen af signaturer genereret på kortet skal være minimal, for at minimere overførselstiden fra Smart Card til f.eks. e-mail applikation.

Vi kan opfylde alle disse krav, ved at bruge en digital signatur algoritme baseret på *elliptiske kurver*, i stedet for mere almindelige algoritmer som RSA, DSA eller ElGamal. Som udgangspunkt er nøglerne ved anvendelse af elliptiske kurver mindre, hvilket uddybes i kapitel 5, afsnit 5.5. Derved reduceres beregningsarbejdet væsentligt og ydermere bevirker de små nøgler, at signering og verificering ikke kræver et stort lager, hvor midlertidige data kan henlægges.

De små nøgler er også en fordel, da den private nøgle skal *opbevares* på kortet. I forhold til DSA-signaturer er der pt. en forskelsfaktor i nøglestørrelse på 7 i forhold til digitale signaturer baseret på elliptiske kurver, og indenfor få år kan denne faktor komme op på 35, se [9][5]. Derfor er det vigtigt, at selve opbevaringen af den private nøgle ikke er et problem i sig selv. Slutteligt bevirker de mindre nøgler også, at signaturen formindskes, hvilket opfylder det sidste krav i listen ovenfor.

Som en sidste fordel ved at anvende elliptiske kurver kan nævnes, at da alle de ovenstående krav bliver opfyldt, så kræver udregningerne ikke nødvendigvis, at Smart Card'et indeholder en seperat *kryptografisk co-processor*[6] eller én meget hurtig processor. Dette betyder, at stykprisen på et Smart Card

[5]Det skal bemærkes, at denne kilde er skrevet af firmaet Certicom, der er den mest kendte udvikler af løsninger baseret på elliptisk kurve kryptografi. Tilhængere af traditionelle systemer som RSA, specielt firmaet RSA Security Inc. har stillet spørgsmålstegn ved disse forholdstal. Måske er tallene overdrevne, men konklusionen bliver den samme. Vi har derfor valgt at bruge sammenligningerne lavet af Certicom.

[6]En kryptografisk co-processor er en chip, der er specifikt designet til kryptografiske udregninger

bliver mindre, da man nu ikke længere er nødsaget til at anvende den seneste teknologi, men kan bruge Smart Cards med mindre kapacitet og ydelse. Dette har stor økonomisk betydning, ved implementering af en PKI, med mange brugere.

4.5 Opsamling

Vi har gennemgået opbygningen af en PKI, og vi har set, hvordan umiddelbart simple udvidelser af vores "verden"kan medføre problemer i forhold til at definere en sikker PKI. Vi har kort været inde på problemerne med krydscertificering og en hierarkisk Nøglecenter-struktur, men de problemer vi har påpeget står endnu tilbage at løse. Vi har motiveret valg af matematisk metode til udregning af digitale signaturer, nemlig elliptiske kurver, og dermed den elliptiske kurve-variant af signaturstandarden DSA.

Kapitel 5

Grupper, digitale signaturer og elliptiske kurver

Vi har i kapitel 4 konkluderet, at digitale signaturer baseret på elliptiske kurver er velegnede til implementering på ressourcebegrænsede enheder. ECDSA *Elliptic Curve Digital Signature Algorithm* er en standard for digitale signaturer baseret på elliptiske kurver, og dette kapitels overordnede mål er at beskrive matematikken i denne standard.

Kapitlet er opdelt i fire dele. Første del omhandler udvalgte dele af gruppeteorien. De tre sidste dele beskriver den matematiske opbygning af signaturer, ved først at se på *ElGamal* signaturer efterfulgt af den *digitale signatur algoritme (DSA)* og dennes elliptiske kurve variant *ECDSA*.

5.1 Indledende matematiske betragtninger

Vi starter med nogle talteoretiske betragtninger omkring grupper. Først beskriver vi grupper generelt, hvorefter vi fokuserer på en bestemt gruppe, hvor elementerne udgør punkter på en elliptisk kurve.

5.1.1 Grupper generelt

En gruppe er et abstrakt matematisk begreb, der omhandler operationer på elementer i en lukket mængde. En mængde G med en binær komposition[1] $\circ$ er en gruppe, hvis følgende fire betingelser er opfyldte:

1. Mængden G er lukket overfor $\circ$.

[1] En binær operation $\circ$ skrives formelt som $\{\circ : G \times G \mapsto G\}$

2. Der eksisterer et neutralt element $e \in G$, således at

$$e \circ a = a \circ e = a, \ for\ alle\ a \in G. \tag{5.1}$$

3. For alle $a \in G$ eksisterer der et inverst element a^{-1}, således at

$$a^{-1} \circ a = a \circ a^{-1} = e \tag{5.2}$$

4. Kompositionen $\circ$ er associativ, hvilket vil sige

$$a_1 \circ (a_2 \circ a_3) = (a_1 \circ a_2) \circ a_3, \ for\ alle\ a_1, a_2, a_3 \in G. \tag{5.3}$$

Gruppen bestående af mængden G og kompositionen $\circ$ betegnes $(G, \circ)$. Hvis G er en endelig mængde, kaldes antallet af elementer i G *ordenen af* G, og betegnes $ord(G)$ eller $|G|$. Er G ikke endelig, siges G at have uendelig orden. Gælder det, at $a \circ b = b \circ a$ for alle $a, b \in G$, siges gruppen at være *kommutativ* eller *abelsk*.

Ser vi på addition af tal i $\mathbb{Z}$, har vi en gruppe $(\mathbb{Z}, +)$. Det ses af følgende:

1. $\mathbb{Z}$ er en lukket mængde mht addition.
2. Det neutrale element i $\mathbb{Z}$ mht addition er 0, da det gælder, at $a + 0 = 0 + a = a$, for alle $a \in G$.
3. Det inverse element i $\mathbb{Z}$ mht addition er $-a$, da det gælder, at $a + (-a) = 0$, for alle $a \in \mathbb{Z}$
4. Associativitet i gruppen $(\mathbb{Z}, +)$ ses nemt.

Da addition i $\mathbb{Z}$ endvidere er kommutativ, er gruppen abelsk.

I en gruppe $(G, \circ)$ kan vi komponere et element med sig selv et antal gange, det vil sige

$$g \circ g \circ \ldots \circ g, \ for\ g \in G \tag{5.4}$$

Dette formuleres

$$g^n = \underbrace{g \circ g \circ \ldots \circ g}_{i\ alt\ n\ gange} \tag{5.5}$$

Vi definerer $\langle g \rangle$ som mængden af elementer, der frembringes af gentagne komponeringer af g med sig selv, det vil sige $\langle g \rangle = \{g^n | n \in \mathbb{Z}, g \in G\}$. Dette

giver os en undergruppe af G. Denne undergruppe siges at *blive frembragt* af g, og g kaldes et *primitivt element*.

Vi definerer nu en egenskab ved grupper, som vi har brug for. En gruppe $(G, \circ)$ er *cyklisk*, hvis der findes et element $g \in G$, således at $G = \langle g \rangle$. Ordenen af et element $g_i \in \langle g \rangle$ deler ordenen af G.

For at få en cyklisk gruppe, skal vi altså indskrænke mængden af elementer i gruppen, og dette gøres ved at introducere den cykliske gruppe $(\mathbb{Z}_p^*, \cdot)$, hvor p er et primtal. Denne gruppe indeholder kun elementer, der er indbyrdes primiske med p, men da p er et primtal, er elementerne $\{1, 2, \ldots, p-1\}$. Blandt elementerne findes mindst ét element, som genererer hele $\mathbb{Z}_p^*$. Ordenen af dette element α vil derfor være $p-1$, hvilket igen betyder, at $\alpha^{p-1} \equiv 1 \, mod \, p$. Et element af orden $p-1$ kaldes et *primitivt element*.

Vi vender os nu mod en specielt nyttig gruppe, baseret på elliptiske kurver.

5.1.2 Gruppen over elliptiske kurver

I det følgende definerer vi en *elliptisk kurve* i det endelige legeme $\mathbb{F}_p$ og beskriver egenskaberne ved en gruppe baseret på en elliptisk kurve.

Vi har et primtal $p > 3$. Vi definerer den elliptiske kurve $E(\mathbb{F}_p)$ som mængden af løsninger $x, y \in \mathbb{F}_p$ til kongruensen

$$y^2 \equiv x^3 + ax + b \, (mod \, p) \tag{5.6}$$

samt det abstrakte punkt $\mathcal{O}$ kaldet *uendelighedspunktet*[2]. Der er nogle begrænsninger på $a, b \in \mathbb{F}_p$, hvis løsningsmængden skal udgøre mængden i en gruppe. Det skal være opfyldt, at $4a^3 + 27b^2 \not\equiv 0 \, (mod \, p)$.

For at danne en gruppe skal vi have en mængde og en komposition, og kravene i afsnit 5.1.1 skal være opfyldte. Vores mængde er alle punkter på den elliptiske kurve $E(\mathbb{F}_p)$ og vores komposition er *addition*. Som udgangspunkt skal gruppen være lukket med hensyn til kompositionen, det vil sige, at komposition af to elementer skal resultere i et element i gruppen. Når vi adderer to punkter på kurven $E(\mathbb{F}_p)$, skal summen således være et punkt på $E(\mathbb{F}_p)$. Når vi skal definere addition, viser det sig nemmest at visualisere elliptiske kurver i det reelle $\mathbb{R} \times \mathbb{R}$-talplan og definere kompositionen geometrisk.

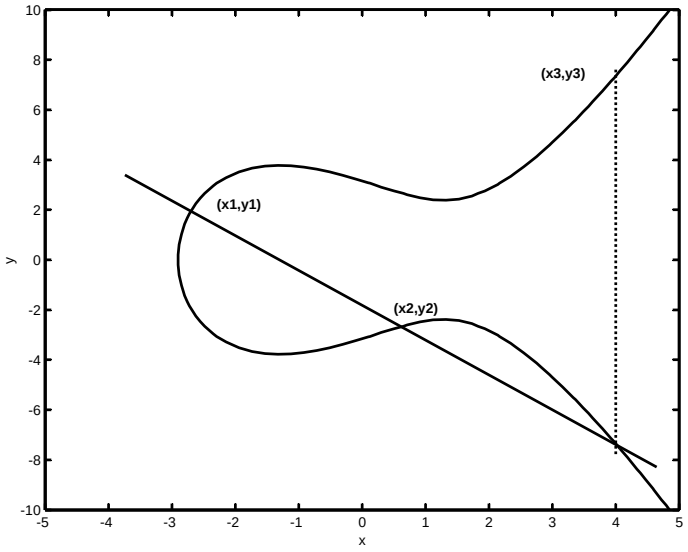
[2]På engelsk "the point at infinity"

Dette gøres i det følgende, hvorefter vi formaliserer kompositionen i gruppen.

Metoden til addition af punkter på elliptiske kurver betegnes normalt *korde og tangent-reglen*. Først gennemgår vi det generelle tilfælde, for derefter at fokusere på to specialtilfælde.

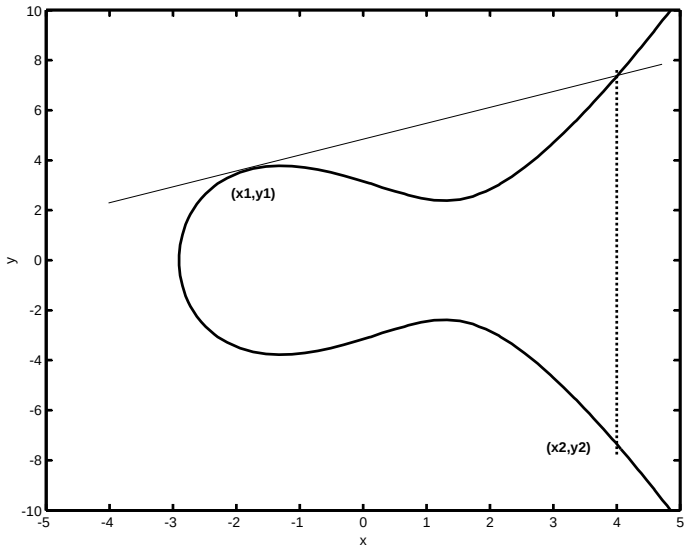
Vi har to punkter, $P = (x_1, y_1)$ og $Q = (x_2, y_2)$ og betegner summen $R = P + Q = (x_3, y_3)$. Vi tegner korden mellem P og Q på kurven, se figur 5.1. Denne korde skærer kurven i et tredje punkt, og her igennem tegner vi en linie parallel med y-aksen. Denne nye linie skærer kurven i et fjerde punkt, som netop er R.

Vi går nu videre til de specialtilfælde hvor ovenstående fremgangsmåde ikke



Figur 5.1: Addition af to punkter

er brugbar. Det første tilfælde er $fordobling$ af et punkt, altså $R = 2P = P + P$. Da vi ikke kan bruge førnævnte metode, tegner vi i stedet punktets tangent, se figur 5.2. Denne tangent skærer kurven i et punkt, og her igennem tegner vi en linie parallelt med y-aksen. Denne nye linie skærer kurven i endnu et punkt, som netop er R.



Figur 5.2: Fordobling af et punkt

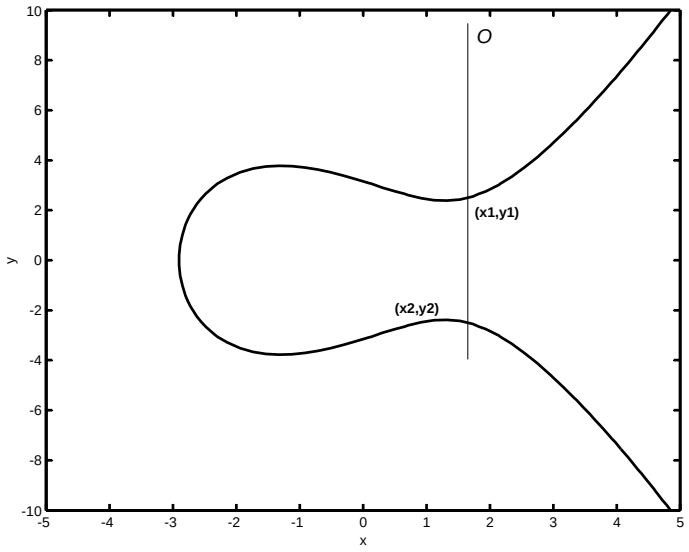
Et andet specialtilfælde er addition af *spejlingspunkter*, det vil sige to punkter, hvor det ene kan spejles i x-aksen over i det andet punkt[3]. Punkterne har altså ens x-koordinater, og y-koordinaterne har modsat fortegn, altså $R = P + Q = (x_1, y_1) + (x_2, y_2)$, hvor $x_1 = x_2$ og $y_1 = -y_2$. Dette er illustreret i figur 5.3. Igen tegner vi en korde mellem de to punkter, men da denne korde ikke skærer kurven andre steder, definerer vi summen som *uendelighedspunktet* $\mathcal{O}$. Når $P + Q = \mathcal{O}$ skrives punktet $Q = (x_2, y_2)$ som $-P$.

Vi bevæger os nu fra det reelle talrum til legemet $\mathbb{F}_p$ og formaliserer de ovenfor beskrevne geometriske metoder.

Vi har to punkter $P = (x_1, y_1) \in E(\mathbb{F}_p)$ og $Q = (x_2, y_2) \in E(\mathbb{F}_p)$, hvor $P \neq \pm Q$. Summen af de to punkter $R = P + Q = (x_3, y_3)$ defineres som:

$$\begin{aligned} x_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \\ y_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1 \end{aligned} \tag{5.7}$$

[3]Såfremt ligning 5.6 defineres mere generelt, ligger kurven ikke symmetrisk om x-aksen. I dette tilfælde er spejlingspunkter en misvisende betegnelse. Vi ser her kun på kurver defineret i ligning 5.6



Figur 5.3: Addition med spejlingspunkt

Ved division i $\mathbb{F}_p$ forstås multiplikation med den inverse modulus p. Den inverse til et tal $a \in \mathbb{F}_p$ er det heltal b, der multipliceret med a giver neutralelementet. Addition, subtraktion og multiplikation foregår ligeledes i $\mathbb{F}_p$.

Ved fordobling af et punkt $P = (x_1, y_1) \in E(\mathbb{F}_p)$, hvor $P \neq -P$, defineres $2P = P + P = (x_3, y_3)$ som

$$\begin{aligned} x_3 &= \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \\ y_3 &= \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1 \end{aligned} \tag{5.8}$$

Er $P = -P$ er vi i det sidste specialtilfælde. Vi ser nu på addition af spejlingspunkter, hvor $R = P+Q = (x_1, y_1)+(x_2, y_2)$, hvor $x_1 = x_2$ og $y_1 = -y_2$, altså $P = -Q$. Dette punkt R defineres som $\mathcal{O}$.

Vi ser nu på gruppestrukturen og vil konkludere, at $E(\mathbb{F}_p)$ sammen med additionskompositionen udgør en gruppe. Da skal kravene nævnt først i afsnit 5.1.1 være opfyldte.

Mængden $E(\mathbb{F}_p)$ er lukket mht addition, hvilket kan anskueliggøres udfra figur 5.1.

Det neutrale element er $\mathcal{O}$, da det for alle $P \in E(\mathbb{F}_p)$ gælder, at $P + \mathcal{O} = \mathcal{O} + P = P$.

Det inverse element til punktet $P \in E(\mathbb{F}_p)$ er $-P$, da det for alle $P \in E(\mathbb{F}_p)$ gælder, at $P + (-P) = \mathcal{O}$.

Endelig er additionskompositionen associativ. Dette vil vi undlade at vise. Det formelle bevis er omfattende, og falder udenfor denne rapports omfang, og selvom det kan vises geometrisk, overlades beviset til læseren.

Gruppen er desuden kommutativ, hvilket nemt ses geometrisk.

Hermed kan vi konkludere, at $(E(\mathbb{F}_p), \circ)$ er en gruppe.

Vi har nu fået vores gruppeteoretiske grundlag på plads. For bedre at forstå ECDSA, ser vi på to systemer, som er gået forud for denne standard. Disse omfatter den udbredte signeringsalgoritme ElGamal, og den amerikanske standard for digitale signaturer, Digital Signature Algorithm (DSA). ECDSA er groft sagt en variant af DSA, som benytter elliptiske kurver.

5.2 Det Diskrete Logaritme Problem

Sikkerheden i digitale signaturer bygger på et bagvedliggende matematisk problem, som er svært at løse. Generelt betegnes et matematisk problem i denne sammenhæng som *svært*, hvis der ikke findes en effektiv metode til at løse problemet, dvs. en algoritme, der kan løse problemet i polynomiel tid[4].

Vi indleder med at introducere det matematiske problem, der ligger til grund for sikkerheden i de digitale signaturer vi vil gennemgå.

Sikkerheden i ECDSA bygger på det *diskrete logaritme problem (DLP)* som genfindes i ElGamal signaturer og i DSA.

Vi har en gruppe $(G, \circ)$, $\alpha \in G$ og $\beta \in H$, hvor H er undergruppen af G, genereret af $\alpha \geq 0$. Det diskrete logaritme problem defineres generelt på følgende vis. Find et a, således at

$$\alpha^a = \beta \tag{5.9}$$

[4]For en længere diskussion af køretider og algoritmer til at bryde talteoretiske problemer se [1], kap. 33, side 802pp

hvor $0 \leq a \leq |H| - 1$. $|H|$ er ordenen af undergruppen H, og α^a er

$$\alpha^a = \underbrace{\alpha \circ \alpha \circ \cdots \circ \alpha}_{i\ alt\ a\ gange} \tag{5.10}$$

I det følgende gennemgår vi ElGamal, DSA og sidst ECDSA, med særligt fokus på, hvordan sikkerheden i alle tre systemer bygger på DLP. For hver algoritme gennemgår vi først de nødvendige *domæneparametre*. Domæneparametre er offentlige, og kendes af alle parter. Dernæst gennemgår vi matematikken og herunder sikkerheden i systemet.

5.3 ElGamal

5.3.1 Domæneparametre

Vi vælger et primtal p og et primitivt element $\alpha \in Z_p^*$. Domæneparametrene er (α, p). Vi regner i gruppen $(\mathbb{Z}_p^*, \cdot)$.

5.3.2 ElGamal i praksis

Nøgler

Vi vælger en privat nøgle $a, 0 \leq a < p - 1$ og udregner den offentlige nøgle β således

$$\beta \equiv \alpha^a \ (mod\, p) \tag{5.11}$$

Signering

Vi ønsker at signere en besked m ved hjælp af den private nøgle a. Vi signerer ikke hele beskeden, men kun en *hash-værdi* e af m. Vi anvender *SHA-1* hash-algoritmen til at udregne e. Dette er en standardalgoritme [10], der kendes af alle parter. Alle parter kan dermed udregne e fra m. Vi vil herefter kun behandle signering af hash-værdier e og abstrahere fra det faktum, at der bagved ligger en besked.

I ElGamal-systemet kan den samme besked have flere forskellige signaturer, da vi hver gang skal vælge et tilfældigt heltal k, som bruges i algoritmen. For et k, hvor $0 < k < p - 1$ og $gcd(k, p - 1) = 1$, definerer vi signaturen af beskeden m som

$$sig(e, k) = (\gamma, \delta) \tag{5.12}$$

hvor

$$\gamma = \alpha^k \ (mod\, p) \tag{5.13}$$

og

$$\delta = (e - a\gamma)k^{-1}\, mod\, (p-1) \tag{5.14}$$

Verificering

Denne signatur kan efterfølgende verificeres af modparten, med kendskab alene til den offentlige nøgle. Vi kan verificere ved at udregne

$$v = \beta^\gamma \gamma^\delta\, mod\, p \tag{5.15}$$

Dette skal modsvare den modtagede værdi, som er $r = \alpha^e\, mod\, p$. Dette er en valid måde at verificere på. Det ses af følgende

$$\begin{aligned} v &= \beta^\gamma \gamma^\delta\, mod\, p \\ &= \alpha^{a\gamma}\alpha^{k\delta} \\ &= \alpha^{a\gamma + k\delta} \\ &= \alpha^{(a\gamma + k\delta)\, mod\, p-1} \\ &= \alpha^{e\, mod\, p-1} \\ &= \alpha^e,\ mod\, p \\ &= r \end{aligned} \tag{5.16}$$

5.3.3 ElGamal og DLP

Der er mange måder at angribe ElGamal-systemet på. Vi ser nærmere på nogle enkelte af disse, for at vise relationen til det underliggende matematiske problem, DLP. Der findes andre typer af angreb, hvor der f.eks. udnyttes protokol-fejl, men disse er ikke behandlede i denne rapport.

En angriber, Eve, kan forsøge at bryde systemet ved at bestemme den private nøgle a. Hun har kendskab til den offentlige nøgle β og domæneparametrene (α, p) og bruger derfor ligning 5.11. Denne ligning er en variant af ligning 5.9, og Eve skal således løse det diskrete logaritme problem for at bestemme a.

Eve kan i stedet forsøge at signere en besked uden at kende a. Hun skal derfor bestemme signaturen (γ, δ), hvor γ udregnes jf. ligning 5.13. Tilbage

står at bestemme et δ, så beskeden verificeres jf. ligning 5.16. Eve skal altså for δ løse følgende

$$\begin{aligned} \beta^{\gamma}\gamma^{\delta} &= \alpha^{e} \, mod \, p \quad \Leftrightarrow \\ \gamma^{\delta} &= \beta^{-\gamma}\alpha^{e} \, mod \, p \quad \Leftrightarrow \\ \delta &= log_{\gamma}(\alpha^{e}\beta^{-\gamma}) \end{aligned}$$

Dette er en variant af ligning 5.9, og igen skal Eve løse det diskrete logaritme problem.

Eve kan i stedet starte med at vælge en δ-værdi og forsøge at udregne γ jf. ligningen 5.16. Hun skal da for γ løse

$$\beta^{\gamma}\gamma^{\delta} = \alpha^{e} \, mod \, p$$

Denne ligning findes der ingen kendte løsningsmetoder til, og den antages derfor at være mindst lige så svær at løse som DLP.

Vi har herover gennemgået de mest oplagte angreb på ElGamal, og vi har set, at et vellykket angreb mindst indebærer at løse DLP.

5.4 Digital Signature Algorithm

DSA er som navnet antyder en amerikansk standardalgoritme til signering[5]. DSA er en praktisk orienteret standard, og derfor indeholder beskrivelsen herunder elementer, som ikke har direkte matematisk interesse. Elementerne er derimod rettede mod graden af sikkerhed i systemet og praktisk implementering af algoritmen. Eksempler på dette er specifik størrelsesbestemmelse af de anvendte parametre og løbende validering af værdierne i algoritmen.

5.4.1 Domæneparametre

Vi vælger et primtal q af størrelsesordenen 2^{160} og et primtal p af størrelsesordenen 2^{1024}. Der skal gælde at $q|p-1$.

Vi vælger et primitivt element $h \in \mathbb{Z}_p^*$ og udregner $g = h^{(p-1)/q} mod p$. Såfremt $g \neq 1$ er g et element af orden $q \in \mathbb{Z}_p^*$. Domæneparametrene er (p, q, g). Vi regner i gruppen $(\mathbb{Z}_p^*, \cdot)$.

[5]For en nærmere beskrivelse af hvordan de forskellige standarder er sammenkædet, se [10] og [3], kap. 15. Dette afsnit er baseret på fortolkningen af DSA standarden givet i [6]

5.4.2 DSA i praksis

Nøgler

Vi vælger en privat nøgle x, således at $1 \leq x \leq q-1$. Vi udregner den offentlige nøgle y således

$$y = g^x \, mod \, p \tag{5.17}$$

Signering

Vi ønsker at signere hash-værdien e af beskeden m. Vi vælger et tilfældigt heltal k, således at $1 \leq k \leq q-1$. Vi definerer signaturen af beskeden m som følger

$$sig(e,k) = (r,s) \tag{5.18}$$

Vi udregner $X = g^k \, mod \, p$, og får

$$r = X \, mod \, q \tag{5.19}$$

Hvis $r = 0$ vælger vi et nyt k. Vi udregner endvidere $k^{-1} \, (mod \, q)$ og endeligt

$$s = k^{-1}(e + xr) \, mod \, q \tag{5.20}$$

Hvis $s = 0$ vælger vi et nyt k.

Verificering

Vi kan nu verificere signaturen. Vi kontrollerer først, at r og s er valide, dvs. at de begge ligger i intervallet $[1, q-1]$.

Vi udregner $w = s^{-1} \, mod \, q$, $u_1 = ew \, mod \, q$ og $u_2 = rw \, mod \, q$. Endelig udregner vi $\widetilde{X} = g^{u_1} y^{u_2} \, mod \, p$ og til sidst

$$v = \widetilde{X} \, mod \, q \tag{5.21}$$

Vi verificerer signaturen, hvis

$$v = r \tag{5.22}$$

Vi ser nu nærmere på, hvorfor verificeringen virker. Vi ved, at

$$\begin{aligned} \widetilde{X} &= g^{u_1} y^{u_2} \, mod \, p \\ &= g^{u_1} g^{x u_2} \, mod \, p \\ &= g^{u_1 + x u_2} \, mod \, p \end{aligned} \tag{5.23}$$

Eksponenten af g kan omskrives til

$$\begin{aligned}(u_1 + xu_2)\ mod\ q &= (es^{-1} + xrs^{-1})\ mod\ q \\ &= (e + xr)s^{-1}\ mod\ q \\ &= (e + xr)k \cdot (e + xr)^{-1}\ mod\ q\end{aligned}$$

da $s = k^{-1}(e + xr)\ mod\ q$ jf. ligning 5.20.

Vi har, at

$$\begin{aligned}(u_1 + xu_2)\ mod\ q &= (e + xr)k \cdot (e + xr)^{-1}\ mod\ q \\ &= k\ mod\ q\end{aligned}$$

Vi vender tilbage til ligning 5.23 og ser, at

$$\begin{aligned}\widetilde{X} &= g^{u_1 + xu_2}\ mod\ p \\ &= g^{k\ mod\ q}\ mod\ p \\ &= g^k\ mod\ p \\ &= X\end{aligned} \tag{5.24}$$

da $ord(g) = q$. Dette giver altså, at

$$\begin{aligned}v &= \widetilde{X}\ mod\ q \\ &= X\ mod\ q \\ &= r\end{aligned} \tag{5.25}$$

5.4.3 DSA og DLP

DSA systemet kan angribes med samme fremgangsmåde som beskrevet i afsnittet om ElGamal systemet. Eksempelvis udregnes den offentlige nøgle y i DSA som

$$y = g^x\ mod\ p \tag{5.26}$$

Udfra denne ligning kan den private nøgle x udregnes, men dette kræver som i ElGamal kendskab til en effektiv algoritme til at løse DLP.

5.5 Elliptic Curve Digital Signature Algorithm

Dette afsnit indeholder ikke en udtømmende gennemgang af ECDSA. Vi har valgt at udelade enkelte detaljer, da vi har vurderet, at de ikke har

relevans for denne rapport. Dette er enkelte domæneparametre, der ikke har relevans i gennemgangen af matematikken, men som er rettede specifikt mod en implementering. For en detaljeret gennemgang af ECDSA henviser vi til Menezes' beskrivelse af standarden [6].

5.5.1 Domæneparametre

Som tilfældet var i DSA, så anvender ECDSA også nogle domæneparametre. I dette tilfælde består domæneparametrene af parametrene til en elliptisk kurve E defineret over et endeligt legeme $\mathbb{F}_p$ samt et basispunkt $G \in E(\mathbb{F}_p)$.

Vi vælger et endeligt legeme $\mathbb{F}_p$, så de mest almindelige algoritmer til løsning af DLP ikke er anvendelige. Antallet af punkter $\sharp E(\mathbb{F}_p)$ på kurven skal være deleligt med et primtal $n > 2^{160}$. Endvidere kræves det, at p er større end 3.

Vi vælger $a, b \in \mathbb{F}_p$, der bestemmer kurven jf. ligning 5.6.

Til sidst vælger vi et basispunkt $G = (x_G, y_G)$, hvor $x_G, y_G \in \mathbb{F}_p$, således at ordenen af G er n.

Domæneparametrene benævnes i rækkefølgen ovenfor som (p, a, b, G, n)

5.5.2 ECDSA i praksis

Til forskel fra afsnittet om ElGamal og DSA, regner vi her med punkter på en elliptisk kurve.

Nøgler

Vi vælger en privat nøgle d, således at $1 \leq d \leq n-1$, og udregner

$$Q = dG \tag{5.27}$$

Den offentlige nøgle er punktet Q.

Signering

Vi ønsker at signere hash-værdien e af beskeden m. Vi vælger et tilfældigt heltal k, så $1 \leq k \leq n-1$. Vi definerer signaturen af beskeden m som

$$sig(e, k) = (r, s) \tag{5.28}$$

Vi udregner $kG = (x_1, y_1)$ og udregner

$$r = x_1 \; mod \; n \tag{5.29}$$

Hvis $r = 0$ vælger vi et nyt k og starter forfra med signeringen. Vi udregner endvidere $k^{-1} \; (mod \; n)$ og får

$$s = k^{-1}(e + dr) \; mod \; n \tag{5.30}$$

Hvis $s = 0$ vælger vi et nyt k og starter forfra med signeringen.

Verificering

Vi kontrollerer først, at r og s er valide, dvs. at de ligger i intervallet $[1, n-1]$.

Vi udregner $w = s^{-1} \; mod \; n$, $u_1 = ew \; mod \; n$ og $u_2 = rw \; mod \; n$.

Herefter udregner vi $\widetilde{X} = (x_1, y_1) = u_1 G + u_2 Q$. Hvis $\widetilde{X} = \mathcal{O}$ afviser vi signaturen, ellers udregner vi

$$v = x_1 \; mod \; n \tag{5.31}$$

Vi accepterer signaturen, hvis

$$v = r \tag{5.32}$$

Vi ser nu nærmere på, hvorfor verificeringen virker. Vi ved, at

$$\begin{aligned} s &\equiv k^{-1}(e + dr) \; mod \; n \Leftrightarrow \\ ks &\equiv (e + dr) \; mod \; n \Leftrightarrow \\ k &\equiv s^{-1}(e + dr) \; mod \; n \\ &\equiv s^{-1}e + s^{-1}dr \; mod \; n \\ &\equiv we + wdr \; mod \; n \\ &\equiv u_1 + u_2 d \mod n \end{aligned} \tag{5.33}$$

Dette giver os umiddelbart, at

$$\begin{aligned} k &= u_1 + u_2 d \Leftrightarrow \\ kG &= u_1 G + u_2 dG \\ &= u_1 G + u_2 Q \\ &= X \end{aligned} \tag{5.34}$$

Kombinerer vi ligning 5.29 med ligning 5.31 ser vi, at

$$v = r \tag{5.35}$$

5.5.3 ECDSA og DLP

I ECDSA har vi en endelig gruppe $(E(\mathbb{F}_p), +)$. Vi laver vi en offentlig nøgle ved at multiplicere basispunktet G med d, dvs. vi adderer G med sig selv d gange. Hvis vi anvender denne komposition på netop elementet G d gange, får vi et udtryk som dette

$$Q = dG \tag{5.36}$$

hvor dG er

$$dG = \underbrace{G \circ G \circ \cdots \circ G}_{i\ alt\ d\ gange} \tag{5.37}$$

Jf. ligning 5.9 er dette en variant af DLP, så sikkerheden i ECDSA bygger på det diskrete logaritme problem. Endvidere viser det sig, at DLP i $(E(\mathbb{F}_p), +)$ er sværere at løse end DLP i $(\mathbb{Z}_p^*, \cdot)$. En sammenligning af nøglestørrelser for ækvivalent sikkerhed ses af nedenstående tabel, der er taget fra [9]

DLP i $(E(\mathbb{F}_p), +)$	DLP i $(\mathbb{Z}_p^*, \cdot)$
2^{106}	2^{512}
2^{132}	2^{768}
2^{160}	2^{1024}
2^{210}	2^{2048}
2^{600}	2^{21000}

Tabellen beskriver de nøglestørrelser, der kræves for at opnå ækvivalent sikkerhed imellem ECDSA og DSA.

Der bliver til stadighed udviklet mere effektive metoder, til at løse det diskrete logaritme problem. Den i øjeblikket mest effektive algoritme til løsning af DLP, *Number Field Sieve (NFS)*, kan ikke anvendes på den elliptiske kurve variant af DLP, og man er derfor nødsaget til at bruge nogle mindre effektive algoritmer til at løse DLP i $(E(\mathbb{F}_q), +)$. Derfor vil det være sværere at løse DLP i $(E(\mathbb{F}_q), +)$ end det almindelige DLP i $(Z_p^*, \cdot)$.

Kapitel 6

Vores implementering

Den konkrete implementering vi dokumenterer i dette kapitel, er baseret på de overvejelser, vi har gjort i kapitel 4. Den PKI vi har implementeret, kan ses i figur 4.3, og er således uden LRA, CRL og Time Server. Vi har undladt dette, da implementering af disse enheder ikke bibringer programmet øget demonstrationsværdi.

6.1 Valg af programmeringssprog

Som udgangspunkt vil vi anvende vores PKI på mobile enheder, og vi har valgt at anvende den i forbindelse med Smart Cards.

Programmering imod et Smart Card indebærer ofte, at der skal laves *chip-nær* programmering. Vi har dog ikke noget umiddelbart kendskab til denne programmeringsteknik, så vi har i stedet kigget på et *API*[1], der findes til Smart Cards, et såkaldt *JavaCard API*. Dette API giver mulighed for at programmere en applikation i Java og derefter overføre applikationen, så den kører i Smart Card'et.

JavaCard API'et har desværre vist sig at medføre en række begrænsninger i funktionaliteten i forhold til regulær Java programmering. Java indeholder en klasse til håndtering af store heltal. Denne klasse, *BigInteger*, indeholder en række funktioner til bl.a. potensopløftning og modulus operationer, og implementerer desuden Euklids udvidede algoritme. Desværre er denne klasse ikke understøttet af JavaCard API'et, og vi har derfor valgt ikke at anvende API'et.

[1] API står for Application Programmer's Interface, og definerer en grænseflade mellem den specifikke hardware enhed til et kendt programmeringssprog.

Da vi således ikke har valgt at bruge JavaCard API'et, har Smart Card implementering vist sig at blive meget omstændig. Vi har derfor udeladt det fysiske Smart Card fra vores løsning, men alligevel udvikle i Java. Dette skyldes hovedsageligt, at vores løsning dermed senere nemt kan overføres til at virke med JavaCard-API'et. Dette vil enten kræve, at JavaCard API'et i en senere version understøtter håndtering af store heltal, eller at man manuelt implementerer denne understøttelse. Sidstnævnte kunne vi have valgt at gøre, men vi har vurderet at det vil blive for omstændigt, samt være uden for vores projekts ramme.

Vi kunne have valgt at bruge programmeringssproget C, da det er meget hurtigt til beregningsopgaver. Vi har dog valgt ikke at sætte hastigheden af vores applikation som primært kriterie, da vores applikation som udgangspunkt skal være en demonstration af vores PKI og den bagvedliggende matematik. Ydermere giver Java os en lang række indbyggede funktioner, heriblandt BigInteger-klassen, netværks-funktionalitet og SHA-1 hash-funktionen, som vi derfor ikke behøver at beskæftige os yderligere med implementering af. Var valget af programmeringssprog faldet på C, skulle vi have udviklet eller tilrettet disse funktioner selv.

Slutteligt er Java et objektorienteret programmeringssprog, hvilket vi har benyttet meget. Vi betragter programmeringssproget som et arbejdsredskab til at lave førnævnte demonstration, og vi har derfor valgt Java, som vi kendte i forvejen og samtidigt opfyldte vores betingelser. Det er muligt, at vi havde valgt anderledes, hvis vores løsning skulle anvendes i et reelt produktionsmiljø.

6.2 Tekniske perler

Vi har foretaget mange valg under udvikling af applikationen. I dette afsnit vil vi redegøre for nogle af disse valg, og fremhæve specielt interessante dele af vores applikation.

6.2.1 Valg af legeme

Ét af de valg, vi har truffet, er valg af det *legeme*, som vi vil definere vores elliptiske kurver over. Man kan vælge at definere kurverne over legemet F_{2^m}, hvor elementerne er binære polynomier af grad mindre end m. Dette legeme

har nogle klare fordele ved implementation, da udregninger med disse polynomier kan klares ved operationer i computeren på bitniveau, hvilket gør det meget hurtigt. Man kan også vælge at definere kurverne over F_p, hvor elementerne er heltal. Dette giver os mulighed for at anvende den førnævnte BigInteger-klasse, og derfor har vi valgt dette legeme.

6.2.2 Valg af kurver

I ECDSA standarden [6] er der defineret en række anbefalede domæneparametre til elliptiske kurver. Konsekvenserne af ikke at anvende disse anbefalede parametre er, at man skal være meget omhyggelig med udregningen af domæneparametrene til kurverne, og en stor del af implementeringen vil derfor bestå af kode, hvis eneste formål er at sikre, at de kurver man laver er valide. Derfor har vi valgt at bruge de parametre, der er inkluderede i standarden.

Som udgangspunkt signerer nøglecentret brugernes offentlige nøgler med den højeste styrke, der er mulig i henhold til standarden, dvs. 521-bit nøglelængde. Som det ses af tabellen i afsnit 5.5.3 i kapitel 5, giver dette en meget høj sikkerhed, men da tilliden til nøglecentret er central giver dette god mening.

Vores brugernøgler har som udgangspunkt størrelsen 192-bit, da det er den mindste nøglelængde, der er tilladt i standarden. Set i forhold til førnævnte tabel, sker signeringen på klientsiden stadig med en meget stor sikkerhed, men det har ikke været muligt for os at gå længere ned. Dette mener vi heller ikke er nødvendigt, da vi i vores applikation ikke fokuserer på hastighed.

De foruddefinerede kurver har endnu en egenskab, der kan komme os til nytte. Det viser sig, at p's binære repræsentation i alle tilfælde er karakteristisk ved at bestå af lange gentagne sekvenser af 1-taller og 0'er. F.eks. består den binære repræsentation af primtallet i kurven P-192 af 127 1-taller, et enkelt 0 og så endnu 64 1-taller. Dette giver muligheder for optimering af algoritmerne, som vi dog ikke har påtaget os. En samlet oversigt over bitrepræsentationerne kan ses i appendix D.

6.2.3 SHA-1

I standarden beregnes SHA-1 værdien som en integreret del af signering og verificering. Dette er dog ikke optimalt, da dette betyder, at hash-værdien af beskeden skal beregnes i kortet. Er beskeden stor, vil den ikke kunne være i

Smart Card'et, og selvom det er muligt at *streame* beskeden igennem Smart Card'et, er et Smart Card begrænset i båndbredden ud og ind af kortet. Derfor vil det ofte tage for lang tid at udregne hash-værdien.

Vi har valgt at lægge SHA-1 funktionen væk fra vores algoritme. I stedet for at udregne hash-værdien i selve algoritmen, beder algoritmen om en hash-værdi, som så bruges som input i algoritmen. Fordelen ved dette er, at funktionen nu ikke nødvendigvis skal ligge på Smart Card'et, men kan placeres på en kraftig stationær PC, og Smart Card'et kan så bede denne PC om at udregne hash-værdien. Det er så kun de 160 bits, der skal sendes ind i kortet.

6.2.4 Double And Add

Ved udregning af den offentlige nøgle i ligning 5.27 multipliceres basispunktet G med et tal d. Da vi ikke har mulighed for multiplikation i vores gruppe, må vi i stedet addere G med sig selv $d-1$ gange. Da de tal vi arbejder med, er af størrelsen 10^{60}, kan dette praktisk set ikke lade sig gøre[2].

Vi har løst dette problem ved at implementere *double and add* algoritmen, der har logaritmisk køretid. Dette betyder, at vi ikke skal lave 10^{60} additioner for at udregne den offentlige nøgle, men blot $log_2(2^{200}) \approx 200$ additioner.

Vi benytter her et eksempel til at beskrive *Double And Add* algoritmen. For en mere formaliseret gennemgang, henviser vi til gennemgangen af den tilsvarende algoritme *Square And Multiply* i [5], figur 4.4.

Vi ønsker, at vise hvordan vi for et punkt Q udregner $d \cdot Q$, hvor $d = 23$. For at lette beskrivelsen nummererer vi de enkelte bits fra højre mod venstre med $i = 0, \ldots, 5$. Først bestemmer vi den binære repræsentation af d

$$23_{dec} = 2^4 \cdot 1 + 2^3 \cdot 0 + 2^2 \cdot 1 + 2^1 \cdot 1 + 2^0 \cdot 1 = 10111_{bin}$$

Hvis vi multiplicerer med Q får vi

$$\begin{aligned} 23 \cdot Q &= 2^4 \cdot 1 \cdot Q + 2^3 \cdot 0 \cdot Q + 2^2 \cdot 1 \cdot Q + 2^1 \cdot 1 \cdot Q + 2^0 \cdot 1 \cdot Q \\ &= 2^4 \cdot Q + 2^2 \cdot Q + 2^1 \cdot Q + 2^0 \cdot Q \end{aligned}$$

[2]Den mindste af de kurver NIST anbefaler, har bitlængden 200. Tallene bliver da af størrelsesordenen $2^{200} \approx 10^{60}$

For at illustrere hvor enkelt dette kan implementeres, viser vi herunder pseudokode for algoritmen (se evt. [5], figur 4.4)

1. $sum = mathcalO$
2. $R_0 = Q$
3. FOR $i = 0$ TO $l - 1$
4. IF $b_i = 1$ THEN $sum = sum + R_i$
5. $R_{i+1} = R_i + R_i$
6. RETURN sum

hvor l er antallet af bits i d og b_i er den i'te bit i d.

Som udgangspunkt sætter vi summen lig 0. Vi ser, at vi lægger $R_i = 2^i \cdot Q$ til summen, hvis den i'te bit er sat. Vi ser endvidere, at det punkt vi lægger til summen, hvis den i'te bit er sat, er det dobbelte af det punkt $R_{i-1} = 2^{(i-1)}$, vi lægger til, hvis den foregående bit er sat. Dette punkt R_{i-1} har vi netop udregnet i det foregående skridt. Bruger vi denne fremgangsmåde, med udgangspunkt i at definere $R_0 = Q$ kan vi udregne $23 \cdot Q$ med kun $8 - 9$ additioner fremfor 22.

Til sammenligning ser vores implementerede algoritme ser således ud

```
public Point multiply(BigInteger d) {
  Point sum = new Point(true);
  Point R = this;
  for (int i = 0; i < d.bitLength(); i++) {
    if (d.testBit(i))
      sum = sum.add(R);
    R = R.add(R);
  }
  return sum;
}
```

Vi viser nu, hvordan vi udregner $23 \cdot Q$ skridt for skridt. Vi noterer alle variable for hvert skridt i nedenstående tabel. Kolonnebeskrivelsen i tabellen henviser til linienummeret i førnævnte algoritme.

i	b_i	Er b_i sat ?	sum før 4.	R_i før 4.	sum efter 4.	R_{i+1} efter 5.
0	1	ja	O	Q	Q	2Q
1	1	ja	Q	2Q	3Q	4Q
2	1	ja	3Q	4Q	7Q	8Q
3	0	nej	7Q	8Q	7Q	16Q
4	1	ja	7Q	16Q	23Q	32Q

Resultatet er understreget i tabellen herover. Vi ser, at vi når frem til $23Q$ med 9 skridt, hvoraf det ene ($R_4 = 16Q + 16Q$) er overflødigt, da $32 > 23$.

6.3 Gennemgang af vores applikation

Vi har med vores applikation lavet en simulering af en Smart Card løsning. Det vil sige, at vi antager, at Alice møder op ved en LRA eller et nøglecenter og får udleveret et Smart Card med sin private nøgle på.

Vi har valgt at dele vores nøglecenter op i to applikationer; en *CAManager* og en *CAServer*. Serveren står og afventer, at en bruger spørger efter en offentlig nøgle til brug i en verificering af en digital signatur. Manageren er den instans af nøglecentret, der laver certifikater og indlægger de offentlige nøgler i serverens database. Vores nøglecenter er altså en hybrid af en LRA og et nøglecenter. Vi har af tidsmæssige årsager valgt kun at implementere ét nøglecenter.

Vi deler beskrivelsen af vores PKI op i to dele, én der ses fra nøglecentrets synspunkt, og én der ses fra klientens synspunkt.

6.3.1 Nøglecenter

CAManager

Når Alice kontakter Manageren for at få lavet en nøgle, udfylder hun en blanket, der indeholder felter med navn og en beskrivelse af nøgle. Der genereres herefter et nøglesæt til hende, hvorefter hun modtager den private nøgle og den offentlige nøgle lagres i nøglecentret. Både den private og den offentlige nøgle gives det samme unikke nøgle ID, der er en hash-værdi af navn, beskrivelse samt hvilken kurve, der er brugt til generering af nøglerne. Nøgle ID bliver brugt til forespørgsel efter offentlig nøgle i nøglecentret. Den offentlige nøgle signeres af nøglecentret, så hun kan se, at den stammer fra et nøglecenter hun stoler på. Derefter sendes den private nøgle i klartekst til klienten. Dette er umiddelbart ikke optimalt, men vores applikation simulerer netop en Smart Card baseret PKI, og det er tanken at nøglen overføres til Smart Card'et under sikre forhold. Derfor vil det ikke blive noget problem at håndtere denne datastrøm.

Punkt for punkt sker følgende:

1. Alice beder Manageren om at generere et nøglesæt.

2. Alice giver Manageren informationer om sig selv.
3. Manageren laver et nøglesæt baseret på Alice's oplysninger.
4. Manageren downloader den private nøgle til Alice's Smart Card.
5. Manageren udleverer Smart Car'et til Alice.
6. Manageren gemmer den offentlige nøgle i databasen.

CAServer

Serveren afventer konstant, at Bob beder om en offentlig nøgle. Når dette sker, modtager Serveren et unikt nøgle ID, som Serveren efterfølgende slår op i sin database. Inden den relevante offentlige nøgle sendes til Bob, signeres den af CAServeren, så Bob kan sikre sig, at nøglen vitterligt kommer fra Serveren. Serverens offentlige nøgle er kodet direkte ind i Bobs applikation, så verificering af Serverens identitet kan foretages. Hvis Bob beder om en offentlig nøgle, der ikke findes, bliver Bob gjort opmærksom på dette.

Punkt for punkt sker følgende:

1. Serveren afventer forespørgsel.
2. Serveren modtager forespørgsel med specifikt nøgle ID fra Bob.
3. Serveren henter nøglen fra databasen.
4. Serveren signerer nøglen.
5. Serveren sender nøglen til Bob eller giver besked, hvis den ikke findes.

6.3.2 Klient

Klienten består af en applikation, der kan bruges til såvel signering som verificering.

Alice Signerer

Når en klient signerer en besked, påhæftes signaturen den oprindelige besked. Tillige inkluderes det unikke nøgle ID, der identificerer brugeren overfor nøglecentret. Beskeden sendes herefter til den tiltænkte modtager Bob. Der er ikke nogen kommunikation imellem klienten og nøglecentret ved signering af en besked.

Punkt for punkt sker følgende:

1. Alice henter en fil fra lokal disk.
2. Alice angiver placering af privat nøgle eller indsætter Smart Card.
3. Alice signerer filen.
4. Alice sender den signerede fil til Bob.

Bob Verificerer

Når en signeret besked modtages af Bob, udtrækkes det unikke ID fra beskeden, og nøglecentret kontaktes. Bob kan som udgangspunkt ikke se, hvem der har signeret beskeden[3]. Nøglecentret slår derefter op i sin database over brugere, og finder den offentlige nøgle, der passer til det certifikat, der er brugt til underskrivning. Den offentlige nøgle signeres og sendes tilbage til Bob. Ved modtagelse af den offentlig nøgle fra nøglecentret, forsøger Bob at verificere, at nøglecentret har sendt nøglen. Dette er muligt, da alle klienter har nøglecentrets offentlige nøgle liggende. Hvis den offentlige nøgle fra nøglecentret kan verificeres, bruger Bob den offentlige nøgle til at verificere den besked, der oprindeligt blev modtaget. Bob får derefter at vide, om signaturen kan verificeres eller ej.

Punkt for punkt sker følgende:

1. Bob modtager den signerede fil.
2. Bob udtrækker nøgle ID fra den modtagne fil.
3. Bob forespørger Serveren om den offentlige nøgle.
4. Bob modtager Alice's offentlige nøgle, signeret af Serveren
5. Bob verificerer, at nøglen er afsendt fra Serveren
6. Bob verificerer Alice's signatur

[3]Hvis beskeden kommer i en e-mail, vil der ofte være afsender på denne e-mail. Det er i selve de signerede data, der ikke findes informationer om underskriver udover et unikt nøgle ID

6.4 Opsamling

Vi har gjort meget ud af at sikre, at man ikke succesfuldt kan angribe vores PKI ved at angribe protokollen for udveksling af nøgler i stedet for selve algoritmen til ECDSA. Man kan umiddelbart vælge at angribe vores PKI ét sted, hvilket er når Bob beder om en offentlig nøgle fra nøglecentret. Bob beder om en nøgle fra nøglecentret, men i princippet kan Bob ikke vide, at det er hans eget nøglecenter han kommunikerer med. Derfor er det i teorien muligt at udgive sig for at være Alice.

Vi har dog taget højde for dette angreb. Inden afsendelse til Bob bliver den offentlige nøgle signeret af nøglecentret. Dette checker Bob ved modtagelsen, og kan dermed sikre sig, at der ikke er blevet ændret i nøglen undervejs. Implementeringen af vores PKI er dermed ikke mulig at angribe fra denne vinkel.

Selvom vi ikke har implementeret alle elementer af en PKI, som beskrevet i kapitel 4, er vi dog kommet omkring samtlige matematiske elementer beskrevet i kapitel 5. Programmet på dog stadig anses for at være en eksperimentiel applikation, der ikke uden videre bør anvendes i et produktionsmiljø.

Kildekoden til vores applikation er gengivet i sin helhed i appendix C og en oversigt over klasserne forefindes i appendix B. Alle Java-filer kan hentes fra rapportens hjemmeside, hvor der også forefindes brugermanual til programmet. Det anbefales at studere denne grundigt, før der eksperimenteres med programmet.

Kapitel 7

Konklusion

Vi har i denne rapport nået de fleste af de mål vi satte os. Vi har belyst de samfundsmæssige perspektiver af digitale signaturer, set på hvordan de implementeres i praksis, og implementeret en PKI, der benytter elliptiske kurver.

Det har vist sig, at digitale signaturer baseret på elliptiske kurver ikke er *raketvidenskab*, men faktisk lader sig implementere i et alment højniveau programmeringssprog. Det eneste der kræves, er understøttelse af store heltal. Opbygning af en PKI har også vist sig at være mulig. Med metoder til signering og verificering på plads, er der kun tilbage omhyggeligt at opbygge strukturen fra en ende af. Vi vurderer, at implementering af de enheder, vi har udeladt, som CRL og Time Server, ikke vil medføre yderligere programmeringsmæssig kompleksitet, men blot vil genbruge de funktioner, der på nuværende tidspunkt findes i vores applikation. Det bliver givetvis mere omstændigt i større skala, men vi har implementeret de nødvendige matematiske algoritmer i vores projekt.

Et mål vi dog ikke har nået, er implementeringen af en Smart Card løsning, der reelt anvender Smart Cards. Vi har anskaffet en Smart Card læser, men vi må nok erkende, at denne del af projektet, ikke er lykkedes os. Vi vurderede undervejs, at arbejdet med at få løsningen til at virke langt ville overstige denne rapports omfang.

Videre projekter

Emnet for denne rapport er enormt. Hver gang vi har dykket ned i en artikel omhandlende digitale signaturer, eller hver gang vi har gået tæt på en egen-

skab ved disse, har det vist sig, at der til umiddelbart enkle problemstillinger ligger mange interessante vinkler. Der er mange oplagte muligheder for at videreføre vores projekt.

Den mest oplagte er at indbygge understøttelse for Smart Cards. En opgave i den forbindelse, er selve programmeringen af en understøttelse af store heltal på Smart Cards.

Vores PKI bruger kun NIST anbefalede standard kurver. Det kunne være interessant, at se på hvordan man kan generere andre kurver. Man kunne også analysere NIST kurverne mere grundigt, da de, selvom de er tilfældige af natur, tilsyneladende er valgt med effektiv implementering for øje.

Det kunne yderligere være interessant at se på optimering af vores applikation, eksempelvis ved brug af et mere hardware-nært programmeringssprog som *C*. Da vil det være oplagt at anvende det matematiske legeme $\mathbb{F}_{2^m}$ og dermed operere med binære polynomier.

Perspektivering

Som beskrevet i specielt kapitel 3 er de første spæde skridt taget i retning af bred anvendelse af digitale signaturer. Vi har fået en lovgivning på området, og nogle af Danmarks største og mest velrenommérede IT-virksomheder har tilkendegivet, at de vil deltage i opbygningen af den nødvendige infrastruktur. Der er dog endnu to uafklarede spørgsmål, der ikke er blevet besvaret. Vil det i *praksis* blive juridisk gyldigt at underskrive et dokument digitalt, og vil danskerne tage den nye teknologi til sig ?

Det første spørgsmål er svært at besvare, da der i lovforslaget ikke er beskrevet en ligestilling imellem digitale signaturer og den håndskrevne signatur. Der er endvidere nogle uafklarede problemer i selve tilblivelsen af den digitale signatur.

Dette leder os til det andet spørgsmål, nemlig om danskerne trygt vil anvende de digitale signaturer. Danskerne har endnu ikke fået et afslappet forhold til at handle på Internettet. Mange er stadig utrygge ved at anvende deres Dankort på et website, selvom der teknisk set ikke er nogen sikkerhedsmæssig forskel på at anvende et Dankort og at anvende home-banking services, som en meget stor del af danskerne flittigt bruger. Det er derfor et godt spørgsmål, om de vil forlade sig på en digital signatur, når de skal underskrive et skøde, købe ny bil eller tegne ny forsikring.

Teknologien er moden til at stå sin prøve på markedet, men der ligger imidlertid en stor opgave i at informere og skabe tillid hos befolkningen til teknologien. Dette ser vi som den største hindring for udbredt brug af digitale signaturer.

Litteratur

[1] Thomas H. Cormen et al., *Introduction to Algorithms*, 18. oplag, MIT Press, 1997.

[2] Donald E. Knuth, *The Art of Computer Programming - Fundamental Algorithms, volume 1*, 2nd edition, Addison-Wesley, Reading, Massachusetts, 1973.

[3] Alfred Menezes et al., *Handbook of Applied Cryptography*, CRC Press, 1996, `http://cacr.math.uwaterloo.ca/hac`

[4] Bruce Schneier, *Applied Cryptography*, 2nd edition, John Wiley & Sons, 1995.

[5] Douglas R. Stinson, *Cryptography - Theory and Practice*, CRC Press, London, 1995, `http://cacr.math.uwaterloo.ca/~dstinson/#books`

Standarder, artikler og vejledninger

[6] Alfred Menezes, Don Johnson, University of Waterloo, *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 24. februar 2000, `http://cacr.math.uwaterloo.ca/~ajmeneze`

[7] Forskningsministeriet, *Praktisk brug af kryptering og digitale signaturer*, 24. maj 2000, `http://www.fsk.dk/cgi-bin/doc-show.cgi?doc_id=33466`

[8] Certicom, *Certicom Online Tutorial, 5.2 The Elliptic Curve Discrete Logarithm Problem*, `http://www.certicom.com/research/ec52.html`

[9] Certicom, *Certicom Whitepapers, The Elliptic Curve Cryptosystem for Smart Cards*, `http://www.certicom.com/research/wecc4.html#inte`

[10] National Institute of Standards and Technology (NIST), *Digital Signature Standard (DSS)*, `http://csrc.nist.gov/cryptval/dss.htm`

[11] National Institute of Standards and Technology (NIST), *FIPS PUB 186-2*, http://csrc.nist.gov/fips/fips186-2.pdf

[12] Retsinfo, *L229 (som vedtaget) Lov om elektroniske signaturer, LOV nr 417 af 31/05/2000*, http://www.retsinfo.dk/_GETDOCI_/ACCN/A20000041730-REGL

[13] Retsinfo, *L63 (som vedtaget) Lov om visse betalingsmidler, LOV nr 414 af 31/05/2000*, http://www.retsinfo.dk/_GETDOCI_/ACCN/A20000041430-REGL

[14] IT-Sikkerhedsrådet, *Udkast til IT-Sikkerhedsrådets vejledning om det offentliges brug af digitale signaturer og kryptering*, http://www.fsk.dk/cgi-bin/doc-show.cgi?doc_id=20806

[15] KMD, *Pressemedelelse d. 29. november 2000 om digitale signaturer*, http://www.kmd.dk/i2.asp?id=1110771&as=1014

Bilag A

Lov om elektroniske signaturer

[1]

Vedtaget af Folketinget ved 3. behandling den 18. maj 2000
Forslag
til
Lov om elektroniske signaturer

Kapitel 1
Formål og anvendelsesområde

§ 1. Lovens formål er at fremme en sikker og effektiv anvendelse af elektronisk kommunikation gennem fastsættelse af krav til visse elektroniske signaturer og til nøglecentre, der udsteder certifikater til elektroniske signaturer.

§ 2. Loven finder anvendelse på nøglecentre etableret i Danmark, der udsteder kvalificerede certifikater til offentligheden, jf. dog § 12.

Stk. 2. Loven finder desuden anvendelse på efterprøvelse af, at signaturgenereringssystemer overholder de opstillede krav til sikre signaturgenereringssystemer.

Kapitel 2
Definitioner

[1] Loven indeholder bestemmelser, der gennemfører Europa-Parlamentets og Rådets direktiv 1999/93/EF af 13. december 1999 om en fællesskabsramme for elektroniske signaturer (EF-Tidende 2000 nr. L 13, s. 12).

§ 3. I denne lov forstås ved:

1. Elektronisk signatur: Data i elektronisk form, der knyttes til andre elektroniske data ved hjælp af et signaturgenereringssystem, og som anvendes til at kontrollere, at disse data stammer fra den person, der er angivet som underskriver, og at de ikke er blevet ændret.

2. Avanceret elektronisk signatur: En elektronisk signatur, der

   a) entydigt er knyttet til underskriveren,

   b) gør det muligt at identificere underskriveren,

   c) skabes med midler, som kun underskriveren har kontrol over, og som

   d) er knyttet til de data, den vedrører på en sådan måde, at enhver efterfølgende ændring af disse data kan opdages.

3. Underskriver: En fysisk person, der besidder et signaturgenereringssystem og handler på egne vegne eller på vegne af en anden fysisk eller juridisk person.

4. Signaturgenereringsdata: Unikke data, som for eksempel en kode eller en privat krypteringsnøgle, som anvendes til at fremstille en elektronisk signatur.

5. Signaturgenereringssystem: Et software- eller hardwarebaseret system til behandling og opbevaring af signaturgenereringsdata.

6. Signaturverificeringsdata: Unikke data, som for eksempel en kode eller en offentlig krypteringsnøgle, som anvendes til at verificere en elektronisk signatur.

7. Signaturverificeringssystem: Et software- eller hardwarebaseret system til behandling af signaturverificeringsdata.

8. Certifikat: En elektronisk attest, som knytter bestemte signaturverificeringsdata til underskriveren og bekræfter dennes identitet.

9. Nøglecenter: En fysisk eller juridisk person, der udsteder certifikater.

Kapitel 3
Kvalificerede certifikater

§ 4. Betegnelsen kvalificerede certifikater eller betegnelser, der er egnede til at fremkalde det indtryk, at der er tale om kvalificerede certifikater, må kun anvendes om certifikater, der opfylder de i stk. 2 og 3 nævnte krav, og som udstedes af et nøglecenter, der opfylder bestemmelserne i kapitel 4 samt regler fastsat i medfør heraf.

Stk. 2. Et kvalificeret certifikat skal indeholde:

1. En angivelse af, at certifikatet er udstedt som et kvalificeret certifikat.
2. Nøglecentrets navn og hjemsted.
3. Underskriverens navn eller pseudonym med angivelse af, at der er tale om et pseudonym.
4. Eventuelle yderligere oplysninger om underskriveren, for så vidt det er nødvendigt for anvendelsen af certifikatet, herunder oplysninger, der sikrer en entydig identifikation af underskriveren.
5. Certifikatets gyldighedsperiode.
6. En tydelig angivelse af eventuelle begrænsninger i certifikatets anvendelsesområde (formålsbegrænsninger).
7. En tydelig angivelse af eventuelle begrænsninger med hensyn til de transaktionsbeløb, certifikatet kan anvendes til (beløbsbegrænsninger).
8. Certifikatets identifikationskode.
9. Signaturverificeringsdata, der svarer til de signaturgenereringsdata, som var under underskriverens kontrol på udstedelsestidspunktet.

Stk. 3. Et kvalificeret certifikat skal være underskrevet med nøglecentrets avancerede elektroniske signatur.

Kapitel 4
Krav til nøglecentres virksomhed

§ 5. Et nøglecenter skal træffe de foranstaltninger, som er nødvendige for et sikkert, pålideligt og velfungerende udbud af kvalificerede certifikater. Nøglecentret skal herunder

1. anvende betryggende administrative og ledelsesmæssige procedurer, som overholder anerkendte standarder,

2. beskæftige personale med den fornødne ekspertise, erfaring og kvalifikationer, herunder personale med sagkundskab inden for elektronisk signaturteknologi og indgående kendskab til korrekte sikkerhedsprocedurer i forbindelse hermed,

3. anvende pålidelige systemer og produkter, som er beskyttet imod uautoriserede ændringer, og som sikrer den tekniske og kryptografiske sikkerhed af de processer, som disse systemer og produkter understøtter,

4. træffe foranstaltninger mod eventuelle muligheder for forfalskning af certifikaterne og

5. til stadighed have tilstrækkelige økonomiske ressourcer til at drive virksomhed i overensstemmelse med bestemmelserne i denne lov, herunder til at opfylde erstatningsmæssige forpligtelser i henhold til loven.

Stk. 2. Nøglecentre, der udsteder kvalificerede certifikater, skal vælge en ekstern statsautoriseret revisor til varetagelse af systemrevisionen i nøglecentret. Telestyrelsen kan i særlige tilfælde dispensere fra kravet om, at systemrevisor skal være statsautoriseret revisor.

Stk. 3. Forskningsministeren fastsætter nærmere regler om kravene i stk. 1.

§ 6. Nøglecentre skal fastsætte og anvende betryggende procedurer til at kontrollere identiteten og andre forhold vedrørende underskriveren forud for udstedelsen af certifikatet.

Stk. 2. Oplysninger om procedurerne som nævnt i stk. 1 skal være offentligt tilgængelige.

Stk. 3. Forskningsministeren kan fastsætte nærmere regler om kravene i stk. 1 og 2.

§ 7. Et nøglecenter skal ved udstedelse af et kvalificeret certifikat sikre, at underskriveren på tidspunktet for udstedelsen er i besiddelse af de signaturgenereringsdata, som korresponderer med de signaturverificeringsdata, der er indeholdt i certifikatet.

Stk. 2. Ved udstedelse af kvalificerede certifikater, hvor det er nøglecentret, der leverer signaturgenereringsdata og signaturverificeringsdata, må der

kun anvendes signaturgenereringsdata og signaturverificeringsdata, som hører sammen på en unik måde. Nøglecentret skal sikre signaturgenereringsdataenes fortrolighed under genereringsprocessen.

Stk. 3. Et nøglecenter skal fastlægge procedurer for udstedelse af certifikater, der gør det muligt at fastslå dato og tidspunkt for udstedelsen.

§ 8. Ved indgåelse af en aftale om udstedelse af et kvalificeret certifikat skal nøglecentret skriftligt oplyse underskriveren om:

1. Vilkårene for anvendelsen af certifikatet, herunder eventuelle formåls- eller beløbsbegrænsninger.

2. Eventuelle krav til underskriverens opbevaring og beskyttelse af signaturgenereringsdataene.

3. Underskriverens omkostninger ved erhvervelse og anvendelse af certifikatet og brug af nøglecentrets øvrige tjenester.

4. Hvorvidt nøglecentret er tilknyttet en frivillig akkrediteringsordning.

5. Procedurer for behandling af klager og bilæggelse af tvister.

Stk. 2. Kontraktsvilkårene kan afgives elektronisk, forudsat at det sker i en for modtageren umiddelbart læsbar form.

Stk. 3. De relevante dele af de i stk. 1 nævnte oplysninger skal på anmodning stilles til rådighed for tredjemand, der forlader sig på et kvalificeret certifikat.

Stk. 4. Forskningsministeren kan fastsætte nærmere regler om kravene i stk. 1-3.

§ 9. Nøglecentre skal sørge for en hurtig og sikker katalog- og tilbagekaldelsestjeneste, som giver mulighed for, at det kan undersøges, om et kvalificeret certifikat er spærret, hvilken gyldighedsperiode certifikatet har, og om certifikatet indeholder formåls- eller beløbsbegrænsninger.

Stk. 2. Et nøglecenter skal spærre et certifikat straks efter at have modtaget anmodning fra underskriveren herom, eller hvis forholdene i øvrigt tilsiger dette.

Stk. 3. Oplysninger efter stk. 1 skal være umiddelbart tilgængelige.

Stk. 4. Et kvalificeret certifikat må kun gøres offentligt tilgængeligt, hvis underskriveren har givet samtykke hertil.

Stk. 5. Forskningsministeren kan fastsætte nærmere regler om kravene i stk. 1-3.

§ 10. Et nøglecenter skal registrere og opbevare alle relevante oplysninger om certifikaterne i en rimelig periode, dog mindst seks år.

Stk. 2. Et nøglecenter skal benytte pålidelige systemer til opbevaring af certifikater i verificerbar form.

Stk. 3. Nøglecentre må ikke opbevare eller kopiere de personers signaturgenereringsdata, som nøglecentret gennem udstedelsen af certifikater måtte have fået kendskab til.

Stk. 4. Forskningsministeren kan fastsætte nærmere regler om kravene i stk. 1 og 2.

Kapitel 5
Erstatningsansvar

§ 11. Nøglecentre, der udsteder kvalificerede certifikater til offentligheden, eller som over for offentligheden indestår for sådanne certifikater udstedt af et andet nøglecenter, er ansvarlig for tab hos den, der med rimelighed forlader sig på certifikatet, såfremt tabet skyldes,

1. at oplysningerne angivet i certifikatet ikke var korrekte på tidspunktet for udstedelsen af certifikatet,

2. at certifikatet ikke indeholder alle oplysninger som krævet i henhold til § 4,

3. manglende spærring af certifikatet, jf. § 9, stk. 2,

4. manglende eller fejlagtig information om, at certifikatet er spærret, hvilken udløbsdato certifikatet har, eller om certifikatet indeholder formåls- eller beløbsbegrænsninger, jf. § 9, stk. 1 og 3, eller

5. tilsidesættelse af § 7.

Stk. 2. Et nøglecenter pådrager sig erstatningsansvar efter stk. 1, medmindre nøglecentret kan godtgøre, at nøglecentret ikke har handlet uagtsomt eller

forsætligt.

Stk. 3. Et nøglecenter er ikke ansvarlig for

1. tab opstået som følge af anvendelse af et kvalificeret certifikat uden for de formålsbegrænsninger, som gælder for certifikatet, eller for

2. tab opstået som følge af en overskridelse af de beløbsbegrænsninger, som gælder for certifikatet,

forudsat at de pågældende begrænsninger tydeligt fremgår af certifikatet, jf. § 4, og på forespørgsel oplyses, jf. 9, stk. 1 og 3.

Stk. 4. Stk. 1-3 kan ikke ved forudgående aftale fraviges til skade for skadelidte.

Stk. 5. Stk. 1-3 finder ikke anvendelse, i det omfang tabet dækkes efter lov om visse betalingsmidler.

Kapitel 6
Supplerende krav til behandling af personoplysninger

§ 12. Et nøglecenter må kun indsamle personoplysninger i forbindelse med nøglecentervirksomheden direkte fra den registrerede eller med den registreredes udtrykkelige samtykke og kun i det omfang, det er nødvendigt for udstedelsen eller opretholdelsen af et certifikat.

Stk. 2. Personoplysninger indsamlet i medfør af stk. 1 må ikke behandles eller videregives til andet formål end nævnt i stk. 1 uden den registreredes udtrykkelige samtykke hertil.

Kapitel 7
Elektronisk signatur og formkrav

§ 13. Bestemmelser i lovgivningen, hvorefter elektroniske meddelelser skal være forsynet med signatur, skal anses for opfyldt, hvis meddelelsen er forsynet med en avanceret elektronisk signatur, der er baseret på et kvalificeret certifikat, og som er fremstillet ved brug af et sikkert signaturgenereringssystem. Ved elektroniske meddelelser til og fra en offentlig myndighed gælder dette dog kun, såfremt andet ikke følger af lov eller bestemmelser fastsat i medfør af lov.

Kapitel 8
Sikre signaturgenereringssystemer

§ 14. Ved et sikkert signaturgenereringssystem forstås et signaturgenereringssystem, der ved hjælp af procedurer og tekniske midler sikrer, at signaturgenereringsdata, der anvendes til at skabe en elektronisk signatur,

1. i praksis kun kan fremtræde en gang,
2. med rimelig sikkerhed forbliver hemmelige og ikke kan udledes,
3. er beskyttet mod forfalskning og
4. på pålidelig vis kan beskyttes af underskriveren mod andres uretmæssige brug.

Stk. 2. Et sikkert signaturgenereringssystem må ikke indrettes således, at det ændrer de data, som en elektronisk signatur knyttes til, eller hindrer, at disse data forevises for underskriveren forud for signeringen.

Stk. 3. De i stk. 1 og 2 nævnte krav skal anses for opfyldt, såfremt et signaturgenereringssystem overholder almindeligt anerkendte standarder for sådanne systemer, som Kommissionen har fastsat og offentliggjort i EF-Tidende i overensstemmelse med proceduren i artikel 9 i Europa-Parlamentets og Rådets direktiv 1999/93/EF af 13. december 1999 om en fællesskabsramme for elektroniske signaturer.

§ 15. Forskningsministeren udpeger et eller flere egnede organer eller myndigheder, som kan medvirke til at efterprøve, om signaturgenereringssystemer opfylder kravene til sikre signaturgenereringssystemer, jf. § 14, stk. 1 og 2, og fastsætter nærmere regler om procedurerne for denne efterprøvelse samt om betaling af gebyr for efterprøvelsen.

Stk. 2. Et signaturgenereringssystem, der betegnes som et sikkert signaturgenereringssystem, må først markedsføres eller anvendes til at fremstille avancerede elektroniske signaturer, der er baseret på et kvalificeret certifikat, når det er blevet efterprøvet, jf. stk. 1.

Stk. 3. Med en efterprøvelse efter stk. 1 ligestilles en efterprøvelse af et sikkert signaturgenereringssystem foretaget af et organ eller en myndighed i et andet land inden for Det Europæiske Økonomiske Samarbejde (EØS).

Kapitel 9
Tilsyn

§ 16. Nøglecentre skal senest samtidig med, at udstedelse af kvalificerede certifikater påbegyndes, foretage anmeldelse til Telestyrelsen.

Stk. 2. Anmeldelsen skal indeholde oplysning om

1. nøglecentrets navn og hjemsted,
2. selskabsform, såfremt nøglecentret drives som selskab,
3. nøglecentrets ledelse og systemrevisor.

Stk. 3. Ændringer i forhold, der er anmeldt i henhold til stk. 2, skal anmeldes inden 8 dage efter, at ændringen er sket.

Stk. 4. Telestyrelsen kan fastsætte nærmere regler om, hvilke yderligere oplysninger anmeldelsen skal indeholde.

§ 17. Nøglecentret skal samtidig med anmeldelse efter § 16 indsende en rapport til Telestyrelsen.

Stk. 2. Rapporten skal indeholde

1. en beskrivelse af nøglecentrets virksomhed og systemer,
2. en erklæring fra nøglecentrets ledelse om, hvorvidt nøglecentrets samlede data-, system- og driftssikkerhed må anses for betryggende og i overensstemmelse med denne lovs regler samt regler fastsat i medfør heraf, og
3. en erklæring fra systemrevisor, jf. § 5, stk. 2, om, hvorvidt nøglecentrets samlede data-, system- og driftssikkerhed efter systemrevisors opfattelse må anses for betryggende og i overensstemmelse med denne lovs regler samt regler fastsat i medfør heraf.

Stk. 3. Nøglecentret skal årligt udarbejde en opdateret rapport. Telestyrelsen fastsætter en frist for, hvornår rapporten senest skal indsendes til Telestyrelsen.

Stk. 4. Telestyrelsen kan fastsætte nærmere regler vedrørende indholdet af nøglecentrets rapporter samt om systemrevisionens gennemførelse i nøglecentre.

§ 18. Telestyrelsen påser overholdelsen af denne lov og bestemmelser udstedt i medfør af loven.

Stk. 2. Telestyrelsen kan påbyde et nøglecenter at

1. foretage anmeldelse til Telestyrelsen, jf. § 16,

2. indsende rapporter til Telestyrelsen, jf. § 17,

3. bringe forhold vedrørende nøglecentrets virksomhed i overensstemmelse med loven eller bestemmelser udstedt i medfør af loven.

Stk. 3. Telestyrelsen fastsætter en tidsfrist for opfyldelse af påbud efter stk. 2.

Stk. 4. Telestyrelsen kan pålægge et nøglecenter tvangsbøder med henblik på at gennemtvinge påbud efter stk. 2, § 19, stk. 1, eller § 20.

Stk. 5. Telestyrelsen kan kræve, at der gennemføres en ekstraordinær systemrevision af et nøglecenter. Telestyrelsen udpeger den systemrevisor, som skal udføre den ekstraordinære systemrevision. Nøglecentret kan pålægges at betale for den ekstraordinære systemrevisions udførelse.

Stk. 6. Telestyrelsen kan fratage et nøglecenter retten til at anvende betegnelsen kvalificerede certifikater, jf. § 4, hvis nøglecentret

1. trods pålæg af tvangsbøder undlader at efterkomme Telestyrelsens påbud efter stk. 2, § 19, stk. 1, eller § 20,

2. groft eller i gentagne tilfælde har overtrådt lovens regler eller regler fastsat i medfør heraf eller

3. anmelder betalingsstandsning eller kommer under konkurs.

Stk. 7. Telestyrelsens afgørelse efter stk. 6 kan af nøglecentret forlanges indbragt for domstolene. Anmodning herom skal være modtaget i Telestyrelsen senest 4 uger efter, at afgørelsen er blevet meddelt nøglecentret. Telestyrelsen anlægger sag mod nøglecentret efter reglerne i den borgerlige retsplejes former.

Stk. 8. Anmodning om sagsanlæg har ikke opsættende virkning, men retten kan ved kendelse bestemme, at det pågældende nøglecenter under sagens behandling skal have adgang til at udstede kvalificerede certifikater. Ankes en dom, hvorved fratagelsen af adgangen til at udstede kvalificerede certifikater ikke findes lovlig, kan den ret, der har afsagt dommen, eller den ret, hvortil sagen er indbragt, bestemme, at nøglecentret ikke må udstede kvalificerede certifikater under ankesagens behandling.

§ 19. Telestyrelsen kan af nøglecentre kræve meddelt alle oplysninger, som findes nødvendige for tilsynet efter § 18, herunder til afgørelse af, om en fysisk eller juridisk person er omfattet af dette tilsyn.

Stk. 2. Nøglecentret og systemrevisor skal straks meddele Telestyrelsen oplysning om forhold, der er af afgørende betydning for nøglecentrets fortsatte virksomhed.

§ 20. Telestyrelsen kan pålægge nøglecentret inden for en fastsat frist at vælge en ny systemrevisor, jf. § 5, stk. 2, såfremt den fungerende systemrevisor findes åbenbart uegnet til sit hverv.

Stk. 2. Telestyrelsen kan pålægge systemrevisor at give oplysninger om nøglecentrets forhold uden accept fra nøglecentret.

Stk. 3. Ved revisorskifte skal nøglecentret og den eller de fratrådte systemrevisorer hver især give Telestyrelsen en redegørelse. Telestyrelsen kan give påbud om at efterkomme 1. pkt.

§ 21. Telestyrelsens afgørelser efter denne lov eller bestemmelser, der er fastsat i medfør heraf, kan ikke indbringes for anden administrativ myndighed.

§ 22. Forskningsministeren kan fastsætte regler om, at udgifterne ved Telestyrelsens tilsyn afholdes af de nøglecentre, der udsteder kvalificerede certifikater.

Kapitel 10
Internationale forhold

§ 23. Kvalificerede certifikater udstedt af et nøglecenter etableret i et land uden for Det Europæiske Økonomiske Samarbejde (EØS), skal anerkendes på samme måde som kvalificerede certifikater udstedt af nøglecentre etableret i et land inden for Det Europæiske Økonomiske Samarbejde (EØS) såfremt

1. nøglecentret opfylder kravene i denne lov og er tilsluttet en frivillig akkrediteringsordning i en medlemsstat eller

2. et nøglecenter etableret i en medlemsstat, der opfylder kravene i denne lov, indestår for certifikater udstedt af det pågældende nøglecenter eller

3. certifikatet eller nøglecentret er anerkendt i henhold til en bilateral eller multilateral aftale mellem Fællesskabet og tredjelande eller internationale organisationer.

Kapitel 11
Strafansvar

§ 24. Medmindre strengere straf er forskyldt efter anden lovgivning, straffes med bøde den, der

1. overtræder § 9, stk. 4, § 10, stk. 3, § 12 eller § 15, stk. 2,
2. afgiver urigtige eller vildledende oplysninger til Telestyrelsen eller
3. overtræder påbud eller afgørelser fra Telestyrelsen i medfør af § 18, stk. 2 og 6, og § 19, stk. 1.

Stk. 2. Der kan pålægges selskaber m.v. (juridiske personer) strafansvar efter reglerne i straffelovens 5. kapitel.

Stk. 3. Forældelsesfristen for strafansvar efter stk. 1 og 2 er 5 år.

Kapitel 12
Ikrafttrædelse m.v.

§ 25. Loven træder i kraft den 1. oktober 2000.

§ 26. Loven gælder ikke for Grønland og Færøerne, men kan ved kongelig anordning sættes i kraft for disse landsdele med de afvigelser, som de særlige grønlandske og færøske forhold tilsiger.

Bilag B

Java klasser

Dette appendix indeholder en oversigt, der kort introducerer de enkelte klasser. Den komplette kildekode er inkluderet i appendix C.

- CA
  Indeholder de standardindstillinger, som vi har valgt til vores demonstration. Dette inkluderer bl.a. valg af kommunikationsport, lokation af filer etc.

- CAServer
  Denne klasse indeholder den serverapplikation, der, på forespørgsel fra en klient, finder certifikater i databasen, signerer dem, og sender dem til klienten.

- CAManager
  Nøglecentrets applikation til udstedelse af nøgler og styring af server.

- Client
  Klient, der kan signere udgående, og verificere modtagne, beskeder.

- Signature
  Foretager signering og verificering af signaturer.

- Curve
  Definerer de forskellige domæneparametre.

- Point
  Repræsenterer et punkt på en elliptisk kurve.

- Math
  Mindre hjælpeklasse, indeholder konstanter for at gøre programmeringen nemmere samt SHA-1 funktionen.

- KeyData
  Indeholder informationer om nøglens art, dvs. om det er en privat eller offentlig nøgle etc., samt metoder til udtrækning af disse informationer.

- KeyInfo
  Indeholder de nødvendige parametre til nøglerne, dvs. den aktuelle kurve, navn på ejer, udløbsdato etc.

- FileData
  Indeholder standardfunktioner til filhåndtering.

- SignedFile
  Håndterer filer med tilhørende signaturer.

- Gui
  Beskriver en abstrakt grafisk brugergrænseflade, som senere kan implementeres.

- PromptGui
  Indeholder den eksisterende prompt-brugergrænseflade, der bruges i vores nuværende version af vores applikation.

Bilag C

Java kildekode

Vi gengiver herunder kildekoden til samtlige klasser. I koden er indsat *JavaDoc kommentarer* såvel som almindelige kommentarer. JavaDoc kommentarer bruges af programmet JavaDoc, som følger med en standard installation af Java. JavaDoc genererer en mængde HTML-filer, som dokumenterer programmets hierarkiske opbygning. Da disse HTML-filer kun giver mening i en browser, har vi lagt dem på rapportens hjemmeside, hvorfra kildekoden også kan hentes.

C.1 CA.java

```
/**
 * CA Class: This is the parent class of all CA classes, both client, server
 * and manager. In this class we mainly define constants.
 */
import java.net.*;  // java package for internet communication (TCP/IP)

public class CA {
  // The Server and Client communicates via TCP/IP. For experimental purposes, however,
  // we uses local directories for storing client and user keys.
  public static final String PATH_BASE = "program\\";
  public static final String PATH_USER = "keys\\users\\";
  public static final String PATH_CERT = "keys\\cert\\";
  public static final int PORT = 234; // default internet server port to use
  public InetAddress host = null;
  public int port = PORT; // actual port used for communicating with client.
  // Below is the built-in server key. It uses the largest NIST recommended curve.
  // The server key is valid for 4 yrs * 365 days per year = 1460 days.
  public static final KeyInfo CERT_KEY_INFO =
     new KeyInfo("Certificate Authority Server",
       "This is the CA Server key (should be hardcoded into the client)",
       Curve.P521,
       4 * 365);
```

```
/**
 * CA constructor. Various initialization.
 */
public CA() {
  try {
    host = InetAddress.getLocalHost();
  }
  catch (Exception e) {
    System.out.println("Network error occured starting CA.\nError message: " + e);
    host = null;
  }
  port = PORT;
}

/**
 * Returns a string decribing the CA system. Used mainly for test purposes.
 */
public String toString() {
  return new String("The CA is running on host: " + host
    +" on port " + port
    + "\nDefault base path is" + PATH_BASE
    + "\n  Cert keys are stored in subdirectory " + PATH_CERT
    + "\n  User Keys are stored in subdirectory " + PATH_USER);
}
}
```

C.2 CAServer.java

```
/**
 * Certificate Authority Server. Serves the clients with public keys on request.
 */
import java.io.*;
import java.net.*;

public class CAServer extends CA {
  private String desc = "Server has not been initialized.";
  private KeyData CAKey = null;
  private Gui myGui = null;
  private ServerSocket ss = null;

public CAServer(String descStr) {
  myGui = new PromptGui();
  desc = new String(descStr);
  myGui.setup();
  myGui.headline("CA Server");
  myGui.display("Server:" + desc);
  CAKey = new KeyData();
  if (!CAKey.load(PATH_CERT + CA.CERT_KEY_INFO.getId(), KeyData.KEY_PRIVATE))
    myGui.display("Problem loading cert key");
  else
    myGui.display("CA Server Keys loaded.");
  Curve curve = new Curve(CA.CERT_KEY_INFO.getCurveId());
}

public KeyData getKeyData(Integer keyId) {
  KeyData kd = new KeyData();
  kd.load(PATH_USER + keyId, KeyData.KEY_PUBLIC);
  kd.sign(CAKey);
  return kd;
}

public void initServer() {
  myGui.display("Initializing server...");
  myGui.display("Properties: Host: " + host + " port: " + port + "\n*****");
  boolean shutDown = true; // assume error and expect to shut down
  try {
    ss = new ServerSocket(port); // create a server socket on the specified port
    shutDown = false;  // no error. we keep server running.
  }
  catch (Exception e) {
    myGui.display("Network error occured" + e);
  }
  while (!shutDown) {
    try {
      System.out.println("Server ready. Waiting for client to request public keydata... ");
      Socket s = ss.accept();
      ObjectInputStream in = new ObjectInputStream(s.getInputStream());
      Integer req = (Integer) in.readObject();
      if (req != null) {
        System.out.println("Request recieved (" + req + "). Retrieving key from repository...");
        KeyData result = getKeyData(req); // here, key is signed according to CAKey
        if (result == null)
          System.out.println("No match was found. Sending empty KeyData");
        else
          System.out.println("Sending key for " + result.getKeyInfo().getName() + " to client...");
        ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream());
```

```
        out.writeObject(result);
      }
      s.close();
      if (req == null)
        shutDown = true;
      else
        System.out.println("Finished sending. Preparing to serve a new request...");
    }
    catch (Exception e) {
      myGui.display("Network error occured" + e);
      shutDown = true;
    }
  }
  System.out.print("Server is shutting down...");
  try {
    ss.close();
  }
  catch (Exception e) {
    myGui.display("Server problem. Exception: " + e);
  }
  System.out.println("Done. Have a nice day :-)");
}

/**
 * Starts the application.
 * @param args an array of command-line arguments
 */
public static void main(java.lang.String[] args) {
  CAServer serv = new CAServer("CA Server");
  serv.initServer();
}
}
```

C.3 CAManager.java

```
/**
 * User interfaced CA mangement application to generate keys.
 */
import java.io.*;
import java.math.*;
import java.security.*;
import java.net.*;

public class CAManager extends CA {
  int CAid;
  String desc = "CAManager";
  KeyData CAKey = null;
  Gui myGui = null;

/**
 * CAManager constructor. Loads server key and initializes manager.
 */
 public CAManager(String initDesc) {
  myGui = new PromptGui();
  desc = initDesc;
  CAKey = new KeyData();
  if (!CAKey.load(PATH_CERT + CA.CERT_KEY_INFO.getId(), KeyData.KEY_PRIVATE)) {
    myGui.display("CA keys not found.");
    if (myGui.askUser("You can either Quit or make new CA Keys. Do you wish to genereate new keys ?"))
      generateCAKeys();
    if (!CAKey.load(PATH_CERT + CA.CERT_KEY_INFO.getId(), KeyData.KEY_PRIVATE)) {
      myGui.display("Unable to read CA Keys from disk");
      System.exit(0); // error - quit manager.
    }
    else
      System.exit(0); // new keys successfully created - exit manager.
  }
}

/**
 * Method to genererate CA Server Key pair.
 */
public int generateCAKeys() {
  Curve curve = new Curve(CA.CERT_KEY_INFO.getCurveId());
  KeyData keyPair = new KeyData();
  keyPair.generateKeys();
  KeyData pri = new KeyData(keyPair.getPrivateKey(), null, CA.CERT_KEY_INFO, KeyData.KEY_PRIVATE);
  pri.save(PATH_CERT + CA.CERT_KEY_INFO.getId());
  KeyData pub = new KeyData(null, keyPair.getPublicKey(), CA.CERT_KEY_INFO, KeyData.KEY_PUBLIC);
  pub.save(PATH_CERT + CA.CERT_KEY_INFO.getId());
  return 0;
}

/**
 * Method to generate user key.
 */
public int generateKey() {
  myGui.display("Please enter information about the new user");
  String name = myGui.getString("Please enter name of user");
  String desc = myGui.getString("Please enter description of key (eg. Usage: ECDSA user keys)");
  int strength = myGui.selectStrength();
  myGui.display("Setting the expiry date to one year from now.");
  KeyInfo keyInf = new KeyInfo(name, desc, Curve.P192, 365);
```

```
  Curve curve = new Curve(keyInf.getCurveId());
  KeyData keyPair = new KeyData();
  keyPair.generateKeys();
  KeyData pri = new KeyData(keyPair.getPrivateKey(), null, keyInf, KeyData.KEY_PRIVATE);
  pri.sign(CAKey);
  pri.save(PATH_USER + keyInf.getId());
  KeyData pub = new KeyData(null, keyPair.getPublicKey(), keyInf, KeyData.KEY_PUBLIC);
  pub.save(PATH_USER + keyInf.getId());
  return 0;
}

/**
 * Starts the application.
 * @param args an array of command-line arguments
 */
public static void main(java.lang.String[] args) {
  CAManager cam = new CAManager("This is the CA Manager");
  cam.menu();
}

/**
 * Method to handle menu selection.
 */
public void menu() {
  myGui.setup();
  int mc = -1;
  while (mc != 0) {
    mc = myGui.managerMenu();
    switch (mc) {
      case 1 :
        {
          generateKey();
          break;
        }
      case 2 :
        {
          generateCAKeys();
          break;
        }
      case 3 :
        {
          serverShutDown();
          break;
        }
      case 0 :
        {
          myGui.shutdown();
        }
    } // end of switch
  } //  end of while
}

/**
 * Method to shutdown server. Sends a null object to the server, which is
 * interpreted as a shutdown-command.
 */
public void serverShutDown() {
  System.out.print("Shutting down server...");
  Socket s = null;
  try {
    s = new Socket(host, port);
    ObjectOutputStream out= new ObjectOutputStream(s.getOutputStream());
```

```
    out.writeObject(null);
    s.close();
  }
  catch (Exception e) {
    System.out.println("Exception " + e);
  }
  System.out.println("Done");

}
}
```

C.4 Client.java

```
/**
 * Client application. This is the main class containing methods to attach a
 * signature and verify a signature.
 */
import java.io.*;
import java.math.BigInteger;
import java.net.*;

public class Client extends CA {
  private String desc = "Client has not yet been initialized";
  private KeyData CAKey = null;
  private Gui myGui = null;

public Client(String initDesc) {
  myGui = new PromptGui();
  desc = initDesc;
  myGui.setup();
  myGui.headline("Client Application");
  myGui.display("Client:" + desc);
  CAKey = new KeyData();
  if (!CAKey.load(PATH_CERT + CA.CERT_KEY_INFO.getId(), KeyData.KEY_PUBLIC))
    myGui.display("Error loading public cert key");
  else
    myGui.display("Public Key from certificate has now been loaded.");
}

public void clientReciever() {
  String str = myGui.getFilename("", "Enter filename of signed file ", true, false);
  SignedFile sf = new SignedFile();
  sf.load(str); // load signed file.
  sf.detachFile(); // save file
  Integer sid = sf.getSignerId();
  myGui.display("Connecting Client to CA server on port " + port);
  Socket s = null;
  KeyData kd = null;
  try {
    s = new Socket(host, port);
    System.out.print("Requesting key " + sid + " from CA server... ");
    ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream());
    out.writeObject(sid);
    // right now, the server is looking for the key and sending it to the client
    ObjectInputStream in = new ObjectInputStream(s.getInputStream());
    kd = (KeyData) in.readObject();
    Signature CASig = kd.getSig();
    Curve curve = new Curve(CAKey.getKeyInfo().getCurveId());
    if (CASig.verify(kd.toHashable(), CAKey.getPublicKey()))
      System.out.println("The CA has signed this key");
    else
      System.out.println("The CA has NOT signed this key");
    s.close();
  }
  catch (Exception e) {
    System.out.println("Error: " + e);
  }
  // We now use the key, sent from the CA, to verify if the file is signed with the private key.
  Signature sig = sf.getSignature(); // extract signature from file.
  Curve curve = new Curve(kd.getKeyInfo().getCurveId());
  if (sig.verify(sf.toHashable(), kd.getPublicKey()))
```

```
    // verifying signature ...
    System.out.println("The signature of the file is correct");
  else
    System.out.println("Unable to verify signature on file");
}

public void clientSender() {
  String kfn = myGui.getFilename(CA.PATH_USER, "Insert Smart Card" +
    " or enter filename of private key file ", false, false);
  KeyData kd = new KeyData();
  kd.load(CA.PATH_USER + kfn, KeyData.KEY_PRIVATE); // loading private key.
  Curve curve = new Curve(kd.getKeyInfo().getCurveId());
  String str = myGui.getFilename("", "Enter filename of file to sign ", false, false);
  String newFilename = str.substring(0, str.indexOf("."));
  SignedFile sf = new SignedFile();
  sf.attachFile(str); // loading original unsigned file.
  sf.sign(kd); // signing file with private key.
  sf.save(newFilename + ".signed"); // saving in file with extension .signed
}

public static void main(java.lang.String[] args) {
  Client cli = new Client("This is the standard user client");
  cli.menu();
}

public void menu() {
  int mc = myGui.clientMenu();
  switch (mc) {
    case 1 :
      {
        clientSender();
        break;
      }
    case 2 :
      {
        clientReciever();
        break;
      }
    case 0 :
      {
        myGui.display("Closing client...");
        myGui.shutdown();
      }
  }
}
}
```

C.5 Signature.java

```
/**
 * The Signature class contains the actual digital siganture. It is created
 * by the constructor, and verified by the 'verify' method.
 */
import java.math.BigInteger;
import java.util.*;
import java.security.*;
import java.io.*;

public class Signature extends Curve implements Serializable{
  BigInteger r = null;
  BigInteger s = null;
  Integer signerId = null;

public Signature(byte[] sha1value, BigInteger privateKey, Integer sigId) {
  int runs = 0;
  BigInteger k = null;
  s = new BigInteger("0");
  while (s.equals(Math.ZERO)) {
    r = new BigInteger("0");
    while (r.equals(Math.ZERO)) {
      k = new BigInteger(bitlength, 10, new SecureRandom());
      while (k.compareTo(n.subtract(Math.ONE)) == 1) {
        k = new BigInteger(bitlength, 10, new SecureRandom());
      }
      Point xy = G.multiply(k);
      r = xy.getX().mod(n);
    } // end of while
    BigInteger kinv = k.modInverse(n);
    BigInteger e = null;
    e = new BigInteger(sha1value);
    s = kinv.multiply(e.add(privateKey.multiply(r)));
    s = s.mod(n);
    runs++;
  } // end of while
  signerId = sigId;
}

public BigInteger getR() {
  return r;
}

public BigInteger getS() {
  return s;
}

public Integer getSignerId() {
  return signerId;
}

/**
 * Create a string containing a describtion of the signature
 * @return java.lang.String A string describing the signature.
 */
public String toString() {
  return new String("Signature:\ns = " + s + "\nr = " + r);
}
```

```
/**
 * Verify a message's signature according to keypair.
 */
public boolean verify(byte[] sha1value, Point publicKey) {

  if ((s.compareTo(n.subtract(Math.ONE)) == 1) || (s.compareTo(Math.ZERO) == -1)) {
    System.out.println("S is not in the inverval 1.. n-1");
    return false;
  }
  if ((r.compareTo(n.subtract(Math.ONE)) == 1) || (r.compareTo(Math.ZERO) == -1)) {
    return false;
  }
  BigInteger e = null;
  e = new BigInteger(sha1value);
  BigInteger w = s.modInverse(n);
  BigInteger u1 = e.multiply(w).mod(n);
  BigInteger u2 = r.multiply(w).mod(n);
  Point X = new Point(true);
  X = X.add(G.multiply(u1));
  X = X.add(publicKey.multiply(u2));
  if (X.isInfinity()) {
    System.out.println("Rejecting. X = 0 (the point at infinity)");
    return false;
  }
  BigInteger v = X.getX().mod(n);

  if (v.compareTo(r) == 0)
    return true;
  else {
    return false;
  }
}
}
```

C.6 Curve.java

```
/**
 * This is the Curve class, used as parent class of most of the elliptic
 * curve methods. Curve defines the standard domain parameters.
 */
import java.math.*;

public class Curve {
  /** Constant to use when refering to the NIST recommended standard curves */
  public static final int P192 = 0;
  public static final int P224 = 1;
  public static final int P256 = 2;
  public static final int P384 = 3;
  public static final int P521 = 4;
  public static final String[] text = {"P192", "P224", "P256", "P384", "P521"};
  /** Standard values which are used when generating random curves. <br>
   *  They aren't used in this program, because we use the NIST curves */
  static final BigInteger STD_COFACTOR = Math.ONE;
  static final BigInteger STD_SEED = new BigInteger("999"); // not used!
  static final BigInteger STD_SHA_OUTPUT = new BigInteger("999"); // not used!
  static final BigInteger STD_A = new BigInteger("-3");
  // the bitlength of the numbers we choose when using a specific curve.
  protected static int bitlength = 0;
  // The fields below are taken from page 37 of the ECDSA standard //
  protected static BigInteger p; // the order of the prime field Fp.
  protected static BigInteger unused_seedE; //random seed of some kind.
  protected static BigInteger unused_r; // output of SHA-1 in algorithm 1.
  protected static BigInteger a, b; // the coefficients of the curve.
  protected static BigInteger xG, yG; // coordinates of the base point G.
  protected static Point G;
  protected static BigInteger n; // the prime order of G.
  protected static BigInteger unused_h; // the co-factor. It's 1 in the standard curves.

public Curve() {}
/**
 * Curve constructor initializing curve domain parameters, according to
 * the standard curves recommended by NIST for use with the ECDSA.
 * Notice that some parameters are given in decimal, and some in hexa-decimal.
 *
 * parameter "s" in FIPS 186-2 is "seedE" in ECDSA and in our implementation.
 *           "c" in FIPS 186-2 is "r" in ECDSA and in our implementation.
 *
 * defaults: "a" is always -3, for improved performance.
 *           "h", the co-factor, is always 1.
 */

public Curve(int standardCurve) {
  switch (standardCurve) {
  case P192 : {
    new Curve(
      new BigInteger("6277101735386680763835789423207666416083908700390324961279"),
      new BigInteger("3045AE6FC8422F64ED579528D38120EAE12196D5", 16),
      new BigInteger("3099D2BBBFCB2538542DCD5FB078B6EF5F3D6FE2C745DE65", 16),
      STD_A,
      new BigInteger("64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1",16),
      new BigInteger("188DA80EB03090F67CBF20EB43A18800F4FF0AFD82FF1012",16),
      new BigInteger("07192B95FFC8DA78631011ED6B24CDD573F977A11E794811",16),
      new BigInteger("6277101735386680763835789423176059013767194773182842284081"),
      STD_COFACTOR, 160);
```

```
    break;
  }
  case P224: {
    new Curve(new BigInteger("26959946667150639794667015087019630673557916260026308143510066298881"),
      STD_SEED,
      STD_SHA_OUTPUT,
      STD_A,
      new BigInteger("b4050a850c04b3abf54132565044b0b7d7bfd8ba270b39432355ffb4",16),
      new BigInteger("b70e0cbd6bb4bf7f321390b94a03c1d356c21122343280d6115c1d21",16),
        new BigInteger("bd376388b5f723fb4c22dfe6cd4375a05a07476444d5819985007e34", 16),
      new BigInteger("26959946667150639794667015087019625940457807714424391721682722368061"),
      STD_COFACTOR, 200);
    break;
  }
  case P256 : {
   new Curve(
       new BigInteger("115792089210356248762697446949407573530086143" +
       "415290314195533631308867097853951"),
       STD_SEED,
       STD_SHA_OUTPUT,
       STD_A,
       new BigInteger("5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b",16),
       new BigInteger("6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296",16),
       new BigInteger("4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5",16),
       new BigInteger("1157920892103562487626974469494075735299969 5" +
         "52241357603424222590610685120443 69"),
       STD_COFACTOR, 230);
   break;
  }


case P384: {
  new Curve(
  new BigInteger("39402006196394479212279040100143613805079739" +
    "27046544666794829340424572177149687032904726" +
    "60882589380018616069731123 19"),
  STD_SEED,
  STD_SHA_OUTPUT,
  STD_A,
  new BigInteger("b3312fa7e23ee7e4988e056be3f82d19181d9c6efe8141120314088f" +
     "5013875ac656398d8a2ed19d2a85c8edd3ec2aef",16),
  new BigInteger("aa87ca22be8b05378eb1c71ef320ad746e1d3b628ba79b9859f741e0" +
    "82542a385502f25dbf55296c3a545e3872760ab7", 16),
  new BigInteger("3617de4a96262c6f5d9e98bf9292dc29f8f41dbd289a147ce9da3113" +
    "b5f0b8c00a60b1ce1d7e819d7a431d7c90ea0e5f", 16),
  new BigInteger("39402006196394479212279040100143613805079739" +
    "27046544666794690527962765939911326356939895" +
    "63081522949135544336539426 43"),
  STD_COFACTOR, 360);
  break;
  }
case P521: {
  new Curve(
  new BigInteger("6864797660130609714981900799081393217269 43"+
    "5300143305409394463459185543183397656052122559" +
    "6406614545549772963113914808580371219879997 1" +
    "6643812574028291115057151"),
  STD_SEED,
  STD_SHA_OUTPUT,
  STD_A,
  new BigInteger("051953eb9618e1c9a1f929a21a0b68540eea2da725b99b315f3" +
                 "b8b489918ef109e156193951ec7e937b1652c0bd" +
```

```
                "3bb1bf073573df883d2c34f1ef451fd46b503f00",16),
  new BigInteger("c6858e06b70404e9cd9e3ecb662395b4429c648139053fb521" +
    "f828af606b4d3dbaa14b5e77efe75928fe1dc127a2ffa8de3348b3c1856a429bf97e7e31c2e5bd66",16),
  new BigInteger("11839296a789a3bc0045c8a5fb42c7d1bd998f54449579b4468" +
    "17afbd17273e662c97ee72995ef42640c550b9013fad0761353c7086a272c24088be94769fd16650",16),
  new BigInteger("6864797660130609714981900799081393217269435 3" +
    "0014330540939446345918554318339765539424505 7" +
    "7463332171975329639963713633211138647686 1244" +
    "0380340372808892707005449"),
  STD_COFACTOR,500);
break;
  }

default: {
    System.out.println("Standard curve is unknown. Standard curve is numbered 0 through 4");
}
}
}

/**
 * Initializes curve domain parameters.
 */
public Curve(BigInteger init_p, BigInteger init_seedE, BigInteger init_r,
  BigInteger init_a, BigInteger init_b, BigInteger init_xG, BigInteger init_yG,
  BigInteger init_n, BigInteger init_h, int init_bitlength) {
  p = init_p;
  unused_seedE = init_seedE;
  unused_r = init_r;
  a = init_a;
  b = init_b;
  xG = init_xG;
  yG = init_yG;
  n = init_n;
  unused_h = init_h;
  G = new Point(xG, yG, false);
  bitlength = init_bitlength;
}

/**
 * Method to check if the curve is valid.
 * The method prints out error messages if the curve isn't valid.
 * @return boolean True if the curve is valid, False otherwise.
 */
public boolean isValid() {
  if (!p.isProbablePrime(100)) {
    System.out.println("- ERROR --");
    System.out.println("Unable to conclude that the curve is valid, due to the following error:");
    System.out.println("The curve must be defined over Z_p where p is a prime!");
    System.out.println("A built-in java check has suggested that p is not a prime!");
    System.out.println("----------");
    return false;
  }
  if (!G.onCurve()) {
    System.out.println("- ERROR --");
    System.out.println("Unable to conclude that the curve is valid, due to the following error:");
    System.out.println("G is not on the curve");
    System.out.println("----------");
    return false;
  }
  BigInteger q1 = (a.pow(3).multiply(new BigInteger("4")));
  BigInteger q2 = (b.pow(2).multiply(new BigInteger("27")));
  q1 = q1.add(q2).mod(p);
```

```
  if (q1.equals(Math.ZERO))
    return false;
  else
    return true;
}
}
```

C.7 Point.java

```
/**
 * Defines a point on an Elliptic Curve. The point can be the point at infinity in which
 * case 'inf' is true. Otherwise the point is defined by two coordinates x and y.<br>
 */
import java.math.*;
import java.io.*;

public class Point extends Curve implements Serializable {
  private java.math.BigInteger xcoord;
  private java.math.BigInteger ycoord;
  private boolean inf;

public Point() {
}

public Point(String x, String y, boolean infval) {
  xcoord = new BigInteger(x);
  ycoord = new BigInteger(y);
  inf = infval;  // true means point at infinity
}

public Point(BigInteger x, BigInteger y, boolean infval) {
  this(x.toString(), y.toString(), infval);
}

public Point(Point p) {
  this(p.xcoord.toString(), p.ycoord.toString(), p.inf);
}

/**
 * Returns the point at infinity. <br>
 * The boolean variable in Point (inf) is set to true
 * regardsless of the value of the boolean parameter. This parameter is merely to
 * make this constructor distinct from the other constructors.
 */
public Point(boolean infval) {
  this("1","1",true);
}

/**
 * Adds a Point to this Point.
 * @return Point The sum of this and the given Point pnt.
 * @param pnt Point The Point to be added to this.
 */
public Point add(Point pnt) {
  /* First we isolate each coordinate according to the terminology given in
     Stinson page 184. We're adding two points: this and pnt */
  BigInteger x1 = this.xcoord;
  BigInteger y1 = this.ycoord;
  BigInteger x2 = pnt.xcoord;
  BigInteger y2 = pnt.ycoord;
  if (pnt.isInfinity() && this.isInfinity())
    return new Point(true);
  if (pnt.isInfinity() && !this.isInfinity())
    return new Point(x1, y1, false); // the sum is this.
  if (pnt.inf == false && this.inf == true)
    return new Point(x2, y2, false); // the sum is p2.
  if (x2.equals(x1) && y2.equals((y1.negate()).mod(p)))
```

```
    return new Point(true);
  BigInteger top = null;  // notation: we look at a fraction (top/down)
  BigInteger down = null;
  BigInteger lambda = null;
  if (!(this.equals(pnt))) {
    top = y2.subtract(y1); // y2-y1;
    down = x2.subtract(x1); // x2-x1;
  }
  else {
    top = x1.pow(2).multiply(Math.THREE);
    top = top.add(a); // 3*x1^2+a;
    down = y1.multiply(new BigInteger("2")); // 2y1;
  }
  down = down.mod(p);
  lambda = top.multiply(down.modInverse(p));
  lambda = lambda.mod(p);

  // calculate x3
  BigInteger x3 = lambda.modPow(Math.TWO, p);
  x3 = x3.subtract(x1);
  x3 = x3.subtract(x2);
  x3 = x3.mod(p);
  // calculate y3
  BigInteger y3 = lambda.multiply(x1.subtract(x3));
  y3 = y3.subtract(y1);
  x3 = x3.mod(p);
  y3 = y3.mod(p);
  return new Point(x3, y3, false);
}

/**
 * Compares two Points for equality. Returns a boolean that indicates
 * whether this Point is equivalent to the specified Point. <br> This method
 * overrides the default equals method, which is used when the Points are
 * stored in structures such as hashtables etc.<br>
 * @param obj Object The Point to compare with.
 * @return boolean. True if Points are equal. False otherwise.
 * @see java.util.Hashtable
 */
public boolean equals(Object obj) {
  Point p = (Point)obj;
  if (p.isInfinity() && this.isInfinity())
    return true; // both are infinity.
  if (p.isInfinity() != this.isInfinity())
    return false;
  if (p.getX().equals(this.getX()) && p.getY().equals(this.getY()))
    return true;
  return false;
}

/**
 * Extracts the x coordinate from the point.
 * @return java.math.BigInteger The x coordinate.
 */
public BigInteger getX() {
  return xcoord;
}

/**
 * Extracts the y coordinate of the point.
 * @return java.math.BigInteger The y coordinate.
 */
```

```
public BigInteger getY() {
  return ycoord;
}

/**
 * Invert this Point.
 * @return Point The inverse of this Point.
 */
public Point inverse() {
  if (isInfinity())
    return new Point(true);
  else
    return new Point(xcoord, (ycoord.negate()).mod(p), false);
}

/**
 * Checks whether this point is the point at infinity.
 * @return boolean True when the Point is the Point at infinity, false otherwise.
 */
public boolean isInfinity() {
  return inf;
}

/**
 * Calculate the j * this point, "Double And Add".
 * @return Point The result
 * @param j java.math.BigInteger The number of times to add the point with itself.
 */
public Point multiply(BigInteger d) {
  Point sum = new Point(true);
  Point R = this;
  for (int i = 0; i < d.bitLength(); i++) {
    if (d.testBit(i))
      sum = sum.add(R);
    R = R.add(R);
  }
  return sum;
}

/**
 * Checks whether this point is on the curve.
 * @return boolean True if the point is on the curve. False otherwise.
 */
public boolean onCurve() {
  BigInteger leftHandSide = ycoord.pow(2);
  BigInteger rightHandSide = xcoord.pow(3);
  rightHandSide = rightHandSide.add(a.multiply(xcoord)).add(b);
  leftHandSide = leftHandSide.mod(p);
  rightHandSide = rightHandSide.mod(p);
  return rightHandSide.equals(leftHandSide);
}

public String toString() {
  if (this.isInfinity())
    return "INFINITY";
  else
    return " (" + xcoord + ", " + ycoord + ") ";
}
}
```

C.8 Math.java

```
/* This class is empty, except for the 4 constants below and a
 *  SHA-1 calculating method. */
import java.math.BigInteger; // big integer support
import java.security.MessageDigest; // SHA-1 support

public class Math {
  public static final BigInteger ZERO = new BigInteger("0");
  public static final BigInteger ONE = new BigInteger("1");
  public static final BigInteger TWO = new BigInteger("2");
  public static final BigInteger THREE = new BigInteger("3");

public Math() {
}

/**
 * This method uses the standard java SHA-1 implementation to calculate
 * the SHA-1 hash value of a byte array. This method presents nothing new,
 * but merely provides an easy-to-use version of the SHA-1 algorithm.
 *
 * @return byte[] The SHA-1 hash value.
 * @param message byte[] The message to be hashed.
 */
public static byte[] SHA1(byte[] message) {
  try {
    MessageDigest md = MessageDigest.getInstance("SHA");
    md.update(message);
    return md.digest();
  }
  catch (Exception excep) {
    System.out.println("Error calculating SHA-1!.");
    System.exit(0);
  }
  return null;
}
}
```

C.9 KeyData.java

```
/**
 * This class contains public and private key data
 */
import java.math.BigInteger;
import java.util.*;
import java.security.MessageDigest; // SHA-1
import java.io.*;

class KeyData extends Curve implements Serializable {
  // Constants to make programming easier:
  public static final String KEY_PRIVATE = "private";
  public static final String KEY_PUBLIC = "public";
  private Signature sig = null;
  private Point Q = null; // public key
  private BigInteger d = null; // private key
  private KeyInfo ki = null; // name, description, create date, expire date, id, etc.
  private String keyType = null; // both, private or public

public KeyData() {
}

public KeyData(BigInteger init_d, Point init_Q, KeyInfo keyInf, String kt) {
  if (init_d != null)
    d = new BigInteger(init_d.toString());
  if (init_Q != null)
    Q = new Point(init_Q);
  keyType = new String(kt);
  ki = keyInf;
}

public void addKeyInfo(KeyInfo keyInf) {
  ki = keyInf;
}

public void generateKeys() {
  System.out.print("Generating Keys: Private key ...");
  d = new BigInteger(bitlength, new java.security.SecureRandom());
  while (d.compareTo(n.subtract(Math.ONE)) == 1) {
    System.out.println("d didn't work. Renewing d...");
    d = new BigInteger(bitlength, new java.security.SecureRandom());
    System.out.println("d is created again and is now: " + d);
  }
  System.out.print("Public...");
  Q = G.multiply(d);
  System.out.println("Done.");
}

public KeyInfo getKeyInfo() {
  return ki;
}

public String getKeyType() {
  return new String(keyType);
}

public BigInteger getPrivateKey() {
  return d;
}
```

```
public Point getPublicKey() {
  return Q;
}

public Signature getSig() {
  return sig;
}

public boolean isPrivate() {
  return (keyType.equals(KEY_PRIVATE));
}

public boolean isPublic() {
  return (keyType.equals(KEY_PUBLIC));
}

public boolean load(String filename, String whichKeys) {
  String fullname = new String(CA.PATH_BASE + filename + "." + whichKeys);
  try {
    FileInputStream fis = new FileInputStream(fullname);
    ObjectInputStream in = new ObjectInputStream(fis);
    String ltype = (String) in.readObject();
    if (!ltype.equals(whichKeys)) {
      System.out.println("You requested a " + whichKeys +
        " key, but this key is " + ltype);
      return false;
    }
    keyType = new String(ltype);
    ki = (KeyInfo) in.readObject();
    if (isPublic())
      Q = (Point) in.readObject();
    if (isPrivate())
      d = (BigInteger) in.readObject();
    in.close();
    return true;
  }
  catch (Exception e) {
    System.out.println("Failed. Error loading KeyFile" + e);
    System.out.println("Filename used: " + fullname);
    return false;
  }
}

public boolean save(String filename) {
  String fullname = new String(CA.PATH_BASE + filename + "." + keyType);
  try {
    FileOutputStream fos = new FileOutputStream(fullname);
    ObjectOutputStream out = new ObjectOutputStream(fos);
    out.writeObject(keyType);
    out.writeObject(ki);
    if (isPublic())
      out.writeObject(Q);
    if (isPrivate())
      out.writeObject(d);
    out.flush();
    out.close();
    return true;
  }
  catch (Exception e) {
    System.out.println("Failed. Error saving KeyFile" + e);
    System.out.println("Filename used: " + fullname);
```

```
    return false;
  }
}

public void setKeyType(String kt) {
  keyType = new String(kt);
  }

public void sign(KeyData keyd) {
  String hashString =   new String("" + d + Q + ki.toHashable());
  byte[] hashValue = Math.SHA1(hashString.getBytes());
  sig = new Signature(hashValue, keyd.getPrivateKey(), keyd.getKeyInfo().getId());
}

public byte[] toHashable() {
  String str = new String("" + d + Q + ki.toHashable());
  return str.getBytes();
}

public String toString() {
  return new String("======================\n"+
    "KeyData:\n\tNumber d = " + d+ "\n" +
    "\tPoint Q = " + Q + "\n" + ki + "\n" +
    "Signature: " + sig);
}
}
```

C.10 KeyInfo.java

```
/**
 * This class holds information about the current key. None of these data are used cryptographically,
 * but a the contained Expiry date can determine whether or not this key is valid and can be used.
 */
import java.util.*;
import java.io.*;

public class KeyInfo implements Serializable {
  private int curveId = Curve.P192; // the strength of the curve. This int defines which std. curve to use.
  private Date created = null;  // the creation date and time of this KeyInfo
  private Date expire = null;  // the expirery date of this KeyInfo. Cannot be used past this date.
  private String name = null;  // A string containing the full name of the owner of the key.
  private String description = null;  // description of the crypto system, it's use, etc.
  private Integer id = new Integer(0);  // hashCode of name, deescription and curveID

public KeyInfo() {
}

/**
 * KeyInfo constructor commonly used. Sets the name, description, curveID of the KeyInfo.
 * Futhermore the creation time is set to the current system time and the expiry date is
 * set to the current time plus the number of days specified in <code>days</code><br>
 * @param n String   The name of the person to which this key belongs.
 * @param d String  The description of the person or the key and it's uses.
 * @param cid int    The curveId. This can be either 0,1,2,3,4 refering to the std. curves P192 through P521.<br>
 * @see Curve
 * @param days int  The number of days from today that this key is valid and usable.
 */
public KeyInfo(String n, String d, int cid, int days) {
   created = new Date();
   expire = new Date();
   expire.setTime(created.getTime()+(1000*60*60*24*days));
   name = new String(n);
   description = new String(d);
   curveId = cid;
   String idStr = new String(n + d + cid);
   id = new Integer(idStr.hashCode());
}

/**
 * Returns the CurveID, as one of the standard curves defined in Curve.
 * @see Curve
 * @return int The curveID of the curve used with this key.
 */
public int getCurveId() {
  return curveId;
}

/**
 * Returns the id of this key. The id is a hashCode formed on the basis of name, description and curveID.
 * @return int The hashCode id.
 */
public Integer getId() {
  return id;
}

/**
 * Returns the name of the person owning this key.
```

```
 * @return java.lang.String The name
 */
public String getName() {
  return name;
}

/**
 * Returns a string describing the entire KeyInfo object.
 * It's a big string, but nothing's left out. ;-)
 * @return java.lang.String The KeyInfo content as a string.
 */
public byte[] toHashable() {
  String str = new String(curveId + name + description + created + expire);
  return str.getBytes();
}

/**
 * Returns a string describing the entire KeyInfo object. It's a big string, but nothing's left out.
 * @return java.lang.String The KeyInfo content as a string.
 */
public String toString() {
  return new String("----------\nKeyInfo:\n"
    + "Curve = " + Curve.text[curveId] + "(" + curveId + ")\n"
    + "Name: " + name + " Desc: " + description + "\n" +
    "Created: " + created + " Expires: " + expire + "\n----------");
}
}
```

C.11 FileData.java

```
/**
 * This class is made only to wrap the file and filename together.
 */
import java.io.Serializable;

public class FileData implements Serializable {
  private String filename = null;
  public byte[] data = null;

public FileData(String fn) {
  filename = new String(fn);
}

public void fill(byte[] b) {
  data = new byte[b.length];
  System.arraycopy(b, 0, data, 0, b.length);
}

public byte[] getBytes() {
  return data;
}

public String getFilename() {
  return filename;
}
}
```

C.12 SignedFile.java

```
/**
 * This class wraps a signature and a FileData class.
 */
import java.io.*;

public class SignedFile {
  private FileData fileData = null;
  private Signature signature = null;
public SignedFile() {
}

public void attachFile(String filename) {
  try {
    File fil = new File(CA.PATH_BASE+filename);
    byte[] data = new byte[ (int) fil.length()];
    FileInputStream fis = new FileInputStream(CA.PATH_BASE + filename);
    DataInputStream in = new DataInputStream(fis);
    in.readFully(data);
    fileData = new FileData(filename);
    fileData.fill(data);
    in.close();
  }
  catch (Exception e) {
    System.out.println("Failed. Error loading file " + e);
    System.out.println("Filename used: " + filename);
  }
  signature = null;
}

public void detachFile() {
  detachFile(fileData.getFilename());
}

public void detachFile(String filename) {
  System.out.println(fileData);
  try {
    FileOutputStream fos = new FileOutputStream(CA.PATH_BASE + filename);
    fos.write(fileData.getBytes());
    fos.close();

  }
  catch (Exception e) {
    System.out.println("Failed. Error saving filename" + e);
    System.out.println("Filename used: " + filename);
  }
}

public Signature getSignature() {
  return signature;
}

public Integer getSignerId() {
  return signature.getSignerId();
}

/**
 * Load a signed file.
 */
```

```
public void load(String signedFilename) {
  try {
    FileInputStream fis = new FileInputStream(CA.PATH_BASE + signedFilename);
    ObjectInputStream in = new ObjectInputStream(fis);
    fileData = (FileData) in.readObject();
    signature = (Signature) in.readObject();
    in.close();
  }
  catch (StreamCorruptedException e) {
    System.out.println("Stream is corrupted. You're probably trying to load" +
      "a file, which isn't a signed file (.signed)");
    System.out.println("Exception error message: " + e);
  }
  catch (Exception e) {
    System.out.println("Failed. Error loading signedFilename" + e);
    System.out.println("Filename used: " + signedFilename);
  }
}

public void save(String signedFilename) {
  try {
    FileOutputStream fos = new FileOutputStream(CA.PATH_BASE + signedFilename);
    ObjectOutputStream out = new ObjectOutputStream(fos);
    out.writeObject(fileData);
    out.writeObject(signature);
    out.close();
  }
  catch (Exception e) {
    System.out.println("Failed. Error saving signedFilename" + e);
    System.out.println("Filename used: " + signedFilename);
  }
}

public void sign(KeyData kd) {
  signature = new Signature(Math.SHA1(toHashable()), kd.getPrivateKey(), kd.getKeyInfo().getId());
}

public byte[] toHashable() {
  return fileData.getBytes();
}
}
```

C.13 Gui.java

```
/**
 * Generic interface to describe a graphical user interface. Makes it possible
 * to replace the entire user inface with a, say, Windows userinteface, without
 * needing to change the underlying code.
 */
public interface Gui {
  public static final String[] curveText = {"P192", "P224", "P256", "P384", "P521"};

public boolean askUser(String question);
public int clientMenu();
public void display(String dStr);
public String getFilename(String path, String msg, boolean mustExists, boolean promptOverwrite);
public String getString(String msg);
public void headline(String hStr) ;
public int managerMenu();
public int selectStrength();
void setup();
void shutdown();
}
```

C.14 PromptGui.java

```
/**
 * Implements a command-line prompt user interface according to "Gui".
 */
import java.io.*;

public class PromptGui implements Gui {

public PromptGui() {
}

/**
 * Prints out the string specified in question and accepts either y,Y,n or N.
 * Returns a boolean value accordingly.
 * @return boolean True means yes and false means no
 * @param question java.lang.String The question to prompt to the user.
 */
public  boolean askUser(String question) {
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
  String line = "";
  System.out.print(question);
  while (true) {
    try {
      line = in.readLine();
    }
    catch (Exception e) {
    }
    line = line.toUpperCase();
    if (line.equals("Y"))
      return true;
    else
      if (line.equals("N"))
        return false;
      else
        System.out.println("Please answer either YES or NO by typing Y or N and pressing <enter>");
  }
}

public int clientMenu() {
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in)); // keyboard input
  int choice = 0; // menuitem chosen by user
  String line = null;

  // Display main menu using beautiful ascii graphics
  System.out.println("##############################################\n ECDSA Client Menu\n");
  System.out.println("1 - Sign   - Load file, sign it, and save it again.\n");
  System.out.println("2 - Verify - Load file, get public blabla.. verify and save unsigned.\n");
  System.out.println("0 - Exit\n");
  System.out.println("---------------------------\n");
  System.out.print("Please select from menu: ");
  try {
    line = in.readLine();
  }
  catch (Exception e) {
    System.out.println("INPUT ERROR");
    return 0;
  }
  try {
    choice = Integer.parseInt(line);
```

```
  }
  catch (Exception e) {
    System.out.println("Please enter a number.");
    return 0;
  }
  // below is the main selection loop. The variable i contains the selected menu item number.
  if (choice >= 0 && choice <= 2) {
    return choice;
  }
  else {
    System.out.println("Invalid choice. Using 0");
    return 0;
  }
}

public void display(String dStr) {
  System.out.println(dStr);
}

public String getFilename(String path, String msg, boolean mustExist, boolean promptOverwrite) {
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
  boolean okFilename = false;
  String line = "";
  while (!okFilename) {
    okFilename = true;
    System.out.println(msg);
    System.out.print("Relative to " + CA.PATH_BASE + path + " ===>");
    try {
      line = in.readLine();
    }
    catch (Exception e) {
    }
    if (line.length() == 0)
      return null;
  //line = line.toUpperCase();
    if (line == null) {
      System.out.println("Please enter a filename ");
      okFilename = false;
    }
    File f = new File(CA.PATH_BASE+path+ line);
    if (mustExist) {
      if (!f.exists()) {
        System.out.println("File doesn't exist. Please enter a path and file name of an already existing file.");
        okFilename = false;
      }
    }
    if (promptOverwrite && f.exists())
      okFilename = askUser("File already exists. Overwrite (Y/N) ?");
  }
  return line;
}

public String getString(String msg) {
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
  String line = "";
  System.out.println(msg);
  System.out.print("===>");
  try {
    line = in.readLine();
  }
  catch (Exception e) {
  }
```

```
  if (line.length() == 0)
    return null;
  return line;
}

/**
 * Prints out a headline - ASCII graphics rules!
 */
public void headline(String hStr) {
  System.out.println("* * * *\t"+ hStr+"\t* * * *");
}

public int managerMenu() {
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in)); // keyboard input
  int choice = 0; // menuitem chosen by user
  String line = null;

  // Display main menu using beautiful ascii graphics
  System.out.println("#########\n CA Manager Menu\n");
  System.out.println("1 - Create Key Pair in .private and .public (signed)\n");
  System.out.println("2 - Init CA. Create CAKeys.\n");
  System.out.println("3 - Shutdown Server\n");
  System.out.println("0 - Exit\n");
  System.out.println("--------------------------\n");
  System.out.print("Please select from menu: ");
  try {
    line = in.readLine();
  }
  catch (Exception e) {
    System.out.println("INPUT ERROR");
    return 0;
  }
  try {
    choice = Integer.parseInt(line);
  }
  catch (Exception e) {
    System.out.println("Please enter a number.");
    return 0;
  }
  // below is the main selection loop. The variable i contains the selected menu item number.
  if (choice >= 0 && choice <= 3) {
    return choice;
  }
  else {
    System.out.println("Invalid choice. Using 0");
    return 0;
  }
}

/**
 * Displays a menu and allows user to select Curve Size. Retiurn P192 if ANYTHING goes wrong
 * @return int The selected curve (0,1,2,3 or 4)
 */
public int selectStrength() {
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in)); // keyboard input
  int choice = 0; // menuitem chosen by user
  String line = null;

  // Display main menu using beautiful ascii graphics
  System.out.println("#####\n ECDSA: Select Size of Elliptic Curve \n");
  System.out.println("0 - NIST Curve P-192\n");
  System.out.println("1 - NIST Curve P-224\n");
```

```
  System.out.println("2 - NIST Curve P-256\n");
  System.out.println("3 - NIST Curve P-384\n");
  System.out.println("4 - NIST Curve P-521\n");
  System.out.println("--------------------------\n");
  System.out.print("Please select from menu: ");
  try {
    line = in.readLine();
  }
  catch (Exception e) {
    System.out.println("INPUT ERROR");
    return 0;
  }
  try {
    choice = Integer.parseInt(line);
  }
  catch (Exception e) {
    System.out.println("Please enter a number.");
    return 0;
  }
  // below is the main selection loop. The variable i contains the selected menu item number.
  if (choice >= 0 && choice <= 4) {
    System.out.println("Using Elliptic Curve " + curveText[choice]);
    return choice;
  }
  else {
    System.out.println("Invalid choice. Using 0");
  return 0;
  }

}

public void setup() {
  System.out.println("Welcome to the Command Line Prompt GUI.");
}

public void shutdown() {
}
}
```

Bilag D

Bitrepræsentation af kurver

Vi gengiver herunder bitrepræsentationen af domæneparameteren p i de enkelte standardkurver. Udregningerne er foretaget i Maple, og scriptet kan hentes fra rapportens hjemmeside.

P-192

decimal

```
2^192-2^64-1=
6277101735386680763835789423207666416083908700390324961279
```

binær

```
111111111111111111111111111111111111111111111111111111111111111111111111\
111111111111111111111111111111111111111111111111011111111111111111111111\
111111111111111111111111111111111111
```

P-224

decimal

```
2^224-2^96+1=
26959946667150639794667015087019630673557916260026308143510066298881
```

binær

```
111111111111111111111111111111111111111111111111111111111111111111111111\
111111111111111111111111111111111111111111111111000000000000000000000000\
000000000000000000000000000000000000000000000000000000000000000000000001
```

P-256

decimal

```
2^256-2^224+2^192+2^96-1=
115792089210356248762697446949407573530086143415290314195533631308\
67097853951
```

binær

```
11111111111111111111111111111110000000000000000000000000000001000000000000\
00000000000000000000000000000000000000000000000000000000000000000000000000\
00000000111111111111111111111111111111111111111111111111111111111111111111\
11111111111111111111111111
```

P-384

decimal

```
2^384-2^128-2^96+2^32-1=
39402006196394479212279040100143613805079739270465446667948293404245\
72177149687032904726608825893800186160697311231 9
```

binær

```
11111111111111111111111111111111111111111111111111111111111111111111111111\
11111111111111111111111111111111111111111111111111111111111111111111111111\
11111111111111111111111111111111111111111111111111111111111111111111111111\
11111111111111111111111111011111111111111111111111111111110000000000000000\
00000000000000000000000000000000000000000000000111111111111111111111111111\
1111
```

P-521

decimal

```
2^521-1=
68647976601306097149819007990813932172694353001433054093944634591855\
43183397656052122559640661454554977296311391480858037121987999716643\
81281574028291115057151
```

binær

```
11111111111111111111111111111111111111111111111111111111111111111111111111\
11111111111111111111111111111111111111111111111111111111111111111111111111\
11111111111111111111111111111111111111111111111111111111111111111111111111\
11111111111111111111111111111111111111111111111111111111111111111111111111\
11111111111111111111111111111111111111111111111111111111111111111111111111\
11111111111111111111111111111111111111111111111111111111111111111111111111\
11111111111111111111111111111111111111111111111111111111111111111
```