Elliptisk Kurve Kryptografi

Jonas F. Jensen

December 2007

Abstract

Today we're using cryptography everytime whenever we're doing transactions online. Some have even adopted cryptography to sign their emails with a digital signature. With the evergrowing holes in our modern and adopted distributed communication protocols like GSM and email, these technologies are not going to be less interessting in the future. Demand for electronic security and privacy is not going away any time soon.

This report is about Elliptic Curve Cryptography with focus on the Elliptic Curve Digital Signature Algorithm. The basic concepts of abstract algebra and asymmetric cryptography will be explained in this report, and it should be able to serve as the mathematical foundation for an implementation of the Elliptic Curve Digital Signature Algorithm. Such an implementation is present in appendix D, this report also features benchmarkings of this implementation.

Based on these benchmarks I've reached the conclusion that performance and security is two sides of the same coin. The slower performance the higher security and low security the higher performance. Thus one must evaluate his security needs based on what performance sacrifice his willing to do. On some mobile platforms this may become a critical issue.

Forord

I dag benytter vi kryptografi hver gang vi bruger netbank og dankort, nogle af os bruger det også når vi tjekker e-mail, installere software eller bare surfer på nettet. Men det er vel faktisk de færreste der ved hvordan disse algoritmer virker, og kan bevise deres integritet.

I dette projekt vil jeg arbejde med en digital signatur algoritme, der hedder ECDSA (Elliptic Curve Digital Signature Algorithm). Algoritmen er bla. standardiseret af ANSI (American National Standards Institute) og IEEE (Institute of Electrical and Electronics Engineers). I projektet vil jeg ligge særlig vægt på det matematiske aspekt af denne algoritme, og denne rapport skal gerne kunne danne grundlag for en praktisk implementering af ECDSA.

Efter at have kigget i flere bøger om emnet har jeg erfaret at der ikke findes en fælles notations form. Notationerne afviger fra hinanden, specielt afhængig af kildens alder. Derfor har jeg valgt at benytte det bedste fra de forskellige notations former og skabt min egen notation, der hvor der ikke eksisterede noget logisk forbillede. Denne rapport er, lige som de fleste af de bøger jeg referer til, skrevet i LaTeX. Dette skulle gerne give mulighed for at benytte korrekte notationer. Jeg har forsøgt at være konsekvent gennem min rapport, og forklaret min notation. Tiltider har jeg vægtet læsevenlighed over informationstæthed, hvorved jeg muligvis har overskredet omfanget af denne rapport en smule. Men jeg mener at det er svært forståeligt at $k = \sum_{i=0}^{\lfloor \log_2(k) \rfloor} k_i \cdot 2^i$, betyder at k kan læses i en binær repræsentation, derfor har jeg, imodsætningen til flere af mine kilder, forklaret denne slags notationer.

Jeg har i dette projekt benytte rigtig manger kilder, mange af dem primært som opslagsværker. Jeg har fundet at særlig Certicom's "Online Elliptic Curve Cryptography Tutorial", giver en god overordnet introduktion til emnet. Deres matematik appendiks giver også en god praktisk orienteret indgangsvinkel på abstrakt algebra. Dernæst "Handbook of Applied Cryptography", da som titlen antyder er en prakisk orienteret bog, der præsentere en masse fakta, teorier og algoritmer. Bogen et fantastisk opslagsværk der godt nok ikke dækker ECDSA, men har mange gode nummerteoretiske algoritmer, fakta og teorier. Denne bog kan også findes gratis online, se litteraturlisten appendiks A. Sidst men bestemt ikke mindst skal Wikipedia nævnes, som et godt opslagsværk til mange af disse emner.

Indhold

Figurer

Tabeller

Algoritmer

1 Introduktion til kryptografi

Kort introduktion til de forskellige former for kryptografiske algoritmer og deres anvendelses muligheder.

1.1 Hash algoritmer

En hash algoritme er en algoritme der kan kryptere data så det efterfølgende ikke kan dekrypteres. Et eksempel på en hash algoritme er md5. Hvis man tager en hashsum af "Quizdeltagerne spiste jordbær med fløde, mens cirkusklovnen Walther spillede på xylofon." med md5, får man "34d676994890bef4a0dd9b106b375790". Det smarte med en sådan algoritme er, at man altid får den samme streng når inputet er det samme. Længden af resultatet afhængiger af hvilken hash algoritme man benytter. Lignende hash algoritmer bruges bla. til at verificere en fils integritet efter netværkstransmission. Sådanne algoritmer benyttes ofte på internettet, f.eks. når du opretter en bruger på et website, vil serveren normalt ikke gemme dit password, men istedet blot en hashsum af dit password. Når du logger ind, tager serveren en hashsum af det oplyste password, og sammenligner det med den hashsum, der blev gemt fra da du oprettede brugeren. Skulle det ske at webserveren blev kompromitteret, vil en evt. hacker ikke kunne se dit password, men kun en hashsum af dit password, som hackeren ikke kan bruge til at logge ind med. Der vil også blive benyttet hash algoritmer senere i denne rapport, det matematiske aspekt af disse algoritmer vil dog ikke blive diskuteret.

1.2 Symmetrisk kryptografi

Symmetrisk kryptering er nok den type kryptering som er lettest at forholde sig til. Hvis man i sine unge dage har været spejder eller FDF'er har man sikkert prøvet at løse koder. Det kunne f.eks. være at alle tegn i alfabetet var blivet forskudt. Symmetrisk kryptografi betyder at både modtager og afsender har samme nøgle. En symmetrisk krypterings algoritme, tager 2 parametre en medelelse m og et kodeord d, funktionen returnere så en krypteret meddelse $Q = f(d, m)$. Derudover eksistere der en invers funktion sådan at $m = f^{-1}(d, Q)$. På den måde kan 2 personer sende krypterede postkort til hinanden, uden at postmanden kan læse beskederne. Denne type kryptografi kræver at både modtager og afsender har adgang til kodeordet, og at kodeordet er hemmeligt. Det er denne type kryptografi anvendes når vi kryptere et fil-arkiv eller opsætter vores harddisk til være krypteret.

1.3 Asymmetrisk kryptografi

Dette er en af de algoritme typer der kan være lidt svære at forholde sig til. I modsætningen til symmetrisk kryptografi kræver denne algoritmetype ikke at både modtager og afsender kender det samme kodeord.

1.3.1 Offentlig/privat nøgle kryptering

Hvis Per og Lise vil sende hemmelige medelelser til hinanden gennem en usikker kommunikationsvej, som f.eks. postkort, kan de benytte asymmetriske kryptering. Det gør de ved at Per først generer et nøglesæt, bestående af en offentlig nøgle Q, og en privat nøgle d. Den private nøgle d er selvfølgelig hemmelig, mens han giver den offentlige nøgle Q til Lise. Nu kan Lise benytte en asymmetrisk krypterings algoritme til at kryptere hendes medelelse m, sådan at $v = f(m, Q)$. v for volapyk, Lisa kan nemlig ikke dekryptere v med den offentlige nøgle Q. Men hun kan godt udregne $v = f(m, Q)$, når hun så har gjort det sender hun et postkort med v til Per. Per kan så tage den inverse funktion $m = f^{-1}(v, d)$, hvorved kan han læse medelelsen m. Men da den inverse funktion tager d som parameter er der ingen andre der kan læse Pers medelelser, alle kan dog skrive medelelser til Per. Denne type algoritmer benyttes bl.a. til netbank og e-mail kryptering.

1.3.2 Digital signatur

Hvis Per og Lise ikke ønskede at sende hemmelige beskeder, men bare ville være sikre på at postmanden ikke ændrer deres breve, kunne de istedet benytte en asymmetrisk signatur algoritme. Igen skal Per genere et nøglesæt, bestående af en offentlig nøgle Q og en privat nøgle d. Den private nøgle d selvfølgelig hemmelig, mens han giver den offentlige nøgle Q til Lise. Nu kan Per bruge en asymmetrisk signatur algoritme til at skabe en signatur af hans medelelse m, sådan at $s = sign(m, d)$. Derefter sender han Lise et brev med m og s. Lise kan så benytte en funktion til at verificere om signaturen er gyldig, dette gør hun med $verify(m, s, Q)$. Funktion afgør om s er blevet genereret med m og den private nøgle d der hører til Q.

Umiddelbart virker asymmetriske krypterings algoritmer og asymmetriske signatur algoritmer matematiske umulige at opbygge. Men man kan, ved at udnytte nogle matematiske problemer, skabe denne slags algoritmer. Denne rapport handler om den asymmetriske signatur algoritme ECDSA. Der antages læseren nu har styr på relationerne mellem signatur s, medelelse m, offentlig nøgle Q og privat nøgle d. Det vil ikke blive yderligere præciseret hvem af de kommunikerende parter der ved hvad.

2 Abstrakt algebra

Introduktion til abstrakt algebra, og forklaring af de forskellige begreber og strukturer, som vil blive benyttet i denne rapport.

2.1 Introduktion til abstrakt algebra

Abstrakt algebra eller moderne algebra, er som navnet antyder er smule abstrakt. Man kan godt betragte abstrakt algebra som et højere abstraktion niveau end almindelig algebra. I almindelig algebra beskæftiger man sig med omskrivning og reduktion af ligninger. På arabisk betyder algebra reduktion, efter titlen på en bog af Muhammad ibn Musa al-Khwarizmi i år 825. Abstrakt algebra er forholdvist nyt og er først blevet skabt og udforsket igennem de sidst to århundreder. Abstrakt algebra er blandt andet blevet til i arbejdet med at løse ligninger af høje grader [5].

I dag benyttes denne teori mange steder, og senere vil man da også opdage at man allerede er bekendt med tal mængder som f.eks. $\mathbb{R}$. I dette afsnit vil jeg forsøge at beskrive nogle af de begreber og strukturer, der indgår i abstrakt algebra. Jeg vil såvidt mulig at benytte danske termer, selvom største delen af litteraturen er på engelsk. Den danske terminologi har jeg i høj grad hentet fra [20], [15] og [16], når jeg har fundet den for obskur har jeg beholdt den engelske terminologi.

2.2 Mængde teori

I matematik har vi arbejdet med definitions mængden (DM) og værdi mængden VM for funktioner. Oftest har vi udtrykt definitions mængden for et funktion sådan: $f(x) = \frac{4}{x}, DM(f) = \{x | x \neq 0 \wedge x \in \mathbb{R}\}$. Altså $f(x)$ er defineret for alle x, hvor x ikke er 0 og x er et reel tal, ofte udelader vi $\mathbb{R}$, eller notere det en smule anderledes. I foregående notering var $\mathbb{R}$ og $DM(f)$ begge talmængder.

En mængde M er en samling af forskellige objekter m kaldet elementer af M [21]. Disse objekter kan være tal, koordinater, polynomier, bogstaver, tegn eller for den sags skyld mennesker. En mængde kan være ordnet eller uordnet, f.eks. mængden er $N_3 = \{1, 2, 3\}$ ordnet, da 2 er større end 1. Mængden af øjnfarver $\{blå, brun, rød\}$ er uordnet, da man ikke rigtig kan sætte dem op mod hinanden [20]. I denne rapport arbejdes der primært med ordnede mængder bestående af tal eller koordinater.

I tabel 1 [22] ses en liste over almindelig kendte mængder, til venstre har jeg skrevet det tegn der representere dem, til højre står der en definition af mængden. F.eks. består mængden af alle primtal af a, hvis a og b er naturlige tal, og a divideret med b ikke er et naturligt tal, samt b ikke er lig med hverken 1, 0 eller a.

Eksempler på kendte talmængder	
Naturlige tal	$\mathbb{N} = \{0, 1, 2, 3, 4 \ldots\}$
Heltal	$\mathbb{Z} = \{\ldots -2, -1, 0, 1, 2 \ldots\}$
Rationelle tal	$\mathbb{Q} = \{\frac{a}{b} \| a, b \in \mathbb{Z} \wedge b \neq 0\}$
Reelle tal	$\mathbb{R} = \{\sqrt{2}, e, \pi \ldots\}$
Komplekse tal	$\mathbb{C} = \{a + bi \| a, b \in \mathbb{R}\}$
Primtal	$\mathbb{P} = \{a \| a, b \in \mathbb{Z} \wedge \frac{a}{b} \notin \mathbb{Z} \wedge b \neq 1 \vee a \vee 0\}$

Tabel 1: Elementære talmængder

2.3 Gruppe teori

En gruppe $(G, +)$ er en mængde G og en operation $+$ som opfylder følgende [4], [5]:

- Indeslutning: Operationer mellem elementer i en gruppe giver et element i gruppen: $a + b \in G$ hvis $a, b \in G$.
- Associativitet: Operationen $+$ er associativ, dvs: $(a + b) + c = a + (b + c)$ for alle $a, b, c \in G$.
- Neutralt element: Gruppen har et neutralt element e sådan at $a + e = e + a = a$ for alle $a \in G$.
- Invers element: For et hvert element $a \in G$, findes der et element $b \in G$, sådan at $a + b = e$.

Hvis gruppens operation er kommutativ, dvs. at $a + b = b + a$ for alle $a, b \in G$, så er der tale om abelsk gruppe [4], [15]. I denne rapport arbejdes der med abelske grupper.

Et eksempel på en gruppe kunne f.eks. være $(\mathbb{Z}, +)$, som er alle heltal under addition. Denne gruppe opfylder kravene, fordi et helttal plus et andet heltal altid giver et heltal, altså den indesluttet. Under addition er rækkefølgen også ligegyldig, derfor opfylder den kravene om associativitet. Gruppen har også et neutralt element 0, da et helttal a plus 0 altid giver a. Sidst men ikke mindst vil der også eksistere et invers element, f.eks. vil $6 + (-6) = 0$, altså er -6 det inverse element til 6 i gruppen $(\mathbb{Z}, +)$.

2.4 Legeme teori

Et legeme $(F, +, \cdot)$ er en mængde F, en operation $+$ ofte kaldet addition og en anden operation $\cdot$ ofte kaldet multiplikation, der opfylder følgende [13]:

- Indeslutning: Opertioner mellem elementer i legemet giver et element i legemet, dvs: $a + b \in F$ og $a \cdot b \in F$ for alle elementer $a, b \in F$.
- Associativitet: Begge operationer i legemet er associative, $(a+b)+c = a+(b+c)$ og $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for alle $a, b, c \in F$.
- Neutrale elementer: Der eksister et neutralt element for addition og et neutralt element multiplikation.
- Inverse elementer: Der eksistere et additiv og et multiplikativ invers elementer for alle elementer.
- Hiraki: Multiplikation er højere rangerende end addition, eksempel: $a \cdot (b + c) = a \cdot b + a \cdot c$ for alle elementer $a, b, c \in F$.

Ifølge overstående definition er $(F, +)$ og $(F \backslash \{0\}, \cdot)$ begge abelske grupper [5]. $F \backslash \{0\}$ betyder F, uden elementet 0. Et eksempel på legeme kunne være $\mathbb{Q}$, alle de rationelle tal, under almindelig addition og multiplikation [13].

3 Aritmetik i legemet F_p

Forklaring af hvordan man foretager aritmetik i legemet F_p.

3.1 Legemet F_p

Legemet F_p består som vist i (1) af alle naturlige tal (inklusiv 0) op til primtallet p, der lukker legemet. Et eksempel på dette legeme er F_7 (2), med primtallet 7. F_7 og F_p er uanset hvilket primtal p er et legeme, i de næste afsnit vil jeg gennemgå de forskellige operationer og bevise denne påstand.

$$F_p = \{a | a < p \wedge a \in \mathbb{N} \wedge p \in \mathbb{P}\} \quad (1)$$

$$F_7 = \{a | a < 7 \wedge a \in \mathbb{N}\} = \{0, 1, 2, 3, 4, 5, 6\} \quad (2)$$

3.2 Addition

Addition i F_p foregår under modulo p (noteret: mod p). Hvis læseren ikke er bekendt med mod, kan jeg anbefale at man kigger i vedlagt appendiks C: Divisions sætningen, der giver en kort intro til heltals division, oprindelsen af mod, samt regneregler for modulo. Addition, af elementet a og elementet b til element $(a + b) \in F_p$, foretages sådan for alle $a, b \in F_p$:

$$(a + b) = a + b \bmod p \quad (3)$$

$$a + (b + c \bmod p) \bmod p = (a + b \bmod p) + c \bmod p \quad (4)$$

$$a + 0 \bmod p = a + 0 = a \quad (5)$$

Ifølge divisions sætningen skal resten r altid være mindre end dividenten. Når vi tager resten af additionen $a+b$ ved division af p skal denne rest være mindre end p, hvorved den også må være repræsenteret som et element i legemet. Således er kravet om indesluttehed ved addition opfyldt. På baggrund af regnereglerne for mod, må vi også sige at kravet om associativitet under addition også opfyldt, som vist i (4). I F_p er $e = 0$ det neutrale element under addition, da et tal a mindre en p under modulo p giver a, som i (5).

3.3 Subtraktion

For at F_p kan være et legeme skal, der findes et invers element $(-a)$ for et hvert element a, sådan at $a+(-a) = e = 0$. Da addition foregår under modulo p, må et sådan element kunne findes hvis additionen de to elementer a og $(-a)$ giver p, da $p \bmod p = 0$. Derfor må det additiv inverse element $(-a)$ til et hvilket somhelst element a kunne udtrykkes ved:

$$(-a) = p - a \quad (6)$$

$$a + (-a) \bmod p = a + p - a \bmod p = p \bmod p = 0 \quad (7)$$

Da a er et element af legemet F_p, må $a < p$, hvorved $(-a)$ også må være mindre end p, men større end 0 (såfremt $a \neq 0$). Hvorved $(-a) \in F_p$, altså $(-a)$ er også et element i legemet F_p. Som det ses i (7) må $(-a)$, fundet ved (6) være det inverse element til a i legemet F_p. Hvorved alle elementer i F_p har et additivt invers element.

Da der ikke findes noget subtraktions operation i et legeme, benytter man addition med det additive inverse element istedet [1]. Derfor sorter afsnittet om legemets additive inverse element under subtraktion. Det kan dog være en smule irriterende, først at skulle finde det inverse element før man kan subtraktere to elementer. Men som vist nedenunder kan det hele også regnes på en gang.

$$a + (-b) \bmod p = a + p - b \bmod p = a - b \bmod p \quad (8)$$

3.4 Multiplikation

Multiplikation i F_p foregår som vist på (9), på samme måde som addition, dvs. også under modulo p. På samme måde som ved addition kan man sige, at resten ved division af produktet af a og b må være mindre end dividenten p, hvorved resultatet ab også er et element i F_p.

$$ab = a \cdot b \bmod p \tag{9}$$

$$a \cdot (b \cdot c \bmod p) \bmod p = (a \cdot b \bmod p) \cdot c \bmod p \tag{10}$$

$$a \cdot 1 \bmod p = a \bmod p = a \tag{11}$$

Som resultat af regnereglerne for mod (appendiks C.1) må associativiteten som vist i (10) også være iorden. Derudover er det neutrale element under multiplikation $e = 1$. Der skeldes kraftigt mellem additivt neutralt element og multiplikativt neutralt element, selvom de har samme respektive egenskaber og jeg bruger samme variabel til at repræsentere dem. Det multiplikative neutrale element er vist ved (11).

3.5 Division

Ligesom ved subtraktion findes der ingen operation for at dividere i et legeme, derfor benyttes det multiplikative inverse element. Man kan forestille sig at der til er hvert element $a \in F_p$ findes et element $a^{-1} \in F_p$ sådan at $a \cdot a^{-1} \bmod p = 1$. Men det er straks vanskeligere at finde eller bare bevise eksistensen af et sådant element. F.eks. er 2 det multiplikative inverse element til 4 i legemet F_7, fordi $2 \cdot 4 \bmod 7 = 8 \bmod 7 = 1$, da resten ved division af 8 med 7 er 1.

Et multiplikativ inverse element til at andet element, kan findes med the Extended Euclidean algorithm [3], [12], [19]. Denne algoritme tager to tal som input a og b og finder så en ligningen $a \cdot x + b \cdot y = d$, hvor d er den største fællesdeler for a og b. Hvis vi skriver p istedet for b og får $d = 1$, må der findes et multiplicativ invers element til a, fordi ligningen ser sådan ud som i (12). Hvis vi tænker på hvordan multiplikation mellem 2 elementer foregår skrives det som i (13). Omskriver vi dette med regnereglerne for modulo (Appendiks C.1), finder vi at (14). Så er det jo logisk nok at (15), da $p \cdot y | p$ (altså $p \cdot y$ går op i p) hvilket betyder at resten er 0. Dette sætter vi ind i (14) hvorved vi får en ligningn udtryk som (16). Altså må x være det multiplikative inverse element til a i legemet F_p.

$$a \cdot x + p \cdot y = d = 1 \tag{12}$$

$$a \cdot x + p \cdot y \bmod p = d = 1 \tag{13}$$

$$(a \cdot x \bmod p) + (p \cdot y \bmod p) \bmod p = d = 1 \tag{14}$$

$$p \cdot y \bmod p = 0 \tag{15}$$

$$a \cdot x \bmod p = d = 1 \tag{16}$$

Således kan the Extended Euclidean algorithm benyttes til at finde multiplikative inverse elementer med. Således blev det også bevist, at der kun findes et multiplikativ invers element til a mod b hvis den største fællesdeler til a og b er 1. Hvis der lyser en lampe hos læseren nu, er det fordi man har fundet ud af hvorfor der netop skal et primtal med til at skabe F_p. Det er nemlig fordi, den største fællesdeler til et hvilket som heltal og så et primtal altid er 1. Hvorved det lykkedes at bevise det sidste krav for at F_p er et legeme, nemlig at der findes multiplikative inverse elementer til alle elementer i F_p.

3.5.1 The Extended Euclidean algorithm

The Extended Euclidean algorithm som præsenteret tidligere kan ses i algoritme 1. Dette er en mindre omskrivning af en pseudo implementering fra [12] og er tekniske set mangen til algoritme 2.107 i [3]. Det er den jeg har benyttet i min implementering. Den er ret simpel at udføre og jeg tror ikke notationen behøver yderlig forklaring, udover at $\lfloor x \rfloor$ betyder at man runder x ned.

Algoritme 1 The Extended Euclidean algorithm

Input: $a, b \in \mathbb{N}$
Output: $d = a \cdot x_1 + b \cdot y_1$

```
1:  x_0 ← 0
2:  y_0 ← 1
3:  x_1 ← 1
4:  y_1 ← 0
5:  mens b ≠ 0 udfør
6:      q ← ⌊a/b⌋
7:      r ← a mod b
8:      a ← b
9:      temp ← x_0
10:     x_0 ← x_1 − q · x_0
11:     x_1 ← temp
12:     temp ← y_0
13:     y_0 ← y_1 − q · y_0
14:     y_1 ← temp
15:     b ← r
16: slut mens
```

3.6 Modular eksponentiation

Når man kan multiplicer i legemet F_p må man også kunne eksponenere. Dette kan man jo gøre ved at multiplicere et element med sig selv et antal gange. F.eks. elementet $4 \in F_7$ opløftet i eksponenten 3 giver $4^3 \bmod 7 = 4 \cdot 4 \cdot 4 \bmod 7 = 64 \bmod 7 = 1$. Hvis eksponenten er meget høj vil det tage lang tid at multipilere elementet med sig selv. Derfor findes der en algoritme til at regne eksponentiation meget hurtigere, se algoritme 2, algoritme 2.143 i [3].

Algoritme 2 Modular eksponentiations algoritme

Input: $a \in F_p$ og eksponenten $k \in F_p$ en binær representation af $k = \sum_{i=0}^{\lfloor \log_2(k) \rfloor} k_i \cdot 2^i$
Output: $b = a^k \bmod p$

```
1:  b ← 1
2:  A ← a
3:  hvis k_0 = 1 så
4:      b ← a
5:  slut hvis
6:  for i = 1 til ⌊log_2(k)⌋ udfør
7:      A ← A^2 mod p
8:      hvis k_i = 1 så
9:          b ← A · b mod p
10:     slut hvis
11: slut for
```

Hvor $\lfloor x \rfloor$ betyder at man runde x ned, og $\log_2()$ er 2 tals logaritmen, sådan at $\lfloor \log_2(k) \rfloor + 1$ er længden af den binære repræsentationen af tallet k, altså må $\lfloor \log_2(k) \rfloor$ være det sidste bit. Så $k = \sum_{i=0}^{\lfloor \log_2(k) \rfloor} k_i \cdot 2^i$ betyder ganske enkelt af k_i er en binær repræsentation af k, hvor i peger på det enkelte bit. F.eks. $k = 9$, så er den binære repræsentation 1001 og $k_0 = 1$, $k_1 = 0$ og $k_3 = 4$. Læg mærke til at det første bit ikke er bit nummer 1, men bit nummer 0. Det forudsættes at læseren har grundlæggende forståelse af binære tal.

Hvis vi fortsætter eksemplet med $k = 9$, vil $b = a \cdot a^8 \bmod p = a^9 \bmod p$. Dette skyldes at b i linje 4 bliver sat til a, fordi $k_0 = 1$. Derefter vil A blive multipliceret med sig selv 3 gange, hvilket giver $A = a^{2^{2^2}} = a^8$, dette sker i linje 7. Til sidst i linje 9 bliver $b = A \cdot b \bmod p = a^8 \cdot a \bmod p = a^9 \bmod p$ fordi $k_3 = 1$. Dette er måske ikke betydeligt hurtigere end at multiplicere a med sig selv 9 gange, men havde eksponenten bestået 10-20 decimaler er denne algoritme betydeligt hurtigere.

En anden detalje der er værd at nævne, er at den modsatte operation, altså find k i $b = a^k \bmod p$ hvor a, b og p er kendte, er kendt som det diskrete logaritme problem [11]. Der findes altså ingen effektiv algoritme for at finde k. Man kan selvfølgelig prøve sig frem, ved at starte med 1, og arbejde sig op efter. Men hvis tallet er tilstrækkelig stort, kan dette tage et par årtusinder at løse problemet, også selvom man benytter nogle bedre algoritmer.

3.7 Kvadratrod modulo primtal

Når der findes et element $a \in F_p$, så må der også findes et element $a^2 = b$. Man kan selvfølgelig forholdsvist hurtigt finde elementet $b \in F_p$, med simpel multiplikation af a under modulo p: $b = a \cdot a \bmod p$. Omvendt kan man også finde a udfra b, dette kræver selvfølgelig at a eksister.

3.7.1 Legendre symbolet

Legendre symbolet er et nummerteoretisk koncept, der bruges til at bestemme om et givent tal er et kvadarttal modulo et givent primtal. Legendre symbolet ser sådan ud $\left(\frac{a}{p}\right)$, hvor a er et heltal og p er et primtal. Legendre symbolet kan have tre værdier, 0 hvis $a \bmod p = 0$, 1 hvis a er en kvadrattal modulo p, -1 hvis a ikke er et kvadrattal modulo p. Legendre symbolet kan beregnes sådan [17], [3]:

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p} \tag{17}$$

$$\left(\frac{a}{p}\right) - a^{(p-1)/2} \bmod p = 0 \tag{18}$$

$$\left(\frac{a}{p}\right) \bmod p = a^{(p-1)/2} \bmod p \tag{19}$$

Det første udtryk (17) er beskrevet med kongruens relationen, noget som jeg bevidst har forsøgt ikke at introducere i denne rapport (Se evt. appendiks C.3). Men ignorer det blot, og se på de 2 andre omskrivninger.

3.7.2 Kvadratrods algoritme i F_p

Man kan beregne kvadratrødder i F_p med algoritme 3, [3]. På linje 3, sættes b til at være et tilfældigt heltal mellem 1 og $p-1$, dette gør vi indtil legendre symbolet $\left(\frac{b}{p}\right) = -1$. På linje 5, finder vi s og t der opfylder $p-1 = 2^s \cdot t$, dette kan man gøre ved gentagende division med 2. I min implementering har jeg gjort det ved at skanne $p-1$ igennem efter det første høje bit bagfra. Denne metode kræver en smule kendskab til binære tal, se evt. kommentarer til min implementering i filen "numbertheory.c". Udregningerne i linje 6 og 7, benytter sig af algoritme 2. Alle operationerne på linje 9 er selvfølgelig legemeoperationer mellem elementer i F_p, og skal derfor foregår med de algoritmer og metoder beskrevet ovenfor.

Algoritme 3 Kvadratrods algoritme i F_p

Input: $a \in F_p$ og primtallet p
Output: r, sådan at $a = r^2 \bmod p$

1: **hvis** $\left(\frac{a}{p}\right) \neq -1$ **så**
2: **gentag**
3: $b \in \{a | a < p-1 \wedge a \in \mathbb{N}\}$
4: **indtil** $\left(\frac{a}{p}\right) = -1$
5: Find s og t sådan at $p-1 = 2^s \cdot t$, hvor t er ulig.
6: $c \leftarrow b^t \bmod p$
7: $r \leftarrow a^{(t+1)/2} \bmod p$
8: **for** $i = 1$ til $s-1$ **udfør**
9: $d \leftarrow (r^2 \cdot a^{-}1)^{2^{s-i-1}} \bmod p$
10: **hvis** d = p-1 **så**
11: $r \leftarrow r \cdot c \bmod p$
12: **slut hvis**
13: $c \leftarrow c^2 \bmod p$
14: **slut for**
15: $r \leftarrow r \vee -r \bmod p$
16: **ellers**
17: $r \leftarrow \varnothing$
18: **slut hvis**

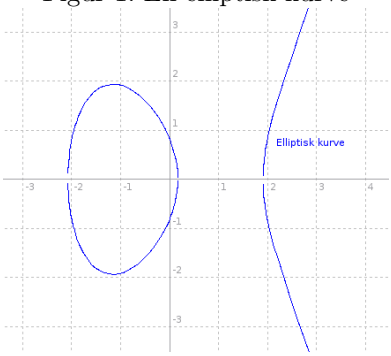
4 Aritmetik i en elliptisk kurve gruppe over F_p

Forklaring af en hvad elliptisk kurve gruppe over F_p er, samt introduktion til aritmetik i en sådan gruppe.

4.1 Elliptisk kurve gruppe over F_p

En elliptisk kurve kan udtrykkes som i (20), hvis den bliver plottet ser den ud som på figur 1. Hvis man tager mængden af koordinater, der opfylder denne ligning, vil man få en talmængde der ser ud som (21). Såfremt $4 \cdot a^3 + 27 \cdot b^2 \neq 0$ og med specielle operationer er denne talmængde faktisk en gruppe, en elliptisk kurve gruppe. Den før omtalte elliptiske kurve gruppe er sådan set over $\mathbb{R}$ alle reelle tal, fordi plottet bevæger sig over alle tal. Jeg ville gerne have gemmegået en elliptisk kurve gruppe over $\mathbb{R}$, men dette ligger uden for rammer for denne rapport og langt uden for størrelsen af rapporten.

Figur 1: En elliptisk kurve



$$y^2 = x^3 + a \cdot x + b \tag{20}$$
$$E = \{(x, y) | y^2 = x^3 + a \cdot x + b\} \tag{21}$$

Indenfor computerverden er man ikke specielt glad for at arbejde med irrationelle tal, uendelige brøker og mærkelige reelle tal [1]. Det ville være ubehageligt hvis krypterede beskeder mistede bogstaver pga. afrundinger i krypterings algoritmen. Derfor arbejder vi med en elliptisk kurve gruppe over F_p istedet. Dette gør vi ved at benytte de aritmetik regler der gælder for F_p, hvilket giver en elliptisk kurve ligning der ser sådan ud (22). Den elliptiske kurve gruppe ser nu sådan ud (23), der er et krav at $a, b \in F_p$ og at de opfylder $4 \cdot a^3 + 27 \cdot b^2 \bmod p \neq 0$.

$$y^2 \bmod p = x^3 + a \cdot x + b \bmod p \tag{22}$$
$$E_p = \{(x, y) | y^2 \bmod p = x^3 + a \cdot x + b \bmod p \wedge x, y \in F_p\} \tag{23}$$
$$E_7 = \{(x, y) | y^2 \bmod p = x^3 + 2 \cdot x + 3 \bmod 7 \wedge x, y \in F_7\} \tag{24}$$

Et eksempel på en elliptisk kurve gruppe ville være $a = 2$ og $b = 3$ over F_7. a og b opfylder kravet $4 \cdot 2^3 + 27 \cdot 3^2 \bmod 7 = 2 \neq 0$. Den elliptiske kurve gruppe indeholder alle koordinater der opfylder (24), hvilket er: $E_7 = \{(2, 6)\ (3, 6)\ (6, 0)\}$, altså E_7 har 3 koordinater, noteret sådan $|E_7| = 3$.

Sidst men ikke mindst bør det nævnes at elliptisk kurve gruppe, udover at indeholde alle koordinater der opfylder ligningen, også indeholder O, punktet ved uendeligheden. Det er ikke et kunstigt element som man har bestemt skal være gruppens neutral element [8].

4.2 Punkt addition

Punkt addition i en elliptisk kurve gruppe foregår på forskellige metoder alt efter hvilke punkter man ligger sammen. Forestil dig additions operationen som en gaffel funktion. Jeg gennemgår her operationerne og undtagelserne, der til sammen danner operationen for punkt addition i en elliptisk kurve gruppe over F_p.

4.2.1 Distinkt punkt addition

Hvis vi ligger to forskellige punkter sammen gøres det med distinkt punkt addition som foregår sådan [8]:

$$
\begin{aligned}
(x_1, y_1) + (x_2, y_2) &= (x_r, y_r) && (25)\\
s &= \frac{y_1 - y_2}{x_1 - x_2} \bmod p && (26)\\
x_r &= s^2 - x_1 - x_2 \bmod p && (27)\\
y_r &= s \cdot (x_1 - x_r) - y_1 \bmod p && (28)\\
& && (29)
\end{aligned}
$$

Hvor punkt et P_1 repræsenteres af (x_1, y_1), punkt to P_2 af (x_2, y_2) og (x_r, y_r) repræsentere selvfølgelig resultatet P_r. Som det ligger i navnet "distinkt punkt addition" er det et krav til denne operation at $P_1 \neq P_2$ og at $P_1 \neq -P_2$. Til det skal det også nævnes at inverse element til $P_1 = (x_1, y_1)$ er $-P_1 = (x_1, -y_1 \bmod p) = (x_1, p - y_1)$.

4.2.2 Punkt fordobling

Hvis $P_1 = P_2$ skal man lægge dem sammen ved at fordoble P_1, man kan nemlig ikke benytte distinkt punkt additions operationen som præsenteret overfor. Man kan fordoble et punkt sådan [8]:

$$
\begin{aligned}
(x_1, y_1) + (x_1, y_1) &= (x_r, y_r) && (30)\\
s &= \frac{3 \cdot x_1^2 + a}{y_1} \bmod p && (31)\\
x_r &= s^2 - 2 \cdot x_1 \bmod p && (32)\\
y_r &= s \cdot (x_1 - x_r) - y_1 \bmod p && (33)\\
& && (34)
\end{aligned}
$$

Sammen med "distinkt punkt addition" danner denne operation grundlag for punkt addition, for hvis $P_1 \neq P_2$ benytter man "distinkt punkt addition", hvis $P_1 = P_2$ benytter man punkt fordobling og hvis $P_1 = -P_2$ er resultatet O, punktet ved uendeligheden. Da O er gruppens neutrale element gælder $P + O = P$ selvfølgelig også.

4.3 Punkt skalar multiplikation

Der findes som sagt ikke multiplikation i en gruppe. Men man kan godt lægge et punkt sammen med sig selv $P + P = 2P$ og dermed fordoble det. Man kan også ligge det sammen med sig selv 3 gange, 4 gange eller 28 gange. Det er der ikke noget problem i. Punkt skalar multiplikation er altså at ligge et punkt sammen med sig selv et antal gange. Dette kan gøres ved at addere det $2P + P + P + P = 5P$, dette kræver først en fordoblings operation og derefter fire distinktive punkt additions operationer. Men man kunne også gøre det med 2 punkt fordoblings operationer og en distinktiv punkt additions operation: $P + P + P + P + P = 2(2P) + P = 5P$.

Algoritme 4 udfører punkt skalar multiplikation på en rimelig effektiv metode. Det er muligt at gøre dette endnu mere effektivt [6], men sådanne algoritmer ligger langt udenfor rammerne af denne rapport. Det skarpe øje vil se, at denne algoritme har en slående lighed med modular eksponentiations algoritmen (algoritme 2).

Algoritme 4 Punkt skalar multiplikation

Input: $P \in E_p$ og faktoren $k \in F_p$ en binær repræsentation af $k = \sum_{i=0}^{\lfloor \log_2(k) \rfloor} k_i \cdot 2^i$
Output: $R = kP = P + P + P + \ldots$
1: $R \leftarrow O$ {Punktet ved uendeligheden}
2: **for** $i = 0$ til $\lfloor \log_2(k) \rfloor$ **udfør**
3: $P \leftarrow P + P$
4: **hvis** $k_i = 1$ **så**
5: $R \leftarrow R + P$
6: **slut hvis**
7: **slut for**

I linje 1 R sættes til punktet ved uendeligheden O, fordi $R + P$, i linje 5, første gang bare skal give P, hvilket det vil når R indeholder gruppens neutrale element.

4.3.1 Det elliptisk kurve diskrete logaritme problem

Lige som ved modular eksponentiations algoritmen, findes der ikke nogen effektiv algoritme til at reversere punkt skalar multiplikation. Dette kaldes det elliptisk kurve diskrete logaritme problem [1]. Ligesom det diskrete logaritme problem, findes der flere angreb på dette problem. Det er også muligt at finde k i $Q = kP$ ved gentagende addition af P, hvorved k må være antallet af gange vi gentog additionen. Dette vil tage lang tid, specielt hvis parameterne der bestemmer E_p er store, f.eks. primtallet, a og b.

Certicom har en ECC challenge hvor du kan få 100.000$, hvis du kan bryde det elliptisk kurve diskrete logaritme problem i en kurve på 358 bit [1]. De har også sat priser på en række mindre kurver, men indtil videre er der ingen kurver over 100 bit der er blevet knækket. Beregningstiden stiger da også enormt når der ligges ekstra bit til, eller ændres lidt ved kurvens egenskaber. Bitlængden fortæller hvor store tallene er.

5 Elliptisk Kurve Digital Signatur Algoritme

Forklaring af ECDSA en digital signatur algoritme bygget op omkring det elliptisk kurve diskrete logaritme problem.

5.1 Domæne parametre

ECDSA er bygget op omkring en elliptisk kurve gruppe, som tidligere nævnt er denne gruppe afhængig af nogle parametre (Ansnit 4.1). Helt konkret: p, a, b, G, n, h, disse kaldes for domæne parametre, man vælger dem udfra krav om sikkerhed og operations hastighed. p er som tidligere det primtal der danner gruppen F_p, som den elliptiske kurve er dannet over. a og b er de 2 parametre der indgår i den ellitiske kurve ligning (22). G er generator-punktet i kurven, et punkt valgt til kryptografiske operationer. n er kurvens orden, et tal der opfylder ligningen $h = \frac{|E_p|}{n}$, hvor $h \leq 4$, jo mindre jo bedre. $|E_p|$ er antallet af punkter på kurven [8], [6]. Forklaring af hvad G, n og h skal bruges til kommer senere.

Disse domæne parametre er standardiserede og kan slåes op i [10] og [2], samt ANSI X9.62:2005 og IEEE 1363-2000. Modtager og afsender i et asymmetriske signatur system baseret på ECDSA, må nødvendigvis have samme domæne parametre for at kunne arbejde sammen, derfor er disse standardiserede. Det er dog muligt at generer sine egen domæne parametre, men dette ligger uden for rammerne af dette projekt. Der er nemlig rigtig mange ting man skal tage hensyn til, for sikre en god operations hastighed og høj sikkerhed.

5.2 Private nøgle

Den private nøgle d i ECDSA er et tal mellem 0 og $n-1$, der bruges til at genere signaturer med. I min implementering benytter jeg $d = sha1(pass) \bmod n$, hvor $sha1()$ er en sha1 hash implementering, $pass$ er brugerens kodeord. Sådan kan man benytte en tekst som private nøgle. Det vigtigste er at det er tal der er svært at gætte [6].

5.3 Offentlig nøgle generering

Den offentlig nøgle Q, genereres ved punkt skalar multiplikation af G med faktoren d (Algoritme 4). G er et domæne parameter, og d er den private nøgle. På grund af elliptisk kurve diskrete logaritme problem, kan man ikke finde d, selvom man kender G og Q [1].

Hvis man prøver ved gentagende addition af G, indtil man får Q, bliver man aldrig færdig. Det tog eksempelvis 13 sekunder at tjekke 10^6 faktorer i secp112r1 kurven, der har et primtal på ca. $4,5 \cdot 10^{33}$. Nu lyder 13 sekunder ikke af så meget, men hvis man skal tjekke alle faktorer mellem 0 og n i kurven, som også er ca. $4,5 \cdot 10^{33}$, siger min hoved regning at det vil tage ca. $4,5 \cdot 10^{27}$ gange så lang tid, hvilket svarer til $1,8 \cdot 10^{21}$ år. Med en anslået levetid for solen på $5 \cdot 10^9$ år, skal slags udregning ikke foregå i dette solsystem. Den modsatte vej tager det mig kun ca. 1 sekundt at foretage 500 skalar multiplikationer fra bunden.

5.3.1 Punkt kompression

Da alle elementer i gruppen E_p er koordinater (x, y) med en relation der hedder $y^2 = x^3 + a \cdot x + b$, behøver man ikke oplyse både x og y for at kunne bestemme et punkt. Man kan faktisk nøjes med at vide om y er lig eller ulig og x. Kvadratroden modulo p har altid 2 løsninger: $r \vee p - r$ (Se algoritme 3), derfor skal man altså både kende x og hvorvidt y er lig eller ulig. Vi kan altså beregne y med følgende formel (35), såfremt vi kender x og ved om y skal være lige eller ulig.

$$y = \pm\sqrt{x^3 + a \cdot x + b} \bmod p \tag{35}$$

Ifølge [9] består et komprimerede punkt af en hexadecimal repræsentation af x prefikset med "02" eller "03" hvis y er henholdvis lig eller ulig. På denne måde kan en offentlig nøgle komprimeres til kun at være en hexadecimal streng.

5.4 Signatur generering

En ECDSA signatur består af to tal r og s, begge skal bevares for at en besked kan verificeres. En signatur kan genereres med algoritme 5 [8], [6]. Denne algoritme kræver en forklaring, da den tager udgangspunkt i mange af de operationer, der tidligere er gennemgået i denne rapport. I linje 2 sættes k til at være et tilfældig heltal mellem 1 og $n-1$. Det vigtigt at denne tilfældighed er helt tilfældig, og ikke bare pseudo-tilfældig, hvis dette tal kan findes kan den private nøgle brydes. I min implementering har jeg til dette formål hentet data fra "/dev/random" tilfældighedsgeneratoren på et Linux systemer, der henter tilfældighed ud fra signalstøj på hardwaren.

I linje 3, foretages en skalar multiplikation af G med faktoren k. I linje 6, omsættes meddelelsen m til et tal e, med en hash algoritme som f.eks. sha1. Afhængig af hash algoritme skal e forkortes da den ikke må være længere end n. I linje 7, foretages der først multiplikation og addition af elementer i F_p, disse operationer foregår altså under modulo p. Derudover regnes den multiplikative inverse til k ud, med algoritme 1. Hvis $s = 0$ startes udregningen forfra, med en anden k, ellers er signaturen generet.

Algoritme 5 Signatur generation

Input: domæne parametre (p, a, b, G, n, h), private nøgle d, meddelelse m
Output: Signatur (r, s)

1: **gentag**
2: $k \in \{a | a < n-1 \wedge a \in \mathbb{N}\}$
3: $(x_1, y_1) \leftarrow kG$
4: $r \leftarrow x_1 \bmod n$
5: **indtil** $r \neq 0$
6: $e \leftarrow HASH(m)$
7: $s \leftarrow (e + d \cdot r) \cdot k^{-1} \bmod n$
8: **hvis** $s = 0$ **så**
9: **goto** 1
10: **slut hvis**

5.5 Signatur verifikation

En ECDSA signatur består af to tal r og s, hvis man har en sådan signatur, en medelelse m, et sæt domæne parametre og en offentlig nøgle Q, kan man verificere om medelelsen blev underskrevet med den samme private nøgle som blev brugt til at genere den offentlige nøgle med. Dette kan gøres med algoritme 6. På linje 1, tjekkes det om r og s er et heltal mellem 1 og $n-1$. På linje 2 oversættes meddelelsen m til et tal, med en hash algoritme, f.eks. sha1, dette skal være den samme algoritme som der er blevet benyttet til generering af signaturen i algoritme 5. I linje 6 foretages to skalar multiplikationer og en punkt addition. Resten burde give sig selv, f.eks. i linje 3 der logisk nok er beregningen af den multiplikative inverse modulo n, med algoritme 1.

5.5.1 Matematisk bevis af meddelelses integritet

Når vi går en underskrevet meddelses kender vi signaturen (36) og (37) og meddelelsen m, hvorved vi også kender hashsummen af m (38). Sidst men ikke mindst kender vi domæne parameterne (p, a, b, G, n, h) og den offentlige nøgle (39), fra den person der har underskrevet meddelelsen.

$$r = [k \cdot G]_x \bmod n \tag{36}$$
$$s = (e + d \cdot r) \cdot k^{-1} \bmod n \tag{37}$$
$$e = hash(m) \tag{38}$$
$$Q = d \cdot G \tag{39}$$

Algoritme 6 Signatur verifikation

Input: domæne parametre (p, a, b, G, n, h), private nøgle d, meddelelse m, offentlig nøgle Q, signatur (r, s)

Output: $R \in \{\text{“gyldig”}, \text{“ugyldig”}\}$

1: **hvis** $r, s \in \{a | a < n - 1 \wedge a \in \mathbb{N}\}$ **så**
2: $e \leftarrow HASH(m)$
3: $w \leftarrow s^{-1} \bmod n$
4: $u_1 \leftarrow e \cdot w \bmod n$
5: $u_2 \leftarrow r \cdot w \bmod n$
6: $(x_1, y_1) \leftarrow u_1 \cdot G + u_2 \cdot Q$
7: $v \leftarrow x_1 \bmod n$
8: **hvis** $(x_1, y_1) \neq O \wedge v = r$ **så**
9: $R \leftarrow$ “gyldig”
10: **ellers**
11: $R \leftarrow$ “ugyldig”
12: **slut hvis**
13: **ellers**
14: $R \leftarrow$ “ugyldig”
15: **slut hvis**

Ud fra disse kan vi beregne følgende værdier. Disse er altså også kendte:

$$w = s^{-1} \bmod n \tag{40}$$
$$u_1 = e \cdot w \bmod n = e \cdot s^{-1} \bmod n \tag{41}$$
$$u_1 = r \cdot w \bmod n = r \cdot s^{-1} \bmod n \tag{42}$$

For at den til sendte meddelelse har en gyldig signatur, skal vi bevise at (43), dette kan vi imidlertid ikke da vi ikke kender k og d. Vi kender r, men pga. det elliptisk kurve diskrete logaritme problem kan vi ikke bruge den til at finde k. Derfor omskriver vi til (44), dernæst omskriver lidt mere og skifter konstanterne ud med dem vi har bestemt ovenfor, sådan at vi får (46). Da vi som sagt ikke kender k og d kan vi stadig ikke undersøge om (46) er tilfældet, og om signaturen dermed er gyldig. Men vi kender en matematisk relation på k, nemlig $k \cdot G$, i form af r. r er godt nok kun x koordinatet til punktet $k \cdot G$, men da der kun er 2 mulige y værdier kan vi godt sige at vi kender punktet. Derfor punkt skalar multiplicere vi G på begge sider (47). Nu kan vi så omskrive til (48), herefter kan vi så bruge den offentlige nøgle Q, istedet for $d \cdot G$ i (49). Hvorved udtrykket kan evalueres og vi kan undersøge om x værdien er den samme som r. Hvis dette er tilfældet er signaturen gyldig. Sådan benytter man altså en ikke reversibel matematisk relation til at lave digital signatur med.

$$s = (e + d \cdot r) \cdot k^{-1} \bmod n \tag{43}$$
$$\Updownarrow$$
$$k = (e + d \cdot r) \cdot s^{-1} \bmod n \tag{44}$$
$$= e \cdot s^{-1} + d \cdot r \cdot s^{-1} \bmod n \tag{45}$$
$$= u_1 + u_2 \cdot d \bmod n \tag{46}$$
$$\Updownarrow$$
$$k \cdot G = (u_1 + u_2 \cdot d \bmod n) \cdot G \tag{47}$$
$$= u_1 \cdot G + (u_2 \cdot d \bmod n) \cdot G \tag{48}$$
$$= u_1 \cdot G + u_2 \cdot Q \tag{49}$$
$$r = [u_1 \cdot G + u_2 \cdot Q]_x \tag{50}$$

6 Operations hastighed i praktisk implementering

Kort beskrivelse af min implementering, samt præsentation og vurdering af benchmark resultater og anvendelsesmuligheder.

6.1 Min implementering: SimpleECDSA

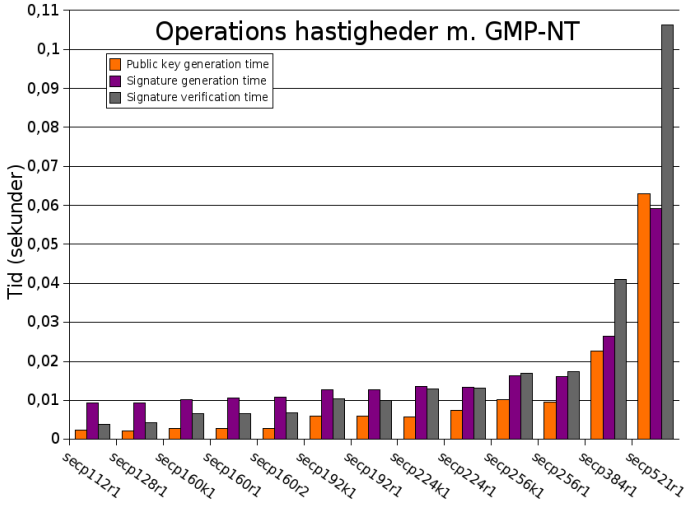
Jeg har til dette projekt udviklet en implementering af ECDSA i C. Kildekoden er printet og ligger appendiks D, med danske kommentarer der burde forklare hvordan den virker. I min implementering har jeg benyttet mig af det eksterne GNU Multiple Precision (GMP) biblioteket og SHA1 kode fra RFC3174 [7], alt derudover har jeg selv skrevet. Jeg mener at min implementering er af en sådan kvalitet at den kan bruges til at vurdere den praktiske operations hastighed af ECDSA. Da jeg har taget udgangspunkt i GMP, og skrevet koden i C, er det svært at forestille sig at udviklere i den kommercielle verden ville spilde tid på at skrive en implementering med yderligere optimeringer. Derfor mener jeg at min implementering giver et meget godt bud på en god operationshastighed, selv i en kommerciel løsning.

I SimpleECDSA har jeg implementeret alle de algoritmer, der er præsenteret i denne rapport. Nogle af disse algoritmer er også implementeret i GMP's nummerteori implementering. I "numbertheory.h" kan man vælge om SimpleECDSA skal komplieres med GMP's nummerteori implementering eller min nummer teori implementering. Ved gentagende benchmarks har jeg erfaret at GMP's nummer teori implementering er ca. 10 gange hurtigere end min egen. Det skal siges at GMP's er godt optimeret, og at den benytter forskellige algoritmer afhængig af tallenes størrelse [14]. Derfor er det også klart at GMP's implementering er betydeligt hurtigere end min egen. Jeg har valgt at præsentere resultaterne både med og uden GMP's nummer teori implementering. For mere information om hvilke nummer teori algoritmer GMP har se "numbertheory.h".

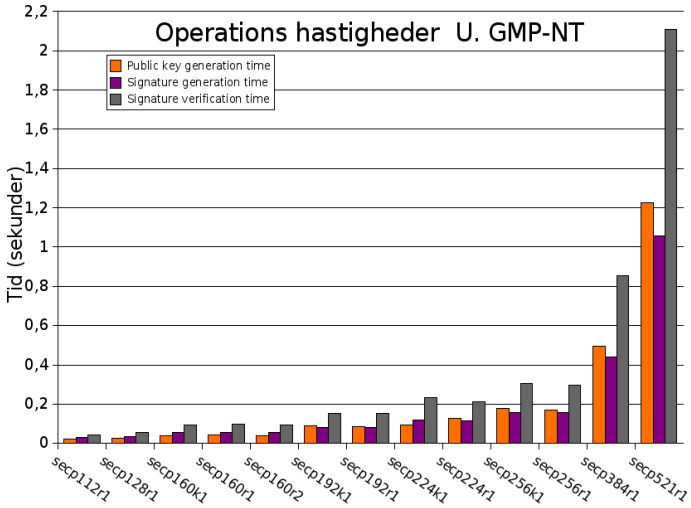
6.2 Benchmark resultater

Jeg har kørt nogle benckmarks på min implementering, som tidligere nævnt har jeg kørt dem både med og uden GMP's nummer teori implementering. I mine benchmarks har jeg holdt de forskellige kurver op mod hinanden. Det har jeg gjort ved at foretage tid på 100 offentlige nøgle genereringer, signeringer og verifikationer, med hver kurve. De tal der kom ud af det har jeg divideret med 100, så jeg har tider for en nøgle generering, en signering og en verifikation, for hver kurve. De originale tal kan ses i appendiks B. Nedenunder ses de to grafer de kom ud af sammenligningen. Figur 2 er resultater med GMP's nummer teori implementering. Mens figur 3 er resultaterne med min egen nummer teori algoritme.

Figur 2: Operations hastigheder med GMP nummer teori implementering



Figur 3: Operations hastigheder med egen nummer teori implementering



Naturligt nok er sikkerheden i disse kurver direkte bestemt af operations hastigheden. Vi se bort fra alle tænkelige angreb på det elliptiske kurve diskrete logaritme problem, da disse kurver fra Certicom er bygget til at modstå disse angreb [10]. Det ligger uden for denne rapport rammer at diskutere operations tiden af disse angreb på baggrund af kurvernes parametre. Kurverne har forskellige navne, det tal der står i dem angiver bitlængden af deres domæne parametre. Det er faktisk dette nummer der afgører hvor lang tid, og hvor sikker en given kurve er.

6.3 Anvendelsesmuligheder

I dag benyttes asymmetriske krypterings algoritmer når vi bruger netbank eller til digital signatur, og mange bruger det også i forbindelse med e-mails. Anvendelsesmulighederne for disse algoritmer bliver ikke mindre i fremtiden. De vil være relevante i alle sammenhænge hvor man ønsker sikker kommunikation.

For 5-10 år siden blev mobil telefoner hver mands eje, idag vil vi have smartphones. Med opbygning af GSM netværk er det ikke særligt vanskeligt at sender fra telefon nummere man ikke ejer. En sådan service har været tilgængelig i Sverige. Den slags kan godt være meget sjovt. Men i det øjeblik person følsome oplysninger og betalinger, giver den slags huller i populære kommunikations protokoller som f.eks. email og GSM, alvorlige problemer i forhold til sikkerhed. På den baggrund er der gode muligheder for ECDSA på mobile platformer. I det man kan regulere sikkerheds nivueat alt efter tilgængelig computer kraft.

Man kunne f.eks. forestille sig at en kurve med på 128 eller 160 bit ville være passende til mobiltelefoner og andre steder hvor man har begrænset computer kraft. Her ville en 512 bit kurve være alt for langsom. Til SMS'er vil 128 bit være acceptabelt, jeg ville ikke beskytte min netbank med det. Fordelene og ulemperne går jo unægteligt lige op, højre bitlængde, mere sikkerhed og langsommer operation. Fordelen ved højere sikkerhed er beskyttelse af data, langt ud i fremtiden, mens fordelen ved lille operations hastighed er at man kan implementere systemet på en mobil platform.

En detalje, der var med til at fange min interesse for ECDSA, er at man kan benytte tilfældige tal som privat nøgle. I mange andre asymmetriske signatur algoritmer skal man benytte et til med en specielt egenskab som privat nøgle, f.eks. et primtal. Men da dette ikke er tilfældet med ECDSA kan brugeren faktisk huske sin private nøgle i hovedet. Hvormed man faktisk ville kunne bruge ECDSA til identifikation i et distribueret netværk, dvs. et netværk uden central server. Det var i udviklingen af et sådant system jeg fangede interessen for ECDSA, daværende tidspunkt kunne jeg dog ikke være med på det matematiske niveau.

7 Konklusion

På baggrund af dette projekt kan man konkludere at abstrakt algebra er et stærkt værktøj til at opnå et højt abstraktions niveau. Da vi ved at skabe nogle kunstige strukturer med kunstige regne operationer og bestemte egenskaber kan vi regne på dem som var det almindelig algebra, og derved kan bevise ting der ellers ville være nærmest umuligt. Her tænker jeg f.eks. på beviset for medelelses integritet.

Jeg er også kommet frem til at sikkerheden og operations hastigheden hænger unægteligt sammmen. Derfor må man altid vurdere hvilket sikkerhedsniveau, der er ønskeligt alt efter hvor meget computer kraft man har tilgængelig. Sådan at man må vurdere sikkerhedsniveau op mod operations hastigheden.

Desuden er jeg kommet frem til at man kan udnytte matematiske problemer til at skabe en ireversabel matematik relation. Og at en sådan relation kan benyttes til at skabe anvendelige asymmetriske krypterings systemer med. Dette er f.eks. tilfældet det diskrete logaritme problem og det elliptisk kurve diskrete logaritme som forklaret i denne rapport.

Jeg kan kan også konkludere at der i fremtiden er gode anvendelses muligheder for ECDSA. Både indenfor allerede kendte og populære kommunikations protokoller, men også i fremtidige systemer der måske hviler mindre på centrale server strukturer.

A litteraturliste

[1] Online elliptic curve cryptography tutorial. URL http://www.certicom.com/index.php?action=ecc_tutorial,home.

[2] Recommended elliptic curves for federal government use. Rapport, National Institute of Standards and Technology, 1999. URL http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf.

[3] Paul C. van Oorschot og Scott A. Vanstone Alfred J. Menezes. *Handbook of Applied Cryptography*. CRC Press, 1996. ISBN 0-8493-8523-7. URL http://www.cacr.math.uwaterloo.ca/hac.

[4] K.R. Pearson Arthur Jones S.A. Morris. *Abstract Algebra and Famous Impossibilities*. Springer, 1991. ISBN 3-54097-661-2.

[5] Allan Clark. *Elements of Abstract Algebra*. Dover Publications Inc., 1985. ISBN 0-48664-725-0.

[6] Alfred Menezes og Scott Vanstone Darrel Hankerson. *Guide to Elliptic Curve Cryptography*. Springer. ISBN 0-387-95273-X.

[7] 3rd Paul E. Jones og Donald E. Eastlake. *US Secure Hash Algorithm 1 (SHA1)*. The Internet Society, 2001. ISBN 0-8493-8523-7. URL http://www.ietf.org/rfc/rfc3174.txt.

[8] Anoop MS. Elliptic curve cryptography – an implementation tutorial. Rapport, Tata Elxsi Ldt, Thiruvananthapuram, India, 2007. URL http://hosteddocs.ittoolbox.com/AN1.5.07.pdf.

[9] Certicom Research. Standards for efficient cryptography: Elliptic curve cryptography. Rapport, Certicom Research, 2000. URL http://www.secg.org/download/aid-385/sec1_final.pdf.

[10] Certicom Research. Standards for efficient cryptography: Recommended elliptic curve domain parameters. Rapport, Certicom Research, 2000. URL http://www.secg.org/download/aid-386/sec2_final.pdf.

[11] Wikipedia. Discrete logarithm — wikipedia, the free encyclopedia, 2007. [Online; accessed 11-December-2007]. URL http://en.wikipedia.org/w/index.php?title=Discrete_logarithm&oldid=177010906.

[12] Wikipedia. Extended euclidean algorithm — wikipedia, the free encyclopedia, 2007. [Online; accessed 10-December-2007]. URL http://en.wikipedia.org/w/index.php?title=Extended_Euclidean_algorithm&oldid=175494541.

[13] Wikipedia. Field (mathematics) — wikipedia, the free encyclopedia, 2007. [Online; accessed 10-December-2007]. URL http://en.wikipedia.org/w/index.php?title=Field_oldid=168964720.

[14] Wikipedia. Gnu multi-precision library — wikipedia, the free encyclopedia, 2007. [Online; accessed 14-December-2007]. URL http://en.wikipedia.org/w/index.php?title=GNU_Multi-Precision_Library&oldid=158002723.

[15] Wikipedia. Gruppe (matematik) — wikipedia, den frie encyklopædi, 2007. [Online; hentet 10-december-2007]. URL http://da.wikipedia.org/w/index.php?title=Gruppe_ldid=1750162.

[16] Wikipedia. Legeme (algebra) — wikipedia, den frie encyklopædi, 2007. [Online; hentet 10-december-2007]. URL http://da.wikipedia.org/w/index.php?title=Legeme_id=1775450.

[17] Wikipedia. Legendre symbol — wikipedia, the free encyclopedia, 2007. [Online; accessed 13-December-2007]. URL http://en.wikipedia.org/w/index.php?title=Legendre_symbol&oldid=170050380.

[18] Wikipedia. Modular arithmetic — wikipedia, the free encyclopedia, 2007. [Online; accessed 10-December-2007]. URL http://en.wikipedia.org/w/index.php?title=Modular_arithmetic&oldid=173746340.

[19] Wikipedia. Modular multiplicative inverse — wikipedia, the free encyclopedia, 2007. [Online; accessed 10-December-2007]. URL http://en.wikipedia.org/w/index.php?title=Modular_multiplicative_inverse&oldid=176388083.

[20] Wikipedia. Mængde — wikipedia, den frie encyklopædi, 2007. [Online; hentet 10-december-2007]. URL http://da.wikipedia.org/w/index.php?title=M1.

[21] Wikipedia. Set — wikipedia, the free encyclopedia, 2007. [Online; accessed 10-December-2007]. URL http://en.wikipedia.org/w/index.php?title=Set&oldid=176109238.

[22] Wikipedia. Tal — wikipedia, den frie encyklopædi, 2007. [Online; hentet 10-december-2007]. URL http://da.wikipedia.org/w/index.php?title=Tal&oldid=1756086.

B Benchmark resultater

Ubehandlede benchmark resultater, der er et linje skift CSV headeren, ellers er det rå CSV output.

B.1 100 operation på forskellige kurver, med GMP nummer teori

Resultatet af kommandoen "./SimpleECDSA –benchmark 100", det eneste jeg har gjort før jeg klippede det ind her var at tilføje et linje skift i CSV headeren, så den ikke længere er RFC4180 kompatibel. Følgende resultater beregnet mens programmet benyttede GMP's nummer teori implementering.

```
Curve, Public key generation time, Signature generation time,
Signature verification time, Operation time
secp112r1, 0.2400, 0.9400, 0.3900, 1.5700
secp128r1, 0.2200, 0.9400, 0.4200, 1.5800
secp160k1, 0.2800, 1.0300, 0.6700, 1.9800
secp160r1, 0.2900, 1.0600, 0.6600, 2.0100
secp160r2, 0.2900, 1.0900, 0.6900, 2.0700
secp192k1, 0.5900, 1.2700, 1.0400, 2.9000
secp192r1, 0.6000, 1.2800, 1.0000, 2.8800
secp224k1, 0.5800, 1.3600, 1.2900, 3.2300
secp224r1, 0.7500, 1.3400, 1.3200, 3.4100
secp256k1, 1.0300, 1.6300, 1.6900, 4.3500
secp256r1, 0.9600, 1.6100, 1.7300, 4.3000
secp384r1, 2.2600, 2.6500, 4.1000, 9.0100
secp521r1, 6.3000, 5.9200, 10.6300, 22.8500
```

B.2 100 operation på forskellige kurver, uden GMP nummer teori

Resultatet af kommandoen "./SimpleECDSA –benchmark 100", det eneste jeg har gjort før jeg klippede det ind her var at tilføje et linje skift i CSV headeren, så den ikke længere er RFC4180 kompatibel. Følgende resultater beregnet mens programmet ikke benyttede GMP's nummer teori implementering.

```
Curve, Public key generation time, Signature generation time,
Signature verification time, Operation time
secp112r1, 2.3300, 2.9700, 4.3300, 9.6300
secp128r1, 2.6000, 3.4900, 5.6700, 11.7600
secp160k1, 4.0300, 5.6600, 9.5900, 19.2800
secp160r1, 4.2000, 5.6400, 9.7200, 19.5600
secp160r2, 4.0200, 5.4900, 9.6100, 19.1200
secp192k1, 8.9200, 8.0900, 15.5000, 32.5100
secp192r1, 8.6800, 8.3500, 15.1600, 32.1900
secp224k1, 9.4700, 11.9100, 23.2600, 44.6400
secp224r1, 13.0100, 11.4700, 21.1400, 45.6200
secp256k1, 17.9500, 15.8200, 30.3300, 64.1000
secp256r1, 16.8300, 15.8700, 29.5400, 62.2400
secp384r1, 49.5500, 44.1400, 85.5500, 179.2400
secp521r1, 122.7200, 105.7800, 210.8300, 439.3300
```

C Divisions sætningen

Kort overordnet introduktion til heltalsdivision og regning med modulo.

Divisions sætningen siger at det gælder for alle naturlige tal a og b at der eksistere unikke positive heltal q og r sådan at $a = q \cdot b + r$, hvor $r < b$ [5]. Altså man kan udtrykke alle naturlige tal a med en dividente b, en kvotient q antal gange som b går op i a samt en rest r.

Overstående var heltals division med både kvotient og rest, dette kan deles i to operationer, heltalsdivision og modulo. Der noteres sådan $q = \lfloor a/b \rfloor$ og $r = a \bmod b$. Notationen $\lfloor a \rfloor$ indikere at a skal rundes ned. Derudover benytter man denne notation $a|b$ for at defineret at a går op i b, altså $r = 0$. Omvendt hvis a ikke går op i b, der notes sådan $a \nmid b$.

C.1 Regneregler for modulo

Man kan godt omskrive ligninger som modulo indgår[18], dette er faktisk også relativt nemt, og kan også godt gå en og blive meget nødvendigt hvis man skal arbejde meget med modulo.

$$a_1 \bmod n = b_1 \bmod n \tag{51}$$
$$a_2 \bmod n = b_2 \bmod n \tag{52}$$
$$\tag{53}$$

Såfremt (51) og (52) gælder kan man som tommelfinger regle over regne med at følgende er gældende:

$$(a_1 + a_2) \bmod n = (b_1 + b_2) \bmod n \tag{54}$$
$$(a_1 - a_2) \bmod n = (b_1 - b_2) \bmod n \tag{55}$$
$$(a_1 \cdot a_2) \bmod n = (b_1 \cdot b_2) \bmod n \tag{56}$$

C.2 Negativt tal modulo

Når man siger at $a \bmod b$ er resten ved heltals division af $\lfloor a/b \rfloor$ kunne man godt få det indtryk at $-a \bmod b$ giver et negativt tal, men dette er ikke tilfældet. $-10 \bmod 7$ giver nemlig ikke -3, det giver 4. Altså operationen $\bmod b$ giver altså et positivt tal, med mindre b er negativ. Derfor kan $-a \bmod b$ regnes som $b - a$, såfremt a er mindre end b.

C.3 Kongruens relation

En kongruens relationer noteres sådan $a \equiv b \pmod p$, det betyder at $a - b \bmod p = 0$, altså at $a - b$ går op i p. Man kan også fortolke det som $a \bmod p = b \bmod p$. Det kan ofte, og pga. af regnereglerne for modulo, være en fordel at arbejde med kongruence relationer istedet for heletiden at have regnereglerne i baghovedet. Jeg har dog i denne rapport forsøgt at undgå brugen af kongruens relationer, for at holder det hele enkelt. Derfor vil der også nogle steder i rapporten stå mod$p$ på begge sider af $=$, andre steder har jeg direkte omskrevet formler og algoritmer for at undgå brug af denne relation. Nedenunder ses regnereglerne for modulo udtryk med kongruens relationer.

$$(a_1 + a_2)n \equiv (b_1 + b_2) \pmod n \tag{57}$$
$$(a_1 - a_2)n \equiv (b_1 - b_2) \pmod n \tag{58}$$
$$(a_1 \cdot a_2) \equiv (b_1 \cdot b_2) \pmod n \tag{59}$$

D ECDSA implmentering i C

Kildekoden til min implementering kaldet SimpleECDSA kan findes på efterfølgende sider. Det er også min intention at ligge den ud på min blog: "http://jopsen.dk/blog/2007/12/elliptic-curve-digital-signature-algorithm". Her vil både rapporten og kildekoden kunne downloades. Kildekoden er frigivet under GNU GPLv3, og hvis man har et GNU/Linux system stående vil jeg da anbefale at man leger lidt med det. Har man en smule C erfaring kan det sikkert også forholdsvist let portes til en hvilken som helst anden platform, da jeg har forsøgt at holde min C kode kompatibel med standarterne.