SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Kryptografins Gränser: En Teoretisk och Praktisk Jämförelse av Moderna Algoritmer

av

Therese Bolid

2024 - No K19

Kryptografins Gränser: En Teoretisk och Praktisk Jämförelse av Moderna Algoritmer

Therese Bolid

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Olof Sisask

2024

Sammanfattning
Denna studie undersöker och jämför två krypteringsalgoritmer, RSA och ECC, med fokus på deras underliggande matematiska teorier, fördelar och praktiska tillämpningar. Genom en initial teoretisk analys belyser vi de matematiska satserna och metoderna som algoritmerna bygger på, och med en avslutande praktisk implementering i Python utvärderas algoritmernas prestanda gällande nyckelgenerering. Resultaten visar att medan RSA förblir en mycket välanvänd och effektiv metod, erbjuder ECC fördelar gällande säkerhet och prestanda.

Abstract
This study examines and compares the two widely used cryptography algorithms RSA and ECC, focusing on their underlying mathematical theories and practical applications. With an initial theoretical analysis of the underlying mathematical theorems and proofs, and a subsequent practical comparison in Python, we evaluate some of their benefits and downsides regarding effectiveness, security, and efficiency. We find that while RSA remains a widely used and effective method of encryption, ECC can offer superior security and efficiency.

Stort tack till min handledare Olof Sisask för all vägledning under arbetets gång. Tack även till min korrekturläsare tillika sambo Andris som har stöttat mig och kommit med kloka råd under hela arbetet.

Innehåll

1 Introduktion

1.1 Syfte och metod

Syftet med denna studie är att undersöka två av de största och mest använda krypteringsalgoritmerna genom tiderna. Frågor som kommer bemötas är:

Hur fungerar ECC och RSA, vilken teori bygger de på?

Vad finns det för olika för- och nackdelar mellan dessa och hur mäter de sig mot varandra?

Studien kommer bestå av två delar, en teoretisk och en praktisk. Först ges en bakgrund och förklaring till de olika algoritmerna som ska analyseras. Dessa jämförs sedan teoretiskt och slutsatser dras utifrån undersökningen och med ansats i frågeställningarna. Vidare följer en praktisk undersökning med hjälp av programmeringsspråket Python för att se om de teoretiska undersökningens resultat och slutsatser även återspeglas i praktiken.

Naturligtvis kommer inte detta kunna undersökas i praktiken lika långt som i teorin då dessa språk används för att kryptera känslig information storskaligt. Men med vissa avgränsningar kommer det gå att undersöka om det är möjligt att koppla teorin till praktiken genom att testa algoritmerna i Python. Framförallt kan det vara intressant om man är intresserad av relationen mellan matematik och data att även undersöka detta rent praktiskt.

När man pratar om kryptering exemplifieras själva metoden oftast med hjälp av tre olika personer: Alice, Bob och Eve, samt ett nyckelutbyte. Premissen går ut på att Alice och Bob vill utbyta hemlig information med varandra och Eve försöker komma åt denna information. Nyckeln är ett stycke information som används för att kryptera samt dekryptera meddelanden. Beroende på vilken algoritm man analyserar ser dessa nycklar olika ut, det kan handla om produkten av två primtal eller en punkt på en kurva. Även i denna uppsats kommer jag använda dessa tre namn som en vedertagen benämning på två parter som vill byta hemlig information och en tredje part som olovligen vill komma åt informationen. När vi talar om kryptering är det tal som krypteras till andra tal, dessa kan representera text i meddelanden men är i grunden tal. Meddelanden som ska krypteras benämns i uppsatsen som *klartext* och krypterade meddelanden som *kryptotext*.

1.1.1 Avgränsning

Det finns naturligtvis många olika krypteringsalgoritmer och vilka som är de "största och mest användbara" kan ha olika innebörd beroende på användarens behov och tillämpningsområde. Den här studien kommer att avgränsas till att

i huvudsak handla om Rivest-Shamir-Adleman (RSA) och Elliptic Curve Cryptography (ECC) med specifikt fokus på Elgamal.

Syftet till valet av just dessa två algoritmer i denna studie är att de sedan de implementerades har haft stor spridning och betydelse i hur kryptering tillämpas idag. Vidare kan en jämförelse mellan dessa vara intressant med anledning av att de tillkommit under olika tidsperioder och har skapats både i samspel och som en förlängning av varandra. Genom en undersökning av dessa algoritmer, vilka har många likheter men även signifikanta skillnader mellan varandra, kommer en del fördelar och nackdelar belysas.

1.2 Symmetrisk och Asymmetrisk kryptering

Under sent 1970-tal har krypteringsalgoritmer av olika slag redan funnits ett tag men kryptering ska snart komma att förändras och nya modernare krypteringsmetoder ska komma att utvecklas.

De första krypteringsmetoderna byggde på så kallad symmetrisk kryptering. Det går ut på att det finns en nyckel k som båda parter känner till således har både Alice och Bob samma nyckel men den hålls dold från Eve. Denna nyckel används både till att kryptera meddelanden samt att dekryptera meddelanden. Själva krypteringen kan ses som en funktion

$$e : K \times M \to C,$$

vars definitionsmängd $K \times M$ är mängden av par (k, m) bestående av en nyckel k och en klartext m. Målmängden är utrymmet av kryptotexter C.
Vi kan även på liknande sätt beskriva dekryptering som en funktion

$$d : K \times C \to M.$$

Vi vill att dekrypteringsfunktionen ska göra ogjort resultatet av krypteringsfunktionen. Det kan matematiskt uttryckas som

$$d(k, e(k, m)) = m \quad \text{för alla} \quad k \in K \quad \text{och} \quad m \in M$$

För varje nyckel k får vi ett par funktioner, där vi kan beskriva beroendet på k matematiskt som:

$$e_k : M \to C \quad \text{och} \quad d_k : C \to M$$

som då uppfyller

$$d_k(e_k(m)) = m. \quad \text{för alla} \quad m \in M.$$

Med andra ord så är dekrypteringsfunktionen d_k inversen till krypteringsfunktionen e_k för varje nyckel k.[1] Vi antar att Eve känner till själva krypteringsmetoden. Vilket med matematiska termer innebär att hon känner till funktionerna e samt d men hon vet inte vad nyckeln k är. Det är en så kallad privat nyckel som inte offentligörs men alla som ska kryptera meddelanden och sedermera även den som ska dekryptera meddelanden måste känna till k. Det krävs med andra ord ett hemligt utbyte av nyckeln för alla parter som ska skicka och ta emot meddelanden emellan varandra. Denna metod att kryptera med en privat nyckel kallas symmetrisk kryptering. Styrkan i symmetrisk kryptering är att det generellt går snabbare än alternativen men en uppenbar svaghet blir att parterna som ska kommunicera behöver genomföra ett hemligt nyckelutbyte med varandra så att båda har tillgång till k utan att en möjlig angripare också får tillgång till denna.

1.2.1 Kerkhoffs princip

Kerkhoffs princip bygger på att en krypteringsalgoritms säkerhet ska bygga på säkerheten kring nyckeln och inte själva algoritmen i sig. Det ska således inte spela någon roll om den som vill komma åt den hemliga informationen känner till krypteringsmetoden, bara personen inte känner till nyckeln. Det innebär att metoden kan vara publik och om någon listar ut vad nyckeln är och hur krypteringsfunktionen fungerar är det bara att välja en ny nyckel istället för att behöva tänka ut en helt ny algoritm.

1.2.2 Diffie-Hellman

Att ha en nyckel som enbart Alice och Bob känner till kallas att ha en privat nyckel. Det var det vedertagna sättet att kryptera på innan idén om en publik nyckel blev välkänt och fick uppmärksamhet i och med publiseringen "New Directions in Cryptography" av Whitfield Diffie och Martin Hellman. Det ska emellertid tilläggas att konceptet att kryptera med publik nyckel redan innan Diffie och Hellman hade upptäckts men på grund av olika anledningar hade den informationen hemlighållits tills dess.[2]

De nya rön som Diffie och Hellman bidrog med i och med sin publikation var banbrytande. Framförallt handlar det om deras nya idéer kring ett offentligt nyckelkryptosystem och allt som kom med det. Detta går helt i motsats till de tidigare nämnda symmetriska krypteringsalgoritmerna. De använde sig av en funktion som har en invers men där denna invers är mycket svår att under rimlig tid beräkna, en så kallad envägsfunktion.[3]
Poängen är att skapa så kallade envägsfunktioner med en "fallucka" (trapdoor

[1] Hoffstein, Jeffery, Pipher, Jill och Silverman, Joseph H. *An Introduction to Mathematical Cryptography.* Springer Science+Business Media New York: 2014 s.37

[2] Hoffstein, Pipher och Silverman, 2014 s.37-38 & 61

[3] Hoffstein, Pipher och Silverman, 2014 s.63

på engelska). Denna fallucka är en bit hjälpinformation som gör det enkelt att beräkna inversen på funktionen som annars var otroligt svår att beräkna. På detta sätt kunde man skapa en mycket säker krypteringsmetod.

Istället för enbart en privat nyckel (k_{priv}) ska man även ha en publik nyckel (k_{pub}). Det är detta system med två nycklar istället för en som kallas asymmetrisk kryptering. Vi har ett par av nycklar $k = (k_{priv}, k_{pub})$ och två motsvarande funktioner. För alla publika nycklar k_{pub} finns en krypteringsfunktion $e_{k_{pub}}$

$$e_{k_{pub}} : M \to C$$

och för alla privata nycklar k_{priv} en dekrypteringsalgoritm $d_{k_{priv}}$

$$d_{k_{priv}} : C \to M.$$

Förutsatt att paret k_{priv} och k_{pub} hör till samma k har dessa funktioner egenskapen att

$$d_{k_{priv}}(e_{k_{pub}}(m)) = m. \quad \text{för alla} \quad m \in M.$$

Krypteringsalgoritmen är offentlig och enkel att beräkna och dekrypteringen ska vara enkel för de med en privat nyckel att beräkna men mycket svår utan den privata nyckeln.

Det innebär att om Alice vill ta emot känslig information från fler parter än enbart Bob och samtidigt inte dela all information parterna emellan kan hon använda sig av samma publika nyckel till alla. Till skillnad från symmetrisk kryptering behöver ingen av dessa känna till den privata nyckeln hon sen använder för att dekryptera meddelandena med och således kan enbart Alice tillgodogöra sig informationen. Styrkan ligger då i att inget privat nyckelbyte behöver ske för att Alice ska kunna ta emot känslig information från flera parter samtidigt och hon kan använda samma publika nyckel till detta.

Att hitta en envägsfuntion med tillhörande fallucka för att enkelt beräkna inversen som beskrivet ovan visade sig emellertid mycket svårt att göra och man vet ännu inte om en sådan funktion faktiskt existerar. Diffie och Hellman försökte men lyckades inte. De kom istället på en metod att genom antagandet att det diskreta logaritmproblemet (DLP) är svårt att lösa går det att uppnå säker delning av information över osäkra kanaler med hjälp av en publik nyckel. Det är den metod de utvecklade utifrån DLP som kommit att kallas Diffie-Hellmans nyckelutbyte. [4]

Det diskreta logaritmproblemet

Innan vi vidare går in på Diffie-Hellmans nyckelutbyte ska vi gå igenom vad DLP faktiskt är. Vi behöver emellertid först definiera några begrepp för att sedan gå vidare.

[4] Hoffstein, Pipher och Silverman, 2014 s.63-64

Definition 1.1 Kropp.
En kropp definieras som en mängd $\mathbb{F}$ och två binära opperationer $+$ samt $\cdot$ där följande villkor är uppfyllda:

(i) $\mathbb{F}$ är en kommutativ grupp under $+$
(ii) $\mathbb{F}\backslash\{0\}$ är en kommutativ grupp under $\cdot$
(iii) Den distributiva lagen håller vilket innebär att för alla element a, b och c i $\mathbb{F}$ gäller $a \cdot (b + c) = a \cdot b + a \cdot c$.

Definition 1.2 Ändlig kropp.
En ändlig kropp upfyller ovanstående villkoren för en kropp men även det ytterligare villkoret (iv) att $\mathbb{F}$ innehåller ett ändligt antal element. [5]

Definition 1.3.
För ett primtal p utgör heltalen modulo p en ändlig kropp, med de vanliga operationerna $+$ och $\cdot$ modulo p. Denna kropp betecknas $\mathbb{F}_p$.

Definition 1.4.
Om $\mathbb{F}$ är en kropp så betecknar $\mathbb{F}^*$ de multiplikativt inverterbara elementen i $\mathbb{F}$. Det vill säga $\mathbb{F}^* = \mathbb{F} \setminus \{0\}$.

Definition 1.5 Ordning av ett gruppelement.
Låt $g \in \mathbb{F}_p$ och $g \neq 0$. Ordningen av g i $\mathbb{F}_p$ är det minsta positiva heltalet m så att $g^m = 1$. Detta betecknas $\text{ord}(g)$.

Sats 1.6 Primitiva rötter.
Låt p vara ett primtal. Då existerar ett element $g \in \mathbb{F}_p^*$ vars potenser ger varje element i $\mathbb{F}_p^*$. Det vill säga

$$\mathbb{F}_p^* = \{1, g, g^2, g^3, \ldots, g^{p-2}\}.$$

Vi kommer inte bevisa denna sats i det här arbetet men bevis går att hitta i *Hoffstein et al.* [6] Istället går vi igenom ett exempel för att visa principen.

Exempel 1.7.
Vi väljer en ändlig kropp $\mathbb{F}_p$ där p är ett primtal, säg 7 och undersöker den

[5] Biggs, Norman L. *Discrete Mathematics*. Oxford University Press. 2002 s.299
[6] Hoffstein, Pipher och Silverman, 2014 s.33

multiplikativa gruppen $\mathbb{F}_p^*$. Då är 3 en primitiv rot till gruppen eftersom

$$\begin{aligned} 3^0 &\equiv 1 \pmod 7. \\ 3^1 &\equiv 3 \pmod 7, \\ 3^2 &\equiv 2 \pmod 7, \\ 3^3 &\equiv 6 \pmod 7, \\ 3^4 &\equiv 4 \pmod 7, \\ 3^5 &\equiv 5 \pmod 7. \end{aligned}$$

Vi har alltså fått tillbaka alla element i $\mathbb{F}_7^*$.

Vi kan nu förklara och definiera DLP som ligger till grund för implementeringen av Diffie-Hellmans nyckelutbyte. Det finns olika varianter på problemet beroende på vilken matematisk struktur som används. Vi ska definiera det som berör modulär exponentiering med primtal nu och i en senare del kommer vi även definiera en annan typ som används för kryptering med hjälp av elliptiska kurvor. Själva konceptet kring DLP bygger på svårigheten att beräkna den diskreta logaritmen i en ändlig kropp. Vi kommer definiera detta utifrån en ändlig kropp med ett prim-antal element, vilket vi hädanefter kommer beteckna $\mathbb{F}_p$. *Hoffstein et al.* definierar då DLP som följande:

Definition 1.8 Det diskreta logaritmproblemet.
Låt g vara en primitiv rot för $\mathbb{F}_p$ och låt h vara ett element skillt från noll i $\mathbb{F}_p$. Det diskreta logaritmproblemet (DLP) är problemet att hitta en exponent x sådan att

$$g^x \equiv h \pmod p$$

Talet x kallas den diskreta logaritmen av h till basen g och betecknas $\log_g(h)$.

Diffie-Hellmans nyckelutbyte

Istället för att använda sig av ursprungsidén kring envägsfunktioner hade Diffie och Hellman nu definierat sitt nyckelutbyte med hjälp av DLP istället. Vi antar att Alice och Bob vill utbyta en privat nyckel under förutsättning att Eve avlyssnar all kommunikation. Det första steget i detta nyckelutbyte går då ut på att de kommer överens om ett stort primtal p och ett heltal $g \pmod p$, där $1 \leq g \leq p-1$. Helst ska detta heltal även ha en stor primtalsordning i $\mathbb{F}_p$, det vill säga att $\text{ord}(g)$ är ett stort primtal. [7]

Båda dessa tal är publika och alla, inklusive Eve, får tillgång till dem. Sedan väljer Alice ett heltal a och Bob ett annat heltal b vilka de håller hemliga även

[7] Biggs, 2002 s.265

för varandra.[8] Efter det beräknar de var för sig följande:

$$\begin{aligned} A &\equiv g^a \pmod{p} \quad \text{Alice beräknar detta} \\ B &\equiv g^b \pmod{p} \quad \text{Bob beräknar detta.} \end{aligned}$$

Sedan skickar Alice A till Bob som i sin tur skickar B till Alice och de utför ännu en beräkning:

$$\begin{aligned} A' &\equiv B^a \pmod{p} \quad \text{Alice beräknar detta} \\ B' &\equiv A^b \pmod{p} \quad \text{Bob beräknar detta.} \end{aligned}$$

Nu har de utbytt en privat nyckel med varandra eftersom $A' \equiv B'$, det är följaktligen samma tal modulo p, eftersom

$$A' \equiv B^a \equiv \left(g^b\right)^a \equiv g^{ab} \equiv \left(g^a\right)^b \equiv A^b \equiv B' \pmod{p}.$$[9]

Som tidigare nämnt kom Diffie och Hellman på att trots att p och g är publika och att vi antar att Eve även har tillgång till A och B då dessa tal skickas mellan parterna så blir det i praktiken omöjligt för Eve att lista ut vad den privata nyckeln är tack vare svårigheten i att lösa DPL. Vidare ska vi med bakgrund i detta undersöka hur dessa koncept används för att kryptera hemlig information.

2 Olika krypteringsalgoritmer

2.1 RSA

Kort efter Diffie och Hellmans publicering om en säker metod för ett nyckelutbyte kom Ron Rivest, Adi Shamir och Len Adleman att förändra sättet vi krypterar på med sina nyskapande idéer kring asymmetrisk kryptering. Innan vi kan gå in på hur kryptering med RSA fungerar behöver vi ha koll på några viktiga satser.

Sats 2.1 Fermats lilla sats.
Låt p vara ett primtal. För alla $a \not\equiv 0 \pmod{p}$ gäller då att $a^{p-1} \equiv 1 \pmod{p}$ och för alla a gäller $a^p \equiv a \pmod{p}$.

Bevis. Låt p vara ett primtal och låt G vara gruppen som består av elementen $1, 2, \ldots, p-1$ med operationen multiplikation modulo p. Denna grupp har ordningen $p-1$. Antag att vi väljer ett element a i G där $1 \leq a \leq p-1$ och beteckna a:s ordning med k, vilket innebär att det är det minsta heltalet så att

[8] Hoffstein, Pipher och Silverman, 2014 s.64-67

$a^k \equiv 1 \pmod{p}$. Enligt Lagranges teorem är k en delare till gruppens ordning, det vill säga $p - 1 = k \cdot n$ för något heltal n. Därmed gäller att

$$a^{p-1} \equiv a^{k \cdot n} \equiv (a^k)^n \equiv 1^n \equiv 1 \pmod{p}.$$

Om vi sedan multiplicerar båda sidor med a, erhåller vi

$$a^p \equiv a \pmod{p}.$$

Det finns även många andra sätt att bevisa detta på om man inte är bekant med gruppteori. [10] □

Definition 2.2.
Vi säger att ett element $g \in \mathbb{F}_m$ är *inverterbart* om det existerar något $x \in \mathbb{F}_m$ sådant att $gx = 1 \in \mathbb{F}_m$. Då kallas x inversen till g och betecknas som $x = g^{-1}$

Sats 2.3.
Låt a och b vara två positiva heltal, och låt $d = \text{sgd}(a, b)$. Då existerar två heltal m och n så att

$$d = ma + nb.$$

Bevis för denna sats finns att hitta i *Biggs* [11]

Sats 2.4.
Elementet $g \in \mathbb{F}_m$ är inverterbart om och endast om g och m är koprima i $\mathbb{F}$. Om p är ett primtal så är varje element i $\mathbb{F}_p$ förutom 0 inverterbart.

Bevis. Antag att g är inverterbart, så att $gx = 1 \in \mathbb{F}_m$. Då följer det att vi i $\mathbb{F}$ har $gx - 1 = km$ för något heltal k eller

$$gx - km = 1.$$

Alla gemensamma delare till g och m måste dela $gx - km$, vilket är 1, så $\text{sgd}(g, m) = 1$.
Omvänt, antag att $\text{sgd}(g, m) = 1$. Enligt 2.3 existerar det då två heltal x och y så att $gx + my = 1$. Detta kan vi skriva om som

$$gx \equiv 1 \pmod{m}.$$

□

Definition 2.5 Eulers ϕ-funktion.
Två heltal x och y vars största gemensamma delare är 1 kallas relativt prima. För varje $n \geq 1$ betecknar $\phi(n)$ antalet x i intervallet $1 \leq x \leq n$ sådana att x och n är relativt prima. [12]

[10] Hoffstein, Pipher och Silverman, 2014 s.30-31
[11] Biggs, 2002 s.69
[12] Biggs, 2002 s.95

En viktig egenskap som följer från Eulers funktion är denna sats:

Sats 2.6.
Låt p vara ett godtyckligt primtal, då gäller följande:

$$\phi(p) = p - 1.$$

Bevis. Alla tal från 1 till $p-1$ är relativt prima till p. Det innebär att deras största gemensamma delare är 1, något som följer av att p är ett primtal. □

Det finns ytterligare en viktig sats som följer av detta.

Sats 2.7 Eulers sats.
Om $\text{sgd}(y, m) = 1$ så innebär det att $y^{\phi(m)} \equiv 1 \pmod{m}$.

Bevis. Låt

$$A = \{a_1, a_2, \ldots, a_{\phi(m)}\}$$

vara mängden som innehåller alla heltal från 1 till m som är relativt prima med m. Vi betraktar också mängden

$$y \cdot A = \{ya_1, ya_2, \ldots, ya_{\phi(m)}\}.$$

Eftersom $\text{sgd}(y, m) = 1$, är varje element i $y \cdot A$ också relativt primt med m, det vill säga $y \cdot A \subseteq A$. Vi vet även att om

$$ya \equiv yx \pmod{m}$$

genom att multiplicera med y^{-1}, som vi vet existerar enligt sats (2.4), så innebär det att

$$a \equiv x \pmod{m}.$$

En produkt av inverterbara element är inverterbart. Notera att $y \cdot A$ är en permutation av A. Alltså är elementen ya_j i $y \cdot A$ alla olika. Så $y \cdot A$ består av av $\phi(m)$ olika element, och eftersom $y \cdot A \subseteq A$ så är därmed $y \cdot A = A$. Således gäller att:

$$ya_1 \cdot ya_2 \cdot \ldots \cdot ya_{\phi(m)} = y^{\phi(m)} \cdot (a_1 \cdot a_2 \cdot \ldots \cdot a_{\phi(m)}) \pmod{m} \equiv a_1 \cdot a_2 \cdot \ldots \cdot a_{\phi(m)} \pmod{m}$$

Eftersom $a_1 \cdot a_2 \cdot \ldots \cdot a_{\phi(m)}$ är relativt prima med m kan vi dividera båda sidorna av kongruensen med denna produkt och vi har:

$$y^{\phi(m)} \equiv 1 \pmod{m}.$$

□

Sats 2.8 Eulers sats (följdsats).
Låt a vara ett heltal och p samt q vara två olika primtal. Om $g = \gcd(p-1, q-1)$ så gäller att $a^{(p-1)(q-1)/g} \equiv 1 \pmod{pq}$ för alla a som uppfyller $\gcd(a, pq) = 1$.

Vi kommer inte att bevisa detta här men bevis på denna sats går att hitta i *Hoffstein et al.* [13].

Kryptering med hjälp av RSA grundat på beskrivning i *Hoffstein et al.*[14] följer nedan.

Nyckelgenerering
Bob väljer två stora primtal p samt q och beräknar produkten av dessa vilket vi benämner som $n = pq$. Sedan beräknas $\phi(n) = (p-1)(q-1)$ där ϕ är Eulers fi-funktion. För att sedan bestämma exponenten till den offentliga nyckeln e väljs ett heltal sådant att $1 \leq e \leq \phi(n)$ vidare ska även e och $\phi(n)$ vara relativt prima. Slutligen bestämmer vi exponenten till den privata nyckeln d så att $de \equiv 1 \pmod{\phi(n)}$. Både e samt $n = pq$ är publika och alla har tillgång till dem.

Kryptera
Vi kallar meddelandet (eller klartexten) som ska krypteras för m där, m är ett heltal och $0 \leq m < n$. Om nu Alice vill skicka ett krypterat meddelande c till Bob så använder hon Bobs publika nyckel (n, e) och gör det med hjälp av formeln $m^e \pmod{n} = c$.

Dekryptera
När Bob ska dekryptera Alice meddelande c använder han sin privata nyckel (n, d) för att få tillbaka meddelandet m. Detta görs genom ekvationen $m' \equiv c^d \pmod{n}$ där $m' = m$. Bevis för varför detta fungerar kommer senare i texten.

Exempel 2.9 Kryptera med RSA.
Nyckelgenerering
Bob väljer två primtal $p = 5$ och $q = 11$. Han beräknar $n = pq = 5 \cdot 11 = 55$. Han beräknar också $\phi(n) = (p-1)(q-1) = (5-1)(11-1) = 4 \cdot 10 = 40$. Han väljer en publik exponent e som är relativt prima till $\phi(n)$, till exempel $e = 3$. Bob behöver sedan beräkna en privat nyckel d sådan att $ed \equiv 1 \pmod{\phi(n)}$. Han beräknar $d = 27$ eftersom $3 \cdot 27 \equiv 1 \pmod{40}$. Bob publicerar sin publika nyckel $(n, e) = (55, 3)$. Hans privata nyckel är $d = 27$, vilket han håller hemligt.

Kryptering
Allice vill skicka meddelandet $m = 13$ till Bob. Hon använder sig av Bob's publika nyckel för att kryptera meddelandet med krypteringsmetoden $c = m^e$

[13] Hoffstein, Pipher och Silverman, 2014 s.119 ff
[14] Hoffstein, Pipher och Silverman, 2014 s.118-123

(mod n), alltså

$$c = 13^3 \pmod{55} = 52.$$

Allice skickar det krypterade meddelandet $c = 52$ till Bob.

Dekryptering
Bob tar emot det krypterade meddelandet $c = 52$. Han använder sig av sin privata nyckel $d = 27$ för att dekryptera meddelandet med dekrypteringsalgoritmen $m = c^d \pmod{n}$, dvs,

$$m = 52^{27} \pmod{55} = 13$$

Exemplet angivet ovan består av alldeles för små primtal för att utgöra en säker kryptering men illusterar metoden RSA.

Som vi såg i avsnitt 1.2.2 byggde Diffie-Hellmans nyckelutbyte på svårigheten i att lösa

$$a^x \equiv b \pmod{p}$$

där p är ett primtal. a,b och p är publika och x är den okända variablen. RSA bygger istället på svårigheten i att lösa

$$x^e \equiv c \pmod{n}$$

och här är istället e, c samt n kända.[15] Det är följaktligen detta samband vi måste visa.

Sats 2.10 RSA.
Låt p och q vara primtal, $n = pq$, $\phi(n) = (p-1)(q-1)$, och låt e och d vara sådana att $ed \equiv 1 \pmod{\phi(n)}$. Om m är ett heltal sådant att $0 \leq m < n$, och $c \equiv m^e \pmod{n}$, då är $m \equiv c^d \pmod{n}$.

Bevis. Eftersom $de \equiv 1 \pmod{(p-1)(q-1)}$ vet vi från definitionen av modulo att $de = 1 + k(p-1)(q-1)$ för något heltal k. Eftersom $c \equiv m^e \pmod{pq}$ får vi

$$\begin{aligned}
&(c^d) \equiv (m^e)^d \pmod{pq} \\
&\equiv m^{de} \pmod{pq} \\
&\equiv m^{1+k(p-1)(q-1)} \text{ eftersom } de = 1 + k(p-1)(q-1) \\
&\equiv m \cdot \left(m^{(p-1)(q-1)}\right)^k \pmod{pq} \\
&\equiv m \cdot 1^k \pmod{pq} \text{ Följer från Eulers sats (följdsats)} \\
&\equiv m \pmod{pq}
\end{aligned}$$

[15] Hoffstein, Pipher och Silverman, 2014 s.119

Vi vet nu att $m = c^d$ är en lösning till kongruensen $m^e \equiv c \pmod{pq}$, alltså det samband vi använder för krypteringen. Vi måste även bevisa att det är den enda lösningen och gör detta genom att anta att $u = m$ är en lösning.

$$
\begin{aligned}
u &\equiv u^{de-k(p-1)(q-1)} \pmod{pq} \quad \text{eftersom } de = 1 + k(p-1)(q-1), \\
&\equiv (u^e)^d \cdot (u^{(p-1)(q-1)})^{-k} \pmod{pq}, \\
&\equiv (u^e)^d \cdot 1^{-k} \pmod{pq} \quad \text{med användning av Eulers formel på samma sätt som tidigare,} \\
&\equiv c^d \pmod{pq} \quad \text{eftersom vi bevisat att } u \text{ är en lösning ovan.}
\end{aligned}
$$

Således har vi fastställt att om $c \equiv m^e \pmod{pq} \iff m \equiv c^d \pmod{pq}$. □

2.2 Kryptering med Elgamal

RSA var den första publika krypteringsalgoritmen men det finns ett annat vida spritt krypteringssystem uppfunnet av den egyptisk-amerikanska kryptografen Taher Elgamal 1985. Algoritmen har nära koppling till Diffie & Helmans metod för delning av publika nycklar och använder sig av asymetrisk kryptering. Även här används en ändlig primtalskropp $\mathbb{F}_p$.

Först ska vi gå igenom hur Elgamal används med hjälp av det diskreta logaritmproblemet och i nästa kapitel kommer vi undersöka hur det kan användas tillsammans med elliptiska kurvor. Det ser till stor del väldigt likt ut hur RSA fungerar då de använder samma metod som Diffie och Hellman utvecklade ett antal år tidigare. Se definition 1.8 om DLP. Precis som för RSA följer Elgamal tre steg, generering av nycklar, kryptering samt dekryptering. Metoden går ut på att Alice delar en publik nyckel och en algoritm som alla har tillgång till. Den publika nyckeln är bara ett stort tal och själva algoritmen är metoden Bob ska använda för att senare kunna kryptera ett meddelande han vill skicka till Alice. Den privata nyckeln, som är ett annat tal, håller Alice för sig själv för att enbart hon ska kunna dekryptera meddelandet.
Beskrivningen av kryptering med Elgamal nedan bygger på diskussionen i *Hoffstein et al.* [16]
Alice använder sig av ett stort primtal p för vilket diskreta logaritmproblemet i $\mathbb{F}_p^*$ är svårt att lösa. Vidare behöver hon även ett element $g \in \mathbb{F}_p$ med stor primtalsordning. Hon kan själv välja p och g, eller få dem tilldelade av en pålitlig tredjepart, exempelvis en myndighet.

Generera nycklar
Det första steget är att välja en privat nyckel a där $1 \leq a \leq p-1$ och sedan beräkna $A \equiv g^a \pmod{p}$ och i likhet med Diffie-Hellman metoden behåller hon a hemlig men publicerar A.

Kryptera
Nästa steg är krypteringsalgoritmen. Att kryptera med Elgamal går ut på att

[16] Hoffstein, Pipher och Silverman, 2014 s.70 ff

Bob har ett meddelande m, vilket (om det är i klartext) görs om till ett heltal mellan 2 och p. Detta kan ske genom att koda tecken i meddelandet till binära representationer, vilket bland annat går att läsa om i *Hoffstein et al* [17]. Om m är större än p, delas det upp i mindre block $m_1, m_2, \ldots, m_n$ där varje $m_i < p$.

Sedan väljer han ett heltal $k \pmod p$ och efter att han använt k till att kryptera meddelandet använder han aldrig samma k igen. Detta kallas ett slumpmässigt element och existerar enbart för att kryptera ett enda meddelande. Vidare kan han använda m, k samt A för att beräkna två kvantiterer som följande:

$$c_1 \equiv g^k \pmod p \quad \text{och} \quad c_2 \equiv mA^k \pmod p.$$

Bobs krypterade meddelande som han sedan skickar till Alice blir följaktligen paret (c_1, c_2).

Dekryptera
Nu kan Alice, med hjälp av sin privata nyckel beräkna den multiplikativa inversen av c_1^a modulo p,

$$x \equiv (c_1^a)^{-1} \pmod p,$$

Genom att använda snabbexponentiering (se metod 2.4.1) kan hon beräkna $c_1^{p-1-a} \pmod p$. Därefter multipliceras x med c_2 för att få fram klartexten m.

Sats 2.11 Kryptering med Elgamal.
Låt p vara ett primtal, m ett heltal $2 \leq m < p$, k och a vara heltal samt $g \in \mathbb{F}_p$. Låt $A \equiv g^a \pmod p$. Om $c_1 \equiv g^k \pmod p$ och $c_2 \equiv mA^k \pmod p$, då är $m \equiv (c_1^a)^{-1} c_2 \pmod p$.

Bevis. Låt x beteckna den multplikativa inversen av $c_1^a \pmod p$

$$\begin{aligned}
x \cdot c_2 &\equiv (c_1^a)^{-1} \cdot c_2 \pmod p, && \text{eftersom } x \equiv (c_1^a)^{-1} \pmod p,\\
&\equiv (g^{ak})^{-1} \cdot (mA^k) \pmod p, && \text{eftersom } c_1 \equiv g^k, c_2 \equiv mA^k \pmod p\\
&\equiv (g^{ak})^{-1} \cdot (m(g^a)^k) \pmod p, && \text{eftersom } A \equiv g^a \pmod p,\\
&\equiv m \pmod p, && \text{eftersom termerna } g^{ak} \text{ tar ut varandra.}
\end{aligned}$$

□

Exempel 2.12 Kryptering med Elgamal.
För att undersöka detta i praktiken kan vi exemplifiera med att Alice väljer primtalet 487 och sätter $g = 5$. Ett krav för kryptering med Elgamal i praktiken är att g har stor primtalsordning (1.5), i detta exempel bortser vi från detta för att tydligare illustrera själva metoden. Vidare väljer hon $a = 123$ som sin privata nyckel och beräknar med hjälp av snabb exponentiering

$$A \equiv g^a \equiv 5^{123} \equiv 228 \pmod{487}$$

[17] Hoffstein, Pihper och Silverman, 2014 s. 39ff

Detta är således hennes publika nyckel hon sedan delar offentligt. Bob kan då kryptera sitt meddelande $m = 299$ genom att välja ett slumpvist valt $k = 233$ och beräkna de två kvantiteterna c_1 samt c_2.

$$c_1 \equiv g^k \equiv 5^{233} \equiv 343 \pmod{487}$$

$$c_2 \equiv m \cdot A^k \equiv 299 \cdot 228^{233} \equiv 452 \pmod{487}.$$

Paret $(c_1, c_2) = (343, 452)$ blir kryptotexten Bob skickar till Alice. Hon kan sen med hjälp av $a = 123$ dekryptera meddelandet genom att först beräkna x.

$$x \equiv (c_1^a)^{-1} \equiv c_1^{p-1-a} \equiv 343^{364} \equiv 14 \pmod{487}.$$

Slutligen beräknar hon

$$c_2 \cdot x \equiv 452 \cdot 14 \equiv 299 \pmod{487},$$

vilket är det ursprungliga meddelandet m som Bob skickade.

Elgamal är vad man brukar benämna som en 2-till-1-meddelandeexpansion vilket innebär att den krypterade texten kräver dubbelt så många bitar minne som klartexten då den består av två heltal av ungefär samma storlek istället för ett. Vidare kan man fundera kring vilka övriga svagheter förutom det faktum att det tar mer lagringsutrymme det finns kring kryptering med Elgamal. Svårigheten för Eve att knäcka koden är lika svårt som för henne att lösa det diskreta logaritmproblemet som beskrivet av Diffie och Hellman, se definition 1.8. Därav blir det en mycket säker metod för kryptering.[18]

2.3 Elliptic Curve Cryptography

2.3.1 Elliptiska kurvor

Vi ska nu undersöka ytterligare en krypteringsalgoritm men innan det behöver vi definiera vad vi menar med elliptiska kurvor.

Definition 2.13 Elliptiska kurvor.
Låt $\mathbb{F}$ vara en kropp och låt $a, b \in \mathbb{F}$ med $4a^3 + 27b^2 \neq 0$. Då är en elliptisk kurva över $\mathbb{F}$ mängden lösningar $(x, y) \in \mathbb{F} \times \mathbb{F}$ till ekvationen

$$y^2 = x^3 + ax + b$$

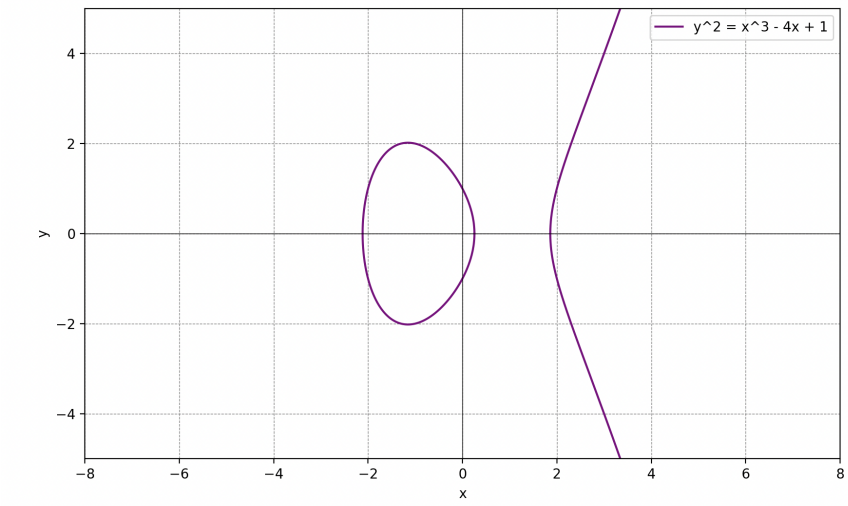
tillsammans med en extra punkt $\mathcal{O}$. Vi betecknar denna mängd $E(\mathbb{F})$.

Vad denna extrapunkten är och hur matematiken kring den spelar sin roll kommer vi in på lite senare. Efter matematikern som studerade dessa ekvationer utförligt under nittonhundratalet uppkallades ekvationer på den här formeln till Weierstrass-ekvationer.[19]

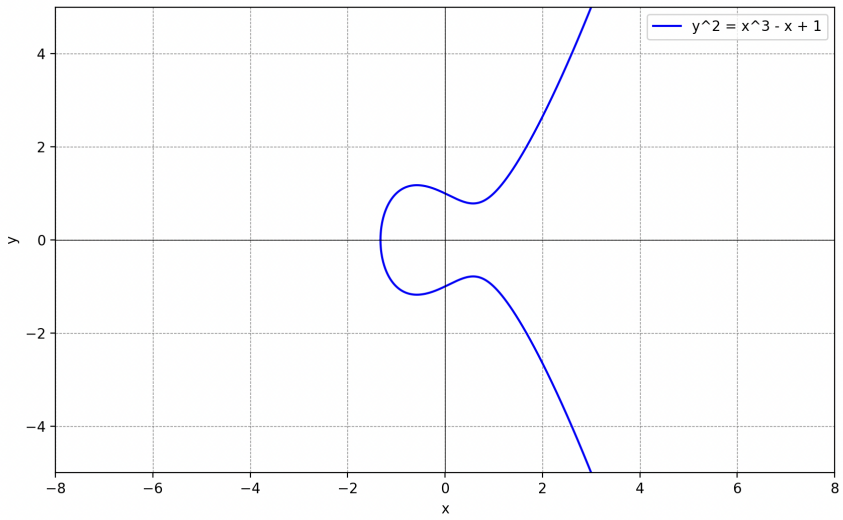
[18] Hoffstein, Pipher och Silverman, 2014 s.72-73
[19] Hoffstein, Pipher och Silverman, 2014 s.299 & 303

Exempel på två elliptiska kurvor kan se ut som följande:

[2.1] Kurvan: $y^2 = x^3 - 4x + 1$ över $\mathbb{R}$



figur[2.2] Kurvan: $y^2 = x^3 - x + 1$ över $\mathbb{R}$



Något vi kan notera är att dessa kurvor är symmetriska över x-axeln, en egenskap som alltid gäller för elliptiska kurvor. Detta beror på att $(-y)^2 = y^2$, följaktligen ligger punkten (x, y) på kurvan om och endast om $(x, -y)$ ligger på

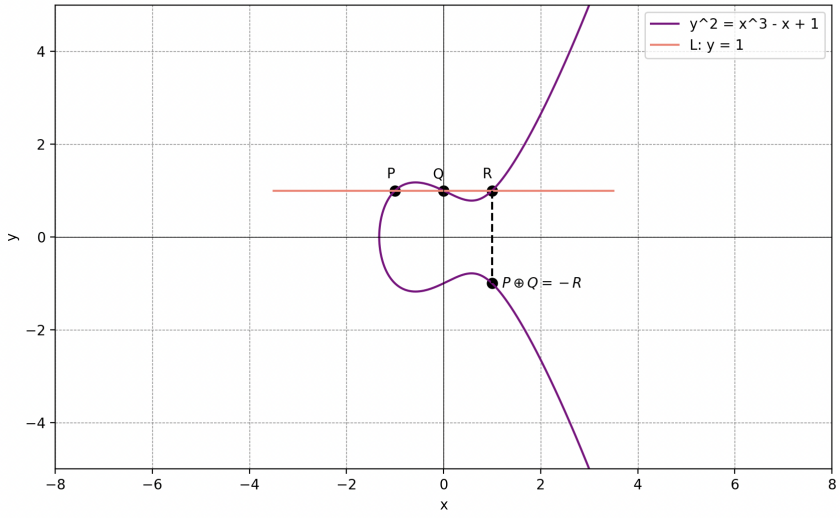
kurvan.[20]

När man pratar om operationen att addera två punkter på en elliptisk kurva och producera en tredje punkt är det till stor del likt addition som vi känner till det men mycket skiljer sig även från hur vi vanligen adderar två punkter.

Definition 2.14 Addition av punkter på en elliptisk kurva över $\mathbb{R}$.
Låt E vara en elliptisk kurva över $\mathbb{R}$ och låt P och Q vara två punkter på E. Låt L vara linjen som förbinder P och Q, eller tangentlinjen till E vid P om $P = Q$. För att addera P och Q, drar man en linje L som går genom båda punkterna. När vi drar linjen L innebär det att vi löser en linjär ekvation tillsammans med kurvans ekvation. Och då den kubiska ekvationen har tre rötter, kommer linjen därför även att skära E i en tredje punkt, vilken vi kallar R. Additionen av P och Q definieras som den punkt $-R$, som är speglingen av R över x-axeln. Resultatet av additionen betecknas $P \oplus Q = -R$. Vi definierar $P - Q$ (eller $P - Q$) som $P \oplus (-Q)$. På samma sätt representeras upprepad addition som multiplikation av en punkt med ett heltal, $nP = P+P+P+\cdots+P$ (n kopior).[21]. (Särskilda fall, till exempel där $P = Q$ eller där linjen genom P & Q är vertikal kommer att bemötas i sats (2.16) efter ett exempel).

Vi illusterar detta nedan i figur 2.3.1, där punkten $-R$ är speglingen av R över x-axeln.

[2.3] Kurvan: $y^2 = x^3 - x + 1$ över $\mathbb{R}$



[20] Hasan, Harady, *Elliptic Curves: A journey through theory and its applications* 2019 s.7
[21] Hoffstein, Pipher och Silverman, 2014 s.300

Exempel 2.15 Addition av två punkter på en elliptisk kurva.
Punkterna $P = (-1, 1)$ och $Q = (0, 1)$ ligger på kurvan $y^2 = x^3 - x + 1$. För att addera dessa två punkter behöver vi först dra en linje mellan punkterna som i figur 2.3 och som vi vet ges ekvationen för denna linje av räta linjens ekvation. På grund av kurvans utformning och de valda punkterna blir denna uträkning väldigt enkel (då linjen L har lutningen 0) men vi går igenom alla steg ändå för att ge en tydlig illustration av uträkningarna. Vi har således att $y = kx + m$ där lutningen $k = \frac{\Delta y}{\Delta x} = \frac{0}{2} = 0$ och m ges av linjens skärningspunkt på y-axeln, alltså 1. Följakligen är ekvationen för L som följande $L : y = 1$.

För att finna punkten R där L skär kurvan sätter vi in ekvationen L som y i ekvationen $y^2 = x^3 + ax + b$, där a, b är konstanter som uppfyller $4a^3 + 27b^2 \neq 0$, och löser för $x = 0$.

$$1^2 = x^3 - x + 1 \iff 0 = x^3 - x + 1 - 1 \iff 0 = x^3 - x \iff 0 = x(x-1)(x+1)$$

Vi känner redan till två av rötterna från punkten P (-1) samt Q (0) därav ser vi sedan genom ekvationen ovan att den tredje måste vara 1. Vi hade en väldigt enkel ekvation till linjen L vilket gav osedvanligt simpla ekvationer för ett tredjegrads polynom. Men tack vare att vi redan hade två rötter sedan tidigare är det generellt sett inga större svårigheter att finna den sista roten som ger x-värdet till punkten R.
Vi kan nu sätta in x-värdet i ekvationen $y^2 = x^3 - x + 1$ för att få fram även y-värdet i den punkten. Allså är $R = (1, 1)$, så $P \oplus Q = (1, -1)$.

Sats 2.16 Additionslagarna för elliptiska kurvor över $\mathbb{R}$.
Låt E vara en elliptisk kurva över $\mathbb{R}$, då gäller följande additionslagar för E:

(i) $P \oplus \mathcal{O} = \mathcal{O} \oplus P = P$ för alla $P \in E$. [Identitet]

(ii) $P \oplus (-P) = \mathcal{O}$ för alla $P \in E$. [Invers]

(iii) $(P \oplus Q) \oplus R = P \oplus (Q \oplus R)$ för alla $P, Q, R \in E$. [Associativ]

(iv) $P \oplus Q = Q \oplus P$ för alla $P, Q \in E$. [Kommutativ]

Bevis. Associativiteten är svår att bevisa men det finns många böcker om elliptiska kurvor där bevis går att finna. Men de andra lagarna kan vi gå igenom. Identitetslagen samt inverslagen är sanna per definition då $\mathcal{O}$ ligger på varje vertikal linje. Den kommutativa lagen är lätt att verifiera, eftersom linjen som går genom P och Q är densamma som linjen som går genom Q och P, så ordningen av punkterna spelar ingen roll.[22] □

[22] Hoffstein, Pipher och Silverman, 2014 s.300-304

Vi vill nu även utöka dessa additionslagar så att de även gäller för elliptiska kurvor över $\mathbb{F}_p$ vilket vi gör på följande sätt:

Sats 2.17 Additionsalgoritm för elliptiska kurvor över $\mathbb{R}$.
Låt

$$E : y^2 = x^3 + ax + b$$

vara en elliptisk kurva över $\mathbb{F}$ och låt P_1 och P_2 vara punkter på E.

(i) Om $P_1 = \mathcal{O}$, då är $P_1 \oplus P_2 = P_2$.

(ii) Annars, om $P_2 = \mathcal{O}$, då är $P_1 \oplus P_2 = P_1$.

(iii) Annars, skriv $P_1 = (x_1, y_1)$ och $P_2 = (x_2, y_2)$.

(iv) Om $x_1 = x_2$ och $y_1 = -y_2$, då är $P_1 \oplus P_2 = \mathcal{O}$.

(v) Annars, definiera λ genom

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{om } P_1 \neq P_2, \\ \frac{3x_1^2 + a}{2y_1} & \text{om } P_1 = P_2, \end{cases}$$

och låt

$$x_3 = \lambda^2 - x_1 - x_2$$

och

$$y_3 = \lambda(x_1 - x_3) - y_1.$$

Då är $P_1 \oplus P_2 = (x_3, y_3)$.

Bevis. (i) och (ii) följer från samma resonemang som 2.3.1. För (iv) gäller att om linjen genom P_1 och P_2 är vertikal, så blir $P_1 + P_2 = O$. Observera att om $y_1 = y_2 = 0$, då är tangentlinjen vertikal, och detta fall fungerar också. För (v) ser vi att om $P_1 \neq P_2$, så är λ lutningen på linjen genom P_1 och P_2, och om $P_1 = P_2$, så är λ lutningen på tangentlinjen vid $P_1 = P_2$. I båda fallen ges linjen L av ekvationen $y = \lambda x + \nu$ där $\nu = y_1 - \lambda x_1$. Substitution av L:s ekvation i E:s ekvation ger:

$$(\lambda x + \nu)^2 = x^3 + ax + b,$$

så

$$x^3 - \lambda^2 x^2 + (a - 2\lambda\nu)x + (b - \nu^2) = 0.$$

Vi vet att denna kubiska ekvation har x_1 och x_2 som två av sina rötter. Om vi kallar den tredje roten för x_3, då faktoriseras den som:

$$x^3 - \lambda^2 x^2 + (a - 2\lambda\nu)x + (b - \nu^2) = (x - x_1)(x - x_2)(x - x_3).$$

Multiplicera ut höger sida och jämför koefficienten för x^2 på vardera sidan. Koefficienten för x^2 på höger sida är $-x_1 - x_2 - x_3$, vilket måste motsvara $-\lambda^2$, koefficienten för x^2 på vänster sida. Detta låter oss lösa för

$$x_3 = \lambda^2 - x_1 - x_2,$$

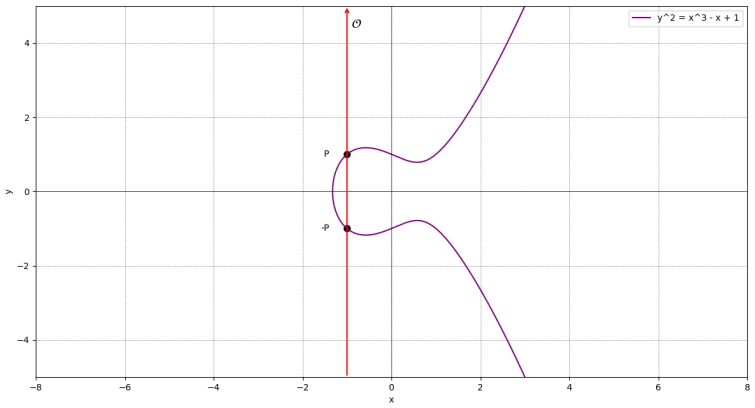
och sedan ges y-koordinaten för den tredje skärningspunkten mellan E och L av $\lambda x_3 + \nu$, eftersom punkten ligger på L. Slutligen, för att få $P_1 \oplus P_2$, måste vi spegla över x-axeln. Det innebär att vi byter ut y-koordinaten till den negativa istället, det vill säga

$$y_3 = -\nu - \lambda x^3.^{23}$$

□

Vi illustrerar nu en annan del av satsen, nämligen då vi vill addera punkten $P = (a, b)$ med dess reflektion över x-axeln $-P = (a, -b)$, i vilken annan punkt kommer linjen L då skära den elliptiska kurvan? Linjen som skär båda punkterna är $x = a$ men den kommer ju endast skära kurvan i punkterna P samt $-P$. Satsen säger då att $P \oplus (-P) = \mathcal{O}$. Vi kan tänka på det som att vi skapar en punkt $\mathcal{O}$ som inte existerar i xy-planet utan vi säger att den existerar i "oändligheten" men vi tänker oss att den ligger på varje vertikal linje (se figur 2.4 nedan).

[2.4] Kurvan: $y^2 = x^3 - x + 1$ över $\mathbb{R}$



Ytterligare ett problem som då kan uppstå är vad som händer om vi vill addera P med punkten $\mathcal{O}$. Linjen L som förbinder dessa två punkter är den vertikala linjen genom P då $\mathcal{O}$ ligger på vertikala linjer och den linjen skär kurvan i punkterna P, $-P$ samt $\mathcal{O}$. Det vi då gör när vi ska addera P till $\mathcal{O}$ är att vi speglar $-P$ över x-axeln och får tillbaka punkten P. Vi kan således se $\mathcal{O}$ som nollelementet för elliptiska kurvor och satsen säger att $P \oplus \mathcal{O} = P$.[24]

2.3.2 Kryptering med Elliptiska kurvor

Innan vi går in på hur man krypterar med hjälp av elliptiska kurvor behöver vi undersöka elliptiska kurvor med punkter i ett ändligt fält $\mathbb{F}_p$.

[24] Hoffstein, Pipher och Silverman, 2014 s.303

Definition 2.18 Elliptisk kurva över en ändlig kropp.
Låt $p \geq 3$ vara ett primtal. Då är en elliptisk kurva över $\mathbb{F}_p$ en ekvation på formen:

$$E : y^2 = x^3 + ax + b \text{ med } a, b \in \mathbb{F}_p \text{ som uppfyller } 4a^3 + 27b^2 \neq 0.$$

Mängden av punkter på E med koordinater i $\mathbb{F}_p$ är mängden

$$E(\mathbb{F}_p) = \{(x, y) : x, y \in \mathbb{F}_p \text{ satisfierar } y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}\}$$

[25] (Vi minns att $\mathcal{O}$ symboliserar nollelementet för elliptiska kurvor). Vi definierar nu additionen av två punkter på $E(F_p)$ genom egenskaper (i)-(v) från sats (2.17).

Exempel 2.19.
För att exemplifiera detta kan vi ta kurvan

$$E : y^2 = x^3 + x + 6 \text{ över fältet } \mathbb{F}_{11}.$$

För att hitta punkterna på $E(\mathbb{F}_{11})$ testar vi alla möjliga värden på $x = (0, 1, 2, ...10)$ och kollar vilka av dessa i ekvationen $y^2 = x^3 + x + 6$ som genererar en kvadrat modulo 11. Vi börjar med att lista alla element i $\mathbb{F}_{11}$ och kvadrerar dem

$$\begin{aligned}
0^2 &= 0 \pmod{11} \\
1^2 &= 1 \pmod{11} \\
2^2 &= 4 \pmod{11} \\
3^2 &= 9 \pmod{11} \\
4^2 &= 16 \equiv 5 \pmod{11} \\
5^2 &= 25 \equiv 3 \pmod{11} \\
6^2 &= 36 \equiv 3 \pmod{11} \\
7^2 &= 49 \equiv 5 \pmod{11} \\
8^2 &= 64 \equiv 9 \pmod{11} \\
9^2 &= 81 \equiv 4 \pmod{11} \\
10^2 &= 100 \equiv 1 \pmod{11}
\end{aligned}$$

Vi testar därefter samtliga värden för x och undersöker vilka värden som ger en kvadrat (mod 11). Vi börjar med $x = 0$ vilket ger 6, vilket vi noterar i listan ovan inte är en kvadrat (mod 11), detsamma gäller för $x = 1$. Om vi däremot undersöker $x = 2$ får vi ekvationen $2^2 + 2 + 6 = 16 \equiv 5 \pmod{11}$ vilket vi ser i listan är en kvadrat (mod 11) med rötterna $y = 4$ & $y = 7$. Fortsätter vi på detta sätt ser vi att $2, 3, 5, 7, 8$ samt 10 alla har rötter och detta visar specifika x-värden där y^2 bildar en kvadrat modulo 11 och de motsvarande y-värdena (kvadratrötterna) som ger giltiga punkter på kurvan. De fullständiga punkterna

[25] Hoffstein, Pipher och Silverman, 2014 s.306

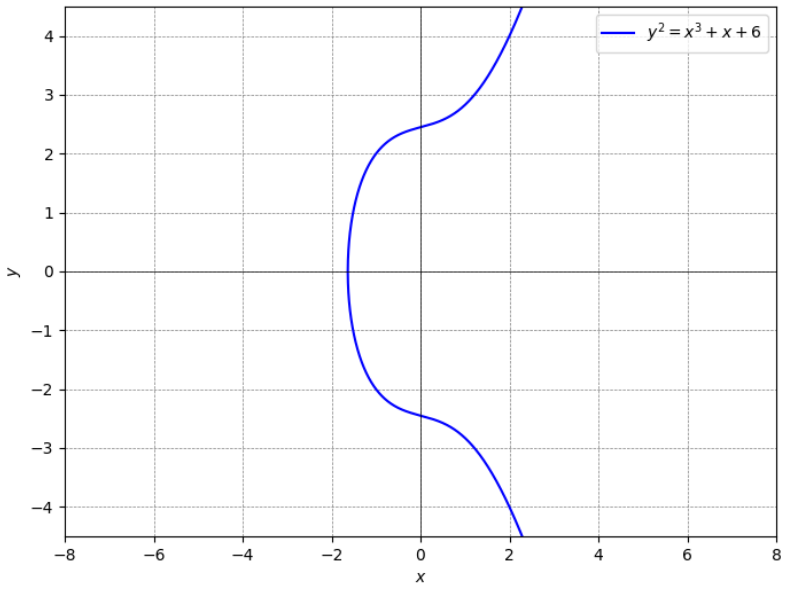
på $E(\mathbb{F}_{11})$ inkluderar dessa par av x och y tillsammans med oändlighetspunkten $\mathcal{O}$, vilket ger oss:

$$E(\mathbb{F}_{11}) = \{\mathcal{O}, (2,4), (2,7), (3,5), (3,6), (5,2), (5,9), (7,2), (7,9), (8,3), (8,8), (10,2), (10,9)\}$$

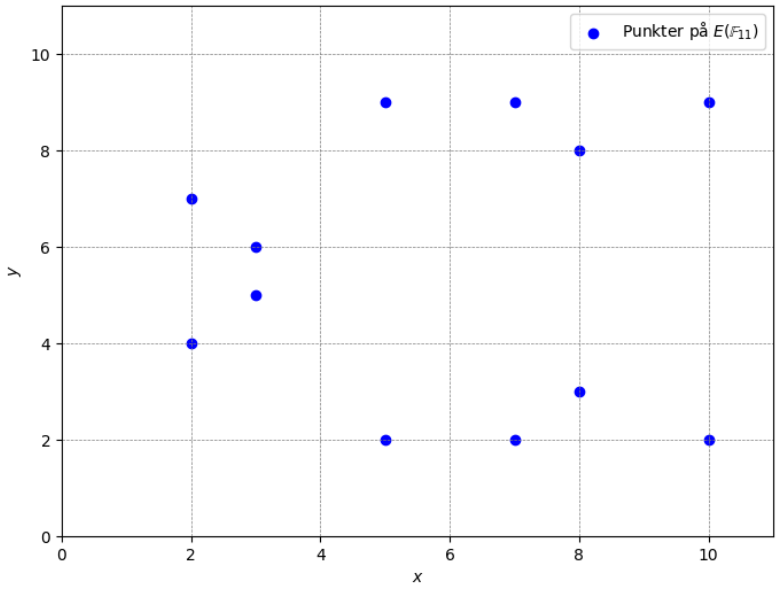
Vi har således hittat 13 punkter till kurvan. För att definiera addition mellan två punkter på en elliptisk kurva i ett ändligt fält använder vi definition 2.18 som beskrivet tidigare.

Notera att kurvan ändar utseende då vi undersöker den i $\mathbb{F}_{11}$, se figur 2.5 samt 2.6 nedan.

[2.5] Kurvan: $E : y^2 = x^3 + x + 6$ över $\mathbb{R}$



[2.6] Kurvan: $E : y^2 = x^3 + x + 6$ över $\mathbb{F}_{11}$



Det diskreta logaritmproblemet för elliptiska kurvor (ECDLP)

Tidigare diskuterade vi det diskreta logaritmproblemet (DLP) i den ändliga kroppen $\mathbb{F}_p$ som beskrivet av Diffie och Hellman, se definition 1.8. Den algoritmen gick ut på att Alice valde två tal g och x och listar ut h. Hon publicerar sedan g och h, hennes hemlighet blir således exponenten x som löser kongruensen

$$h \equiv g^x \pmod{p}.$$[26]

Hur hade Alice kunnat göra något liknande med hjälp av en elliptisk kurva E över $\mathbb{F}_p^*$? Om vi ser g och h som element i gruppen $\mathbb{F}_p$, då skulle det krävas att Eve hittar ett x så att

$$h \equiv \underbrace{g \cdot g \cdot g \cdot g \cdots g}_{x \text{ multiplikationer}} \pmod{p}$$

enligt det diskreta logaritmproblemet. Följaktligen hade Eve behövt lista ut hur många gånger g behöver multipliceras med sig själv för att hitta h. Något vi redan konstaterat verkar vara mycket svårt.[27]

Om vi ser till gruppen av punkter på en elliptisk kurva E över en ändlig kropp $\mathbb{F}_p$ går det att med hjälp av DLP att kryptera, uppläget ser ut som följande.

[27] Hoffstein, Pipher och Silverman, 2014 s.310

Alice väljer P i $E(\mathbb{F}_p)$ och ett heltal n vilket blir hennes privata nyckel. Med denna beräknar hon Q enligt nedan, och publicerar P & Q.

$$Q = \underbrace{P + P + P + \cdots + P}_{n \text{ additioner på } E} = nP.$$

Det som gör kryptering med ECC så säkert är delvis hur vi har definierat addition mellan två punkter. Som vi minns är det en komplicerad process som gör det svårt för Eve om hon vill komma åt informationen.

Definition 2.20 Det diskreta logaritmproblemet för elliptiska kurvor. Låt E vara en elliptisk kurva över det ändliga fältet $\mathbb{F}_p$ och låt P och Q vara punkter i $E(\mathbb{F}_p)$. Det diskreta logaritmproblemet för elliptiska kurvor (ECDLP) är problemet med att hitta ett heltal n så att $Q = nP$. I analogi med det diskreta logaritmproblemet vi beskrivit tidigare (1.8), betecknar vi detta heltal n med

$$n = \log_P(Q).$$

och vi kallar n för den elliptiska diskreta logaritmen av Q med avseende på P.[28]

Elliptisk Diffie-Hellmans nyckelutbyte

Vi kan applicera denna kunskap om elliptiska kurvor och använda till att undersöka hur detta fungerar som krypteringsalgoritm.

Alice och Bob kommer överens om att använda en specifik elliptisk kurva $E(\mathbb{F}_p)$, där p är ett stort primtal vilket de låter vara publikt. Sedan väljer de en specifik punkt $P \in E(\mathbb{F}_p)$. Även här är all information fortsatt öppen för alla men i nästa steg ska de välja varsitt tal de håller hemligt. Alice väljer ett hemligt heltal n_A och Bob väljer ett hemligt heltal n_B. De beräknar de associerade multiplerna

$$Q_A = n_A P \text{ och}$$

$$Q_B = n_B P,$$

och de utbyter värdena på Q_A och Q_B. Själva utbytet är också publik information som även Eve har tillgång till men hon har inte de hemliga talen som använts för att beräkna Q_A samt Q_B. Alice använder sedan sin hemliga multiplikator för att beräkna $n_A Q_B$, och Bob beräknar på liknande sätt $n_B Q_A$. De har nu det gemensamma hemliga värdet $nAQB = (nAnB)P = nBQA$. De har således nu möjlighet att genom ett symmetrisk chiffer kommunicera privat.

Exempel 2.21 Elliptisk Diffie-Hellmans nyckelutbyte.
Alice och Bob bestämmer sig för att använda elliptisk Diffie-Hellman med följande primtal, kurva och punkt:

$$p = 17,\ E : y^2 = x^3 + x + 6,\ P = (7, 13) \in E(\mathbb{F}_{17}).$$

[28] Hoffstein, Pipher och Silverman, 2014 s.310-311

Alice och Bob väljer respektive hemliga värden $n_A = 7$ och $n_B = 14$, och därefter sker följande beräkningar (metoden för att genomföra dessa beräkningar beskrivs senare i 2.25):

$$\text{Alice beräknar } Q_A = 7P = (1, 12) \in E(F_{17}),$$

$$\text{Bob beräknar } Q_B = 14P = (7, 4) \in E(F_{17}).$$

Alice skickar Q_A till Bob och Bob skickar Q_B till Alice. Slutligen,

$$\text{Alice beräknar } n_A Q_B = 7(7, 4) = (1, 5) \in E(F_{17}),$$

$$\text{Bob beräknar } n_B Q_A = 14(1, 12) = (1, 5) \in E(F_{17}).$$

Bob och Alice har nu en gemensam hemlig punkt $(1, 5)$. Anledningen till att de får samma punkt är att de båda genomfört samma beräkning, men utan att veta vilken privat nyckel den andra har använt i beräkningen. Alice får bara produkten av $n_B \cdot P$ utan att veta n_B och multiplicerar denna med n_A, Bob får motsvarande bara produkten av $n_A \cdot P$, utan att veta n_A och multiplicerar denna med n_B. Båda beräknar således $n_A \cdot n_B \cdot P$.
Ett sätt för Eve att upptäcka Alice och Bobs hemlighet är att lösa ECDLP

$$nP = Q_A, \quad \text{eller} \quad nP = Q_B,$$

vilket skulle innebära att hon kan ta reda på n_A eller n_B och på så sätt kan använda det för att beräkna $n_A Q_B$ alternativ $n_B Q_A$.[29]

Kryptering med elliptiska kurvor med Elgamal

Elgamal som vi beskrivit tidigare går att använda för att kryptera med elliptiska kurvor. Det går ut på att Alice och Bob kommer överens om att använda ett specifikt primtal p, en elliptisk kurva E, och en punkt $P \in E(\mathbb{F}_p)$.

Generera nycklar

Alice väljer en hemlig multiplikator n_A och publicerar punkten $Q_A = n_A P$ som sin publika nyckel.

Kryptering

Bobs klartext är en punkt $M \in E(\mathbb{F}_p)$. Han väljer ett heltal k där $1 \leq k < p$ som sitt slumpmässiga element och beräknar

$$C_1 = kP \in E(\mathbb{F}_p) \quad \text{och} \quad C_2 = M + kQ_A \in E(\mathbb{F}_p).$$

Han skickar sedan de två punkterna (C_1, C_2) till Alice.

[29] Hoffstein, Pipher och Silverman, 2014 s.316-318

Dekryptering
När Alice fått de två punkterna från Bob så beräknar hon:

$$C_2 - n_A C_1 = (M + kQ_A) - n_A(kP) = M + k(n_A P) - n_A(kP) = M$$

för att slutligen få tillbaka klartexten.[30]

Normalt sätt används elliptiska kurvor med Elgamal främst för nyckelgenerering samt nyckelutbyte med mycket stark säkerhet och inte nödvändigtvis för att kryptera och dekryptera meddelanden. Men vi kan ta ett förenklat exempel för att visa på själva metoden.

Exempel 2.22 Kryptering med ECC & Elgamal.
Vi väljer ett primtal $p = 11$ och den elliptiska kurvan E: $y^2 = x^3 + x + 6$ över $\mathbb{F}_{11}$. Vi minns från ett tidigare exempel 2.19 att mängden punkter på E med koordinater i $\mathbb{F}_{11}$ är

$$\{\mathcal{O}, (2,4), (2,7), (3,5), (3,6), (5,2), (5,9), (7,2), (7,9), (8,3), (8,8), (10,2), (10,9)\}.$$

Vi väljer punkten $P = (2,4)$ som generator. Denna punkt måste generera gruppen då gruppen har ett primtal element (tack vare Lagrange sats [31]). Sedan väljer Alice en privat nyckel $n_A = 7$ och beräknar sin publika nyckel $Q_A = 7 \cdot P = (7,9)$.

När Bob ska kryptera meddelandet $M = (3,5)$ så väljer han ett först slumpmässigt tal $k = 4$. Sedan beräknar Bob $C_1 = k \cdot P = 4 \cdot (2,4) = (10,9)$ och $C_2 = M + k \cdot Q_A = (3,5) + 4 \cdot (7,9) = (7,9)$. Tillslut skickar Bob $(C_1, C_2) = ((10,9),(7,9))$ till Alice.

Om Alice vill dekryptera meddelandet beräknar hon följande:

$$C_2 - n_A \cdot C_1 = (7,9) - 7 \cdot (10,9) = (3,5)$$

och har således återfått det ursprungliga meddelandet M.

2.3.3 Kryptering av meddelanden som punkter på kurvan

Meddelanden som skickas är ofta i klartext. Som tidigare nämnts används kryptering med elliptisk-Elgamal främst för att skapa och byta nycklar, detta då det inte finns något uppenbart sätt att sätta fast klartexten på en punkt i $E(\mathbb{F}_p)$. Tidigare har det gått enklare att mappa heltal till klartexter i exempelvis RSA eller Elgamal eftersom klartexten är representerad som ett heltal likaså operationerna som utförs. Men detta fungerar inte på samma sätt med elliptiska kurvor då vi först måste mappa klartexten till en punkt på kurvan. Mer specifikt

[30] Hoffstein, Pipher och Silverman, 2014 s.317-318
[31] Denna sats tas ej upp i denna uppsats, men går bland annat att läsa om i Biggs, 2002 s 275 ff

om hur detta går att läsa i *Hoffstein et al.*[32]

Men utöver att sätta fast meddelandet på en viss punkt på kurvan liknar principen vår tidigare beskrivning om kryptering med Elgamal (2.2). Det första steget blir därför att göra om klartexten till en binär representation. Vidare måste den binära representationen finnas med som en x-koordinat i $\mathbb{F}_p$ så att det går att mappa meddelandet till en punkt på kurvan.

Ponera att Bob vill skicka ett meddelande till Alice där x-koordinaten, efter att meddelandet har skrivits om till ett binärt tal, inte finns med bland mängden av punkter på E med koordinater i $\mathbb{F}_p$. Meddelandet kan ändå skickas om Alice och Bob har kommit överens om att tillåta modifiera meddelandet genom att lägga till extra bitar på slutet av den binära representationen. Båda har på förväg bestämt hur många ettor och nollor de vill tillåta att lägga till och sedan efter dekrypteringen tar Alice bort samma antal siffror som de kommit överens om att lägga till, innan hon översätter meddelandet till ett heltal med bas 10 igen.

Exempel 2.23.
Vi väljer samma primtal $p = 11$ och samma kurva E: $y^2 = x^3 + x + 6$ över $\mathbb{F}_{11}$. Vi har även samma generator $P = (2, 4)$ och Alice har återigen den publika nyckeln $(10, 2)$. Men vi ponerar nu att Bob vill skicka meddelandet $m = 4$. Innan meddelandet skickas har Alice och Bob redan kommit överens om att de ska lägga till en extra bit till meddelandet.

Eftersom talet 4 inte finns med som x-koordinat bland mängden punkter så skriver Bob om siffran i binär istället och får att $m = 100$. Sedan kan han testa lägga till 0 och se om det talet finns med bland mängden punkter och hittar han inget sådant tal fortsätter han med att prova 1 (och så vidare, i fall där de tillåts flera 'extra' bitar). Det visar sig att 1000 är lika med 8 i basen 10 och 8 finns som x-koordinat. Bob kan då använda detta för att kryptera med. När Alice sen återfår talet 8 så vet hon att hon ska konvertera det till binär form (1000) och ta bort den sista siffran eftersom det redan var överenskommet sedan innan. Hon får då 100 vilket är 4 i bas 10 och Bobs meddelande. I praktiken då man krypterar mycket större tal tillåts ofta ett tilläg på mellan 30-50 bitar vilket gör att nästan alla meddelanden går att omvandla till ett tal inom mängden.

2.4 RSA och ECC: En teoretisk jämförelse

Ett sätt att jämföra dessa två algoritmer rent teoretiskt är att räkna antalet operationer som utförs under krypteringen. När vi räknar med så stora tal som vi gör under kryptering för att kunna säkerställa att Eve inte ska kunna

[32] Hoffstein, Pipher och Silverman, 2014 s.319

bryta krypteringen krävs långa beräkningar vilka tar mycket datakraft. Därför används metoder inom dessa algoritmer för att korta ner antalet operationer som krävs. Vi kan genom att analysera respektive metod som används till RSA och ECC undersöka vilken av dessa som verkar mer effektiv och således också innebär en högre effektivitet i förhållande till krypteringen.

2.4.1 Fast Powering Algorithm

För RSA används *Fast Powering Algorithm* eller snabba exponentieringsalgoritmen för att beräkna stora exponenter med modulär aritmetik så effektivt som möjligt. De tal som Alice och Bob behöver beräkna är väldigt stora och att beräkna varje operation hade krävt för stor datorkraft. Vill vi beräkna $g^a \pmod{n}$ så kan man göra det genom att upprepade gånger multiplicera g med sig självt på följande sätt:

$$\begin{aligned}
g_1 &\equiv g \pmod{n},\\
g_2 &\equiv g \cdot g_1 \pmod{n},\\
g_3 &\equiv g \cdot g_2 \pmod{n},\\
g_4 &\equiv g \cdot g_3 \pmod{n},\\
g_5 &\equiv g \cdot g_4 \pmod{n},\\
&\vdots\\
g_a &\equiv g^a \pmod{n}.
\end{aligned}$$

Men om a är väldigt stort (vilket det behöver vara för en säker kryptering) så behöver vi utföra beräkningarna på ett annat sätt. Ett alternativ är att använda sig av binär expansion på exponenten a. Det går ut på att vi omvandlar beräkningen av g^a till en serie kvadreringar och multiplikationer istället för genom direkt beräkning genom upprepad multiplikation.[33].

Exempel 2.24.
För att beräkna $6^{158} \pmod{100}$ på ett effektivare sätt kan vi ta hjälp av snabb exponentiering. Vi börjar med att skriva om exponenten som en summa av 2-potenser

$$158 = 2 + 2^2 + 2^3 + 2^4 + 2^8,$$

detta innebär att

$$6^{158} = 6^{2+2^2+2^3+2^4+2^8} = 6^2 \cdot 6^{2^2} \cdot 6^{2^3} \cdot 6^{2^4} \cdot 6^{2^8}.$$

Efter detta skapar vi en tabell där vi räknar ut vad varje potens är modulo 100. Detta är enkelt eftersom varje tar blir kvadraten på föregående.

[33]Hoffstein, Pipher och Silverman, 2014 s.24

Tabell 1: Potenser modulo 100

i	0	1	2	3	4	5	6	7	8
$6^{2^i} \pmod{100}$	6	36	96	16	56	36	96	16	56

Vi använder oss nu av tabellen för att beräkna 6^{158}

$$\begin{aligned} 6^{158} =& 6^2 \cdot 6^{2^2} \cdot 6^{2^3} \cdot 6^{2^4} \cdot 6^{2^8} \\ \equiv& 36 \cdot 96 \cdot 16 \cdot 56 \cdot 56 \pmod{100} \\ \equiv& 56 \pmod{100} \end{aligned}$$

Istället för att behöva utföra massor av operationer kunde vi istället komma fram till samma resultat med mycket färre.

Nu när vi har gått igenom ett exempel följer en mer formell beskrivning av algoritmen, baserad på beskrivningen i *Hoffstein et al.* [34]

Algoritm för snabb exponering
Först beräknas den binära representationen av A som

$$A = A_0 + A_1 \cdot 2 + A_2 \cdot 2^2 + A_3 \cdot 2^3 + \cdots + A_r \cdot 2^r \quad \text{med} \quad A_0, \ldots, A_r \in \{0, 1\},$$

där vi kan anta att $A_r = 1$, detta då A_r är den mest signifikanta biten i den binära representationen av A.

Sedan beräknas potenserna $g^{2^i} \pmod{n}$ för $0 \leq i \leq r$ genom successiv kvadrering,

$$\begin{aligned} a_0 &\equiv g \pmod{n} \\ a_1 \equiv a_0^2 &\equiv g^2 \pmod{n} \\ a_2 \equiv a_1^2 &\equiv g^{2^2} \pmod{n} \\ a_3 \equiv a_2^2 &\equiv g^{2^3} \pmod{n} \\ &\vdots \\ a_r \equiv a_{r-1}^2 &\equiv g^{2^r} \pmod{n}. \end{aligned}$$

Varje term är kvadraten av den föregående, så detta kräver r multiplikationer.

Till sist kan vi beräkna $g^A \pmod{n}$ med formeln

$$\begin{aligned} g^A &= g^{A_0 + A_1 \cdot 2 + A_2 \cdot 2^2 + A_3 \cdot 2^3 + \cdots + A_r \cdot 2^r} \\ &= g^{A_0} \cdot (g^2)^{A_1} \cdot (g^{2^2})^{A_2} \cdot (g^{2^3})^{A_3} \cdots (g^{2^r})^{A_r} \\ &\equiv a_0^{A_0} \cdot a_1^{A_1} \cdot a_2^{A_2} \cdot a_3^{A_3} \cdots a_r^{A_r} \pmod{n}. \end{aligned}$$

[34] Hoffstein, Pipher och Silverman, 2014 s.25-26

Notera att storheterna $a_0, a_1, \ldots, a_r$ beräknades i föregående steg. Genom att slå upp värdena på a_i där exponenten A_i är 1 och multiplicera dem kan vi beräkna produkten. Detta kräver som mest ytterligare r multiplikationer.

Således tar det som mest $2r$ multiplikationer $(\text{mod } n)$ för att beräkna g^A. Eftersom $A \geq 2^r$, ser vi att det tar som mest $2\log_2(A)$ multiplikationer $(\text{mod } n)$ för att beräkna g^A. Så även om A är mycket stor, säg $A \approx 2^{1000}$, är det lätt för en dator att utföra de cirka 2000 multiplikationer som behövs för att beräkna $2^A \pmod{n}$.

2.4.2 Double and Add

Vid kryptering med ECC behöver vi beräkna punkten nP av kända värden n och P. Det uppstår snabbt problem om n är väldigt stort (vilket det oftast är för säker kryptering). Det blir ohållbart att beräkna $P, 2P, 3P...$ för att få nP. Vi kan istället förenkla detta genom att använda en metod som liknar snabb exponering för RSA. Skillnaden är att operationen på en elliptisk kurva är definierad som addition istället för multiplikation, därav kallas den för *double and add*. Innan vi beskriver algoritmen går vi, precis som tidigare, först igenom ett exempel.

Exempel 2.25.
Vi använder kurvan $y^2 = x^3 + x + 6 \in E(\mathbb{F}_{17})$ och väljer $n = 7$ samt punkten $(7, 13)$ från ett tidigare exempel i arbetet (2.21).

Vi skulle kunna uttrycka additionen på det uppenbara sättet

$$7P = P + P + P + P + P + P + P.$$

Men för stora n ser vi att denna punktaddition kommer kräva många operationer. Därför skriver vi istället om n som en binär representation

$$n = 7 = 2^2 + 2^1 + 2^0,$$

och ser att vi istället kan uttrycka additionen som

$$7P = 4P + 2P + P.$$

För att beräkna $7P$ på detta sätt behöver vi med andra ord genomföra två punktdubbleringar och tre additioner. Vilket även i detta exempel endast innebär 5 operationer istället för 7, men för stora n kommer det innebära betydande besparingar i antal operationer.

Vi beräknar dubbleringana som:

$$\begin{aligned} P &= (7, 13) \\ 2P &= P + P = (7, 13) + (7, 13) = (1, 12) \\ 4P &= 2P + 2P = (1, 12) + (1, 12) = (7, 4) \end{aligned}$$

Sedan kan vi således beräkna

$$7P = 4P + 2P + P = (7, 13) + (1, 12) + (7, 4) = (1, 12).$$

Precis som med snabb exponentiering kan vi beskriva detta mer generellt som en algoritm.

Algoritm för double and add

Vi börjar med att skriva om n till en binär representation som tidigare

$$n = n_0 + n_1 \cdot 2 + n_2 \cdot 4 + n_3 \cdot 8 + \cdots + n_r \cdot 2^r \quad \text{med} \quad n_0, n_1, \ldots, n_r \in \{0, 1\}.$$

(Vi antar också som i föregående exempel att $n_r = 1$.) Därefter beräknar vi följande:

$$Q_0 = P, \quad Q_1 = 2Q_0, \quad Q_2 = 2Q_1, \quad \ldots, \quad Q_r = 2Q_{r-1}.$$

Notera att Q_i helt enkelt är dubbelt så mycket som föregående Q_{i-1}, så

$$Q_i = 2^i P.$$

Dessa punkter kallas för 2-potenser av P, och att beräkna dem kräver r dubbleringar. Slutligen beräknar vi nP med högst r ytterligare additioner,

$$nP = n_0 Q_0 + n_1 Q_1 + n_2 Q_2 + \cdots + n_r Q_r$$

Vi kommer att referera till additionen av två punkter i $E(\mathbb{F}_p)$ som en punktoperation. Således är den totala tiden för att beräkna nP högst $2r$ punktoperationer i $E(\mathbb{F}_p)$. Notera att $n \geq 2^r$, så det tar högst $2\log_2(n)$ punktoperationer att beräkna nP. Detta gör det möjligt att beräkna nP även för mycket stora värden på n.

Slutsats teoretisk jämförelse
Vi såg ovan att det tar som mest $2\log_2(A)$ multiplikationer $\pmod n$ för att beräkna g^A, med snabb exponentiering som används för RSA. Samt $2\log_2(n)$ punktoperationer att beräkna nP i $E(\mathbb{F}_p)$ med hjälp av double and add som används för ECC. Båda krypteringsalgoritmer har algoritmer som förkortar antalet operationer drastiskt, på ett likvärdigt sätt. Däremot krävs det en betydligt större nyckelstorlek för RSA (se Tabell 2) för att uppnå samma säkerhet som för ECC, och således kommer operationerna att bli fler.

2.5 RSA och ECC: En praktisk jämförelse

I den praktiska delen kommer vi jämföra nyckelgenerering och inte själva krypteringen som i den teoretiska jämförelsen. För att göra detta har vi skapat ett program i Python som i liten skala kan mäta hastigheten mellan dessa och försöka utröna om vi kan se någon skillnad i effektivitet avseende snabbhet och lagringsutrymme med hänseende till liknande säkerhet.

För att genomföra detta användees Python 3.8 och biblioteket cryptography.hazmat, vilket är en del av cryptography-paketet. En fördel med detta tillvägagångssätt är att implementering av både ECC och RSA stöds av denna modul, för att säkerställa en rättvis jämförelse med minskad risk för skillnader relaterade till olika kodoptimering i olika bibliotek.

Då ECC framförallt används för nyckelgenerering och nyckelutbyte begränsades jämförelsen i programmet till nyckelgenerering i var och en av algoritmerna med tidsmätning för nycklar av olika säkerhetsnivå. *Bit Strenght* eller bit-styrka har använts i jämförelsen för att ge en bild av arbetskraften det kräver att bryta chiffret. Bit-styrka mäts i antal bitar och är en indikation på hur många kombinationer som behöver testas för att bryta krypteringen vid exempelvis bruteforce-attacker. En nyckel med högre bit-styrka tar längre tid och kräver mer resurser att knäcka. En säkerhetsklassning på 80 bit innebär med andra ord att det finns 2^{80} olika kombinationer av ettor och nollor som behöver testas för att bryta krypteringen. I många fall rekommenderas inte längre användning av en bit-styrka på mindre än 112-bit men i denna jämförelse har vi valt att ha med även detta för att kunna bedöma skillnaden i effektivitet hos dessa algoritmer även vid förhållandevis låga säkerhetsnivåer. Rekommenderade kurvor och jämförbara RSA-nyckellängder hämtades från National Institute of Standards and Technology samt Thales.[35]

Datainsamlingen utfördes genom att mäta tiden det tog att generera nyckelpar för RSA och ECC under tio oberoende iterationer. Detta för att generera ett bredare underlag och minska effekten av eventuella avvikelser orsakade av bakgrundsprocesser i operativsystemet. Programmet använder ett antal funktioner som styrs från en main() funktion och som är centrala för utförandet av jämförelsen:

1. generate_rsa_keys(key_size) - genererar ett RSA-nyckelpar baserat på angiven nyckelstorlek.
2. generate_ecc_keys(curve) - genererar ett ECC-nyckelpar baserat på en angiven kurva.

[35] Barker, Elaine. *Recommendation for Key Management: Part 1 – General* (NIST Special Publication No. 800-57 Part 1 Revision 5). National Institute of Standards and Technology. 2020. Tillgänglig: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf` s.54-55 & Thales hemsida. Tillgänglig `https://thalesdocs.com/gphsm/luna/7/docs/network/Content/sdk/using/ecc_curve_cross-reference.htm`

3. print_timings(times, encryption_type) - beräknar medeltiden för nyckelgenerering och skriver ut mätningarna.

RSA-nyckelgenerering involverar genereringen av två stora primtal p samt q som sedan används för att beräkna den publika och privata nyckeln. Den publika exponenten e har valts som 65537 enligt praxis [36]. Den privata exponenten d beräknas sedan som tidigare beskrivet (i 2.9).

ECC-nyckelgenereringen involverar valet av ett slumpmässigt stort tal som privat nyckel och därefter beräkning av den publika nyckeln genom punktmultiplikation på den valda elliptiska kurvan.

Koden mäter tiden det tar att generera dessa nycklar genom att använda 'time.time()' före och efter nyckelgenereringen och jämförelsen mellan dessa ger tiden i sekunder för varje itteration. Koden till programmet finns bifogad i bilaga 1.

Det finns ett antal potentiella begränsningar till detta tillvägagångsätt. Dels kan uträkningen av tiden påverkas både av kodens optimering i biblioteket (exempelvis så kan koden för RSA-nyckelgenerering vara mer optimerad än för ECC även om den kommer från samma bibliotek, eller omvänt). Tidsmätningen kan även påverkas av faktorer så som hur nyligen programmet körts, den undersökande datorns hårdvara, mjukvaruoptimeringar med mera. Jämförelsen fokuserar dessutom enbart på nyckelgenerering och tar inte hänsyn till andra aspekter vid implementeringen av en krypteringsalgoritm, såsom kryptering och dekryptering.

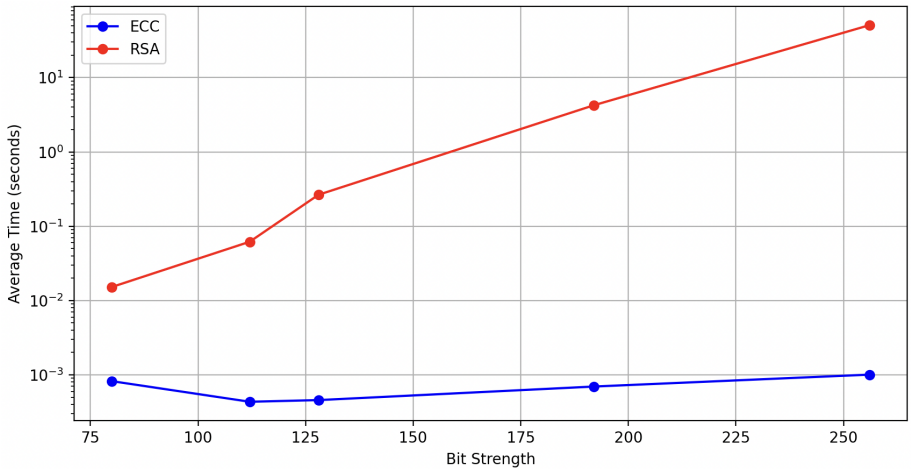
Nedan följer resultaten av undersökningen:

Tabell 2: Jämförelse av bitstyrka och genomsnittlig tid

Bitstyrka	ECC-kurva	ECC snittid (sekunder)	RSA-nyckelstorlek	RSA snittid (sekunder)
80-bitar	SECP192R1	0.000823	1024	0.015249
112-bitar	SECP224R1	0.000435	2048	0.0611816
128-bitar	SECP256R1	0.000458	3072	0.265263
192-bitar	SECP384R1	0.000697	7680	4.247323
256-bitar	SECP521R1	0.001008	15360	50.700948

figur[2.8] Diagram över nyckelgenerering

[36] Ferraiolo, Hildegard. Regenscheid, Andrew *Cryptographic Algorithms and Key Sizes for Personal Identity Verification* (NIST Special Publication No. 800-78-5 ipd). National Institute of Standards and Technology. 2023. Tillgänglig: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-78-5.ipd.pdf`



Vi ser när vi analyserar tabell 2 och kurvorna i figur 2.8 att generering av nycklar med ECC i alla mätningar var snabbare än RSA. Även om båda metoderna var snabba vid lägre bitstyrkor, ökar tiden för nyckelgenerering exponentiellt för längre RSA-nycklar (notera den logaritmiska y-skalan). Däremot är det intressant att ECC vid samtliga styrkor bibehåller effektiviteten och är framförallt är snabbare än RSA. Anledningen till att ECC tycks vara långsammare för en bitstyrka på 80 jämfört med andra högre styrkor kan bero på optimeringsproblem eller slumpmässig avvikelse på grund av det begränsade antalet körningar av varje styrka. För bitstyrkor på 112 och uppåt ser vi att även ECC generellt tar längre tid men inte exponentiellt så som RSA.

Det bör noteras att de teoretiska och praktiska delarna i denna uppsats jämför olika aspekter av algoritmerna. Det finns emellertid tidigare studier som har jämfört säkerheten mot prestandan för RSA och ECC även med avseende på hela algoritmen och som har pekat mot liknande slutsats. [37]. Slutsatsen visar att ECC är mer effektivt än RSA, särskilt när det kommer till hur mycket minne som krävs. Då hårdvara kontinuerligt förbättras och effektiviseras kommer allt högre krav på säkerhet fortsätta att ställas, och betydelsen av effektiva krypteringsalgoritmer fortsätta bli tydligare.

ECC visar sig kräva mindre datakraft och är snabbare än RSA när man jämför med likvärdig säkerhetsnivå även vid praktisk implementering [38]. Vidare ser man även i annan forskning att skillnaden mellan algoritmerna ökar exponentiellt ju större nycklar vi använder, vilket även har betydelse för energieffektivitet. [39] Något som även vi observerade i vår praktiska undersökning.

[37] Gura et al. 2004 *Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs* & Mahto et al. 2017 *RSA and ECC: A comparative analysis*

[38] Gura et al. 2004 *Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs*

[39] El-hajj, Mohammed. *A Comparative Analysis of the Computation Cost and Energy Con-*

Vi kan således dra slutsatsen att för små nycklar är RSA fortfarande en effektiv metod och inte långt efter ECC. I vissa fall kan RSA vara snabbare, beroende på vilken specifik del i kryperingen som undersöks, men säkerhetsmässigt är ECC överlägset [40] RSA kan därför fortfarande vara användbart till att kryptera när det kommer till lägre säkerhetsnivåer. Det är följaktligen en fortsatt välanvänd och säker algoritm än idag. Men då känslig information ska krypteras finns det fördelar med att använda ECC tack vare dess effektivitet kontra styrka. När det kommer till båda dessa algoritmer rekommenderas ingen av dem i implementeringar med lägre bit-styrka än 80, alltså motsvarande RSA-nyckelstorlek på 1024 enligt NISTs senaste riktlinjer. [41]

3 Avslutande diskussion

3.1 Sammanfattning och avslutande diskussion

Genom en initial litteratursammanfattning och djupdykning i matematiken bakom de två algoritmerna har vi undersökt viktiga definitioner och satser som ligger till grund för dem. Med en efterföljande teoretisk och praktisk undersökning har vi vidare analyserat olika aspekter av RSA och ECC vid en direkt jämförelse mot varandra. Både den teoretiska såväl som den praktiska jämförelsen visar på styrkorna hos ECC, vilket även har bekräftats i tidigare forskning. Vi belyser även några svagheter hos båda algoritmerna, där vi i nuläget befinner oss i ett skifte där effektiviteten för RSA fortsatt är acceptabel vid dagligt bruk men där även allt högre krav ställs på säkerhet, prestanda och energieffektivitet, vilket gör att större fokus riktas på ECC med dess fördelar. Användningen av RSA är emellertid bred och väl implementerad, vilket kan medföra att övergången till alternativa krypteringsalgoritmer snarare sker som en gradvis process än ett paradigmskifte.

sumption of Relevant Curves of ECC: International Journal of Electrical and Computer Engineering Research 3(1):1-6, 2023. Tillgänglig: `https://www.ijecer.org/ijecer/article/view/318/28`

[40] Mahto et al. 2017 *RSA and ECC: A comparative analysis*

[41] Barker (NIST Special Publication No. 800-57 Part 1 Revision 5) s.55

4 Referenser

Litteratur

Biggs, Norman L. *Discrete Mathematics*. Oxford University Press. 2002

Hoffstein, Jeffery, Pipher, Jill och Silverman, Joseph H. *An Introduction to Mathematical Cryptography*. Springer Science+Business Media New York: 2014. Tillgänglig: https://link.springer.com/book/10.1007/978-1-4939-1711-2

Källor från internet

Barker, Elaine. *Recommendation for Key Management: Part 1 – General* (NIST Special Publication No. 800-57 Part 1 Revision 5). National Institute of Standards and Technology. 2020. Tillgänglig: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

El-hajj, Mohammed. *A Comparative Analysis of the Computation Cost and Energy Consumption of Relevant Curves of ECC*: International Journal of Electrical and Computer Engineering Research 3(1): 2023. Tillgänglig: https://www.ijecer.org/ijecer/article/view/318/28

Ferraiolo, Hildegard. Regenscheid, Andrew *Cryptographic Algorithms and Key Sizes for Personal Identity Verification* (NIST Special Publication No. 800-78-5 ipd). National Institute of Standards and Technology. 2023. Tillgänglig: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-78-5.ipd.pdf

Gura, Nils & Patel, Arun & Wander, Arvinderpal & Eberle, Hans & Shantz, Sheueling. (2004). *Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs*. Lect Notes Comput Sci. 3156. 119-132. Tillgänglig: https://link.springer.com/chapter/10.1007/978-3-540-28632-5_9

Hasan, Harady, *Elliptic Curves: A journey through theory and its applications* Uppsala universitet: 2019. Tillgänglig: https://www.diva-portal.org/smash/get/diva2:1334316/FULLTEXT01.pdf

Mahto, Dindayal & YADAV, DILIP. (2017). *RSA and ECC: A comparative analysis*. International Journal of Applied Engineering Research. 12. 9053-9061. Tillgänglig: https://www.researchgate.net/publication/322558426_RSA_and_ECC_A_comparative_analysis

Thales hemsida. Tillgänglig https://thalesdocs.com/gphsm/luna/7/docs/network/Content/sdk/using/ecc_curve_cross-reference.htm

5 Bilaga 1 - Python-kod

```python
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa, ec
from cryptography.hazmat.primitives import serialization
import time

def generate_rsa_keys(key_size):
    """
    Genererar ett RSA-nyckelpar baserat på den angivna nyckelstorleken.
    Returnerar privat och publik nyckel.
    """
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=key_size,
        backend=default_backend()
    )
    public_key = private_key.public_key()
    return private_key, public_key

def generate_ecc_keys(curve):
    """
    Genererar ett ECC-nyckelpar baserat på den angivna kurvan.
    Returnerar privat och publik nyckel.
    """
    private_key = ec.generate_private_key(
        curve,
        default_backend()
    )
    public_key = private_key.public_key()
    return private_key, public_key

def main():
    """
    Huvudfunktion för att köra programmet.
    Tillåter användaren att välja krypteringstyp (RSA eller ECC),
    ange nyckelstorlek eller kurva och utföra tidsmätningar.
    """
    while True:
        choice = input("Välj krypteringstyp (RSA/ECC) eller 'avsluta' för att avsluta: ")
        if choice.lower() == 'avsluta':
            print("Avslutar programmet.")
            break
        elif choice.lower() == 'rsa':
            key_size = int(input("Ange RSA-nyckelstorlek (t.ex. 2048): "))
            times = []
            for _ in range(10):  # Utför 10 iterationer för att få ett genomsnitt
                start_time = time.time()
                generate_rsa_keys(key_size)
                end_time = time.time()
                times.append(end_time - start_time)  # Registrera tiden för varje iteration
            print_timings(times, "RSA")
```

```
        elif choice.lower() == 'ecc':
            curve_name = input("Ange ECC-kurva (t.ex. SECP256R1, SECP384R1): ")
            curve = getattr(ec, curve_name.upper())()  # Hämta kurva baserat på namnet
            times = []
            for _ in range(10):  # Utför 10 iterationer för att få ett genomsnitt
                start_time = time.time()
                generate_ecc_keys(curve)
                end_time = time.time()
                times.append(end_time - start_time)  # Registrera tiden för varje iteration
            print_timings(times, "ECC")
        else:
            print("Ogiltigt val.")

def print_timings(times, encryption_type):
    """
    Skriver ut tidmätningarna för varje iteration och beräknar genomsnittlig tid.
    """
    print(f"\nTidsmätningar för varje {encryption_type}-operation:")
    print("-----------------------------------------")
    print("| Iteration | Tid (sekunder)            |")
    print("-----------------------------------------")
    for index, timing in enumerate(times):
        print(f"| {index + 1:<9} | {timing:.6f}                |")
    print("-----------------------------------------")
    average_time = sum(times) / len(times)  # Beräkna genomsnittlig tid
    print(f"Genomsnittlig tid: {average_time:.6f} sekunder")
    print("-----------------------------------------")

if __name__ == "__main__":
    main()
```

Listing 1: Python-kod för den praktiska jämförelsen