Cryptografie met elliptische krommen

L.Y. Pan

TU Delft

Cryptografie met elliptische krommen

Elliptic curve cryptography

door

Li Yong Pan

als onderdeel ter verkrijging van de graad van

BACHELOR OF SCIENCE

in

TECHNISCHE WISKUNDE

aan de Technische Universiteit Delft,
in het openbaar te verdedigen op maandag 16 juli 2018 om 10:30 uur.

Studentnummer:	4466411	
Projectduur:	1 april, 2018 – 16 juli, 2018	
Thesis commissie:	Dr. J. G. Bosman,	TU Delft, supervisor
	Dr. J. G. Spandaw,	TU Delft
	Dr. J. L. A. Dubbeldam,	TU Delft

Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Delft Institute of Applied Mathematics

TU Delft

Voorwoord

Deze bachelor thesis is gemaakt als onderdeel van mijn bacheloropleiding Technische Wiskunde aan de TU Delft.

Gedurende het project hebben een aantal mensen mij enorm veel gesteund en waren zeer behulpzaam. Vandaar dat ik de volgende personen graag wil bedanken: Dr. J.G. Bosman voor zijn toewijding tijdens de begeleiding en het willen uitleggen van wiskundige stof die ik nog niet begreep en het nalezen van de conceptversies van dit verslag; Dr. J.G. Spandaw voor de tijd die hij heeft vrijgemaakt om mij te helpen met het voorbereiden van een wiskundige presentatie voor onder andere het algemene publiek, maar ook voor zijn interesse tijdens de ontmoetingen met Dr. J.G. Bosman; Li Jie Pan voor zijn geduld en hulp bij het perfectioneren van de lay-out en algemene verzorging van het verslag.

Daarnaast zou ik de volledige commissie willen bedanken: Dr. J.G. Bosman, Dr. J.G. Spandaw en Dr. J.L.A. Dubbeldam voor hun interesse en tijd om mijn werk te evalueren. Ten slotte wil ik mijn familieleden en vrienden bedanken voor de steun tijdens het project.

L.Y. Pan
Delft, Juli 2018

Abstract

Met de huidige digitalisering van onder meer bankgegevens is de waarborging van privacy en persoonlijke gegevens zeer belangrijk geworden. Wiskundige objecten genaamd elliptische krommen bieden tot op heden wegens hun algebraïsche structuur een uitstekende manier om online gevoelige informatie veilig uit te wisselen, omdat er voorlopig geen efficiënte algoritmen zijn om alle zogeheten ECDLP's te kraken. Het doel van dit verslag is om te onderzoeken hoe deze elliptische krommen worden toegepast voor deze waarborging en de methodiek hiervan toegepast op eerder gepubliceerde cryptografische methoden zoals Diffie-Hellman en Digital Signature Algorithm. De veiligheid van een systeem gebaseerd op ECC wordt vergeleken met het bekende RSA cryptosysteem en er zal getest worden welke elliptische krommen ongeschikt zijn door middel van implementaties in SageMath, om vervolgens te concluderen welke krommen geschikt zijn voor cryptografisch gebruik.

Inhoudsopgave

1 Introductie **1**

2 Elliptische krommen **3**

2.1 Definitie: Weierstrass vergelijking en visualisatie . . . 3
2.2 Groepsstructuur: groep met bewerking + . . . 4
2.2.1 Optelling van punten . . . 5
2.2.2 Het neutrale element ∞ . . . 7
2.2.3 Groepsaxioma's . . . 8
2.2.4 nP berekenen . . . 9
2.2.5 Torsiepunten . . . 10
2.3 Elliptische krommen over eindige lichamen . . . 10
2.4 Afbeeldingen: pairings en de Frobenius . . . 12
2.4.1 Weil pairing . . . 12
2.4.2 Frobenius . . . 13
2.4.3 Tate pairing . . . 15

3 Cryptografie **17**

3.1 Enkele zwakke cryptosystemen . . . 17
3.2 DH, RSA en DSA . . . 19
3.2.1 Diffie-Hellman . . . 19
3.2.2 Rivest-Shamir-Adleman . . . 23
3.2.3 Digital Signature Algorithm . . . 29
3.3 ECC . . . 32
3.3.1 ECDH . . . 32
3.3.2 ECDSA . . . 34
3.3.3 ECC tegenover RSA . . . 35
3.4 Algoritmes voor DLP's . . . 36
3.4.1 Brute-force . . . 36
3.4.2 Baby-step Giant-step . . . 37
3.4.3 Pollard's rho . . . 37
3.4.4 Pohlig-Hellman . . . 38
3.4.5 Index Calculus . . . 39
3.5 Speciale methoden voor ECDLP . . . 41
3.5.1 MOV . . . 41
3.5.2 Frey-Rück . . . 42
3.5.3 Smart . . . 42

4 Voorbeelden van het kraken van ECDLP's **45**

4.1 Complexe vermenigvuldiging (CM) . . . 45
4.2 ECDLP: Pohlig-Hellman . . . 47
4.3 ECDLP: MOV . . . 48
4.4 ECDLP: Smart . . . 51

5 Conclusie en toepassingen in de praktijk **55**

Bibliografie **59**

1 Introductie

Dit verslag zal opbouwen naar het gebruik van elliptische krommen in de cryptografie. Hiertoe zullen in Hoofdstuk 2 enkele definities en de structuur van elliptische krommen behandeld worden. De informatie over elliptische krommen komt voornamelijk uit Washington [31]. Vervolgens zal in Hoofdstuk 3 een algemeen idee en de achterliggende gedachten van cryptografie aan bod komen met informatie uit Hankerson et al. [12], Menezes et al. [20] en Koblitz [16], gecombineerd met een aantal voorbeelden om daarna deze kennis te samen te voegen tot cryptografie met elliptische krommen. Daar zullen we het hebben over hoe elliptische krommen worden gebruikt in de praktijk; onder andere de werking van ECC in een mobiele telefoon, WhatsApp en de Bitcoin. In Hoofdstuk 4 zal er gekeken worden naar implementaties van algoritmes die het discrete logaritme probleem kunnen kraken, waarbij elliptische kromme worden geconstrueerd met complexe vermenigvuldiging. Ten slotte zal in Hoofdstuk 5 gekeken worden naar de voorwaarden voor een bruikbare elliptische kromme voor de cryptografie die deels gebaseerd zijn op eigen voorbeelden van Hoofdstuk 4. Daarbij zal het centrale idee van dit verslag zich richten op de volgende vraagstelling:

Hoe worden elliptische krommen gebruikt in de cryptografie?

Daartoe worden deelvragen impliciet beantwoord die opbouwen naar de uiteindelijke vraagstelling. De eerste deelvraag kan gelinkt worden aan Hoofdstuk 2 en de laatste deelvragen kunnen gelinkt worden aan Hoofdstuk 3. In Hoofdstuk 4 en Hoofdstuk 5 zijn er geen echte corresponderende vragen, omdat deze meer zijn bedoeld om te zien hoe het kraken van ECDLP's te werk gaat over ongeschikte krommen respectievelijk voorwaarden en toepassingen in de praktijk toelicht.

1. Wat zijn de eigenschappen van elliptische krommen?
2. Wat is cryptografie?
3. Hoe kan de kennis over elliptische krommen gecombineerd worden met cryptografie?

2

Elliptische krommen

2.1. Definitie: Weierstrass vergelijking en visualisatie

Om te beginnen zullen we proberen om een algemeen concept te ontwikkelen over elliptische krommen. We kunnen beginnen met een definitie over de reële getallen $\mathbb{R}$. Dan kunnen we een elliptische kromme E voorstellen als de verzameling van punten (x, y) in $\mathbb{R}^2$

$$E = \{(x, y) \in \mathbb{R}^2 \mid y^2 = x^3 + ax + b)\} \cup \{\infty\},$$

waar $a, b \in \mathbb{R}$ en waarbij ∞ een punt is waar we later nog op terug zullen komen. Vergelijkingen van de vorm

$$y^2 = x^3 + ax + b \tag{2.1}$$

worden *Weierstrassvergelijkingen* genoemd. Voor een ander lichaam dan $\mathbb{R}$ is deze vergelijking niet per se de algemene vergelijking voor elliptische krommen. Als er gewerkt wordt in een lichaam met karakteristiek 2 of 3, dan is er een algemene vergelijking te noteren

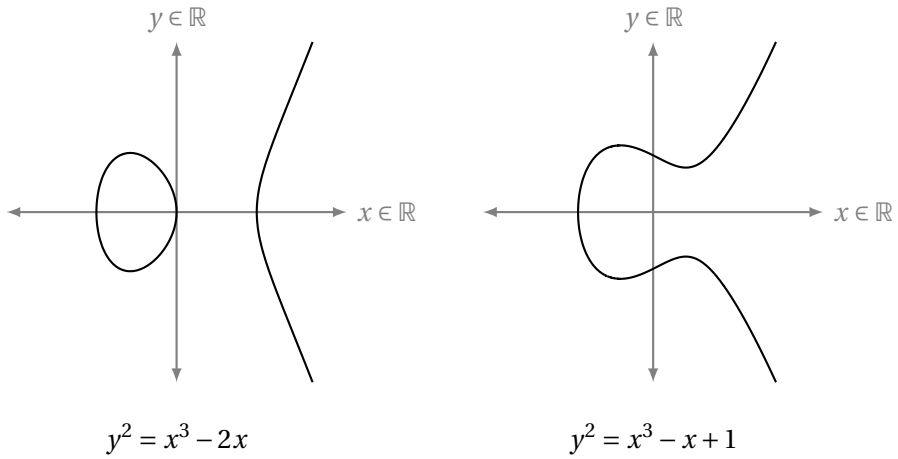
$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{2.2}$$

met $a_1, a_2, a_3, a_4, a_6 \in \mathbb{R}$. Voor een lichaam met karakteristiek ongelijk aan 2 en 3 is het middels coördinatentransformatie mogelijk om van Vergelijking (2.2) terug te werken naar Vergelijking (2.1). We zullen verder in dit verslag voornamelijk werken met de kortere vergelijking puur voor de eenvoud. Met bovenstaande definitie zien we dus dat een elliptische kromme gezien kan worden als een grafiek in het platte vlak. Het is dus heel goed mogelijk om deze te visualiseren, zoals in Figuur 2.1.

Kijkende naar Figuur 2.1, zien we dat deze elliptische krommen symmetrisch zijn in de x-as. Dat betekent: als een punt op de kromme (x_P, y_P) ligt, dan ligt de spiegeling van dit punt ten opzichte van de x-as, $(x_P, -y_P)$, ook op de kromme. Dit gaat echter niet alleen op voor deze specifieke keuzes voor a, b. Aan de vorm van de vergelijking was al te zien dat dit voor elke kromme gedefinieerd als in Weierstrass vorm (2.1) geldt. Aangezien $(y_P)^2 = y_P^2 = (-y_P)^2$, geldt er ook dat $(-y_P)^2 = x_P^3 + ax_P + b$, zoals we wilden aantonen. We zullen deze eigenschap van symmetrie in de x-as in § 2.2 gebruiken om de groepsstructuur te definiëren.

Bovendien is in Figuur 2.1 te zien dat er onderscheid gemaakt kan worden tussen het type kromme. Logischerwijs hangt dit af van de keuzes voor a en b, aangezien deze parameters de kromme typeren. Deze zijn weer direct verbonden met het aantal nulpunten van de krommen, dat wil zeggen, de waarden $x_1, x_2, x_3 \in \mathbb{R}$ zodanig dat

$$x^3 + ax + b = (x - x_1)(x - x_2)(x - x_3) = 0$$



Figuur 2.1: Twee krommen met verscheidene waarden voor a, b.

Wegens de Hoofdstelling van de Algebra is de eerste identiteit hierboven welgedefinieerd. Aangezien we op dit moment nog in $\mathbb{R}$ werken, is het alleen mogelijk om precies één of drie *reële* nulpunten te hebben. Inderdaad, als $z \in \mathbb{C}$ een nulpunt is van een functie f, dan geldt dat $\bar{z}$ ook een nulpunt is van f. Hierdoor liggen twee *complexe* nulpunten van de totale punten x_1, x_2, x_3 vast en is het alleen nog maar mogelijk om één reëel nulpunt te hebben. Er zijn echter nog restricties wat betreft de uniciteit van de nulpunten. Uniciteit voor het geval dat er maar één reëel nulpunt is, mag duidelijk zijn. Het geval dat er drie reële nulpunten zijn is een lastiger 'probleem' om te bekijken. Uiteindelijk is het punt dat E zodanig gedefinieerd wordt dat er geen samenvallende nulpunten zijn, dus dat er geen nulpunten zijn met multipliciteit groter dan 1.

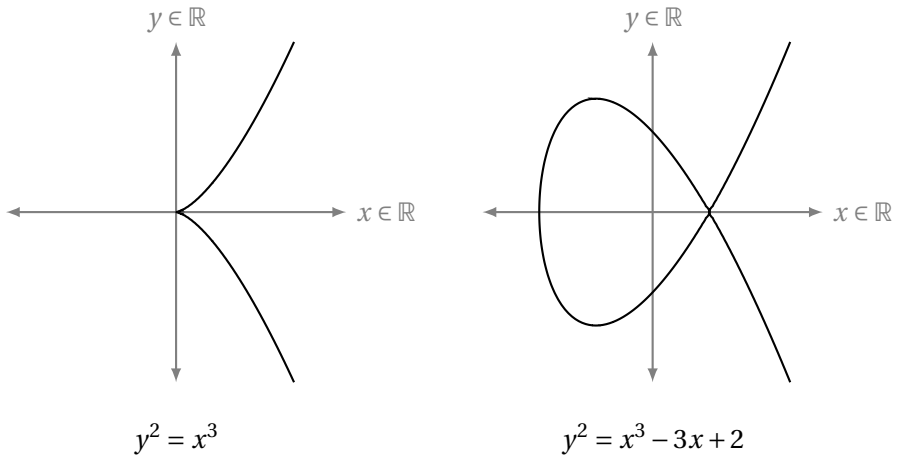
Het punt is dat we voor de groepsbewerking die we zometeen willen definiëren op E gebruik zullen maken van lijnen die de krommen snijden en in het bijzonder raaklijnen aan de krommen. Als x_1, x_2, x_3 niet uniek zijn, wil dat zeggen dat één van deze punten een multipliciteit heeft groter dan 1. Voor de grafiek betekent het dat de kromme niet *glad* is op dat punt. De raaklijn in singuliere punten is dan niet gedefinieerd. Als we kijken naar de algebra van dit probleem, kunnen we kijken voor welke waarden van a, b dit precies het geval is. Net als bij polynomiale vergelijkingen van graad 2 is er in dit geval een *discriminant* te definiëren. In ons geval voor $x^3 + ax + b = 0$ wordt de discriminant D gedefinieerd als

$$D = -16(4a^3 + 27b^2).$$

Er geldt dat de rechterkant van de Weierstrassvergelijking geen nulpunten heeft van mulitpliciteit hoger dan 1 dan en slechts dan als $D \neq 0$. Slechts beperkt op a, b betekent dit dat een kromme van de vorm (2.1) een elliptische kromme is als $4a^3 + 27b^2 \neq 0$. De afleiding van deze discriminant is voor dit verslag niet belangrijk, maar voor de geïnteresseerde lezer kan deze gevonden worden bij [15]. In Figuur 2.2 zijn twee voorbeelden te zien van krommen met singulariteiten; het is na te gaan dat in deze gevallen geldt dat $D = 0$, waardoor dit dus géén elliptische krommen zijn. Zulke krommen heten *singuliere* krommen. Op hun singuliere punten na zijn er toch nog een aantal overeenkomende eigenschappen met elliptische krommen. Voor de geïnteresseerde lezer verwijzen we naar Washington [31].

2.2. Groepsstructuur: groep met bewerking +

We werken voor nu nog steeds in $\mathbb{R}$ en kiezen E met als Vergelijking (2.1). Voor de toepassingen in de cryptografie willen we dergelijke termen als $3Q, 10R$ en $-4S$ definiëren, waarbij $Q, R, S \in E$. Sterker nog, we zijn geïnteresseerd naar het algemene geval nP met $n \in \mathbb{N}$ en $P \in E$. Daartoe zullen we kijken naar hoe we punten op E kunnen 'optellen'. Hieruit zal volgen dat de punten op E waaronder



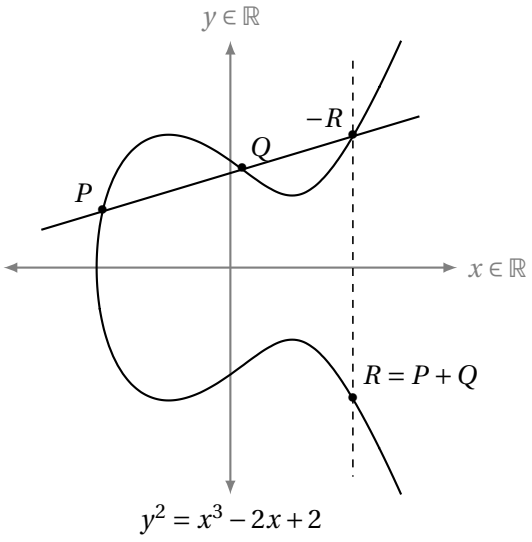
Figuur 2.2: Twee voorbeelden van krommen met singulariteiten; géén elliptische krommen.

∞ een groep vormen met de bewerking + die we nu introduceren.

2.2.1. Optelling van punten

Laat $P = (x_P, y_P)$ en $Q = (x_Q, y_Q)$ twee punten zijn op E. We zullen een nieuw punt R, de optelling van P en Q, definiëren op de volgende meetkundige wijze en wordt in Figuur 2.3 geïllustreerd.

1. Teken de lijn k door P en Q.
2. Noteer het snijpunt van deze lijn en de kromme als $-R$.
3. Spiegel het snijpunt in de x-as.
4. Verkrijg het punt R.



Figuur 2.3: Optelling van punten P en Q meetkundig weergegeven.

Er zijn een aantal vragen die we kunnen stellen over deze meetkundige werkwijze. Laten we eenvoudig beginnen. Bestaat de spiegeling van $-R$, namelijk R, überhaupt wel? Het antwoord daarop is: ja, we hebben in § 2.1 gezien dat E symmetrisch is. Nu een lastigere vraag: snijdt de lijn k de

kromme E altijd? Dat hangt van een aantal factoren af. Er zijn een aantal gevallen die we moeten onderscheiden, dus laten we dit keer beginnen met het meest interessante geval: $P,Q \neq \infty$ en $x_P \neq x_Q$. Dit geval is het meest generiek en hieruit kunnen ook de andere gevallen beschreven worden.

Voor de volgende gevallen zullen we gebruik maken van de aanpak in Washington [31]. De vergelijking van de lijn k kan gegeven worden door

$$k : y = m(x - x_P) + y_P \quad \text{met} \quad m = \frac{y_Q - y_P}{x_Q - x_P}.$$

Merk op dat m welgedefinieerd is vanwege de keuze in $\mathbb{R}$, maar ook voor andere lichamen is dit welgedefinieerd. Voor de snijpunten van k en E zijn we eigenlijk op zoek naar de oplossingen voor de vergelijking

$$(m(x - x_P) + y_P)^2 = x^3 + ax + b \implies x^3 - m^2x^2 + \cdots = 0.$$

Het zal blijken dat we enkel de coëfficiënt van x^2 nodig hebben. Aangezien we twee van de drie nulpunten kennen, namelijk x_P en x_Q, kunnen we een derdegraads polynoom van de vorm $x^3 + Ax^2 + Bx + C$ omschrijven als volgt:

$$x^3 + Ax^2 + Bx + C = (x - x_P)(x - x_Q)(x - x_{-R}) = x^3 - (x_P + x_Q - x_{-R})x^2 + \cdots.$$

Daaruit blijkt dus dat $-A = x_P + x_Q + x_{-R}$. Dit alles wil zeggen dat het achterhalen van een derde nulpunt voor een derdegraadsvergelijking eenvoudig gaat indien twee nulpunten al bekend zijn. Voor ons geval geldt $-A = m^2$, waaruit met $x_{-R} = x_R$ en $-y_{-R} = y_R$ volgt dat

$$x_R = m^2 - x_P - x_Q \tag{2.3a}$$
$$y_R = m(x_P - x_R) - y_P. \tag{2.3b}$$

Ons doel was om optelling te definiëren van P en Q en te laten zien dat het snijpunt $-R$ daadwerkelijk bestaat. Dat hebben we nu gedaan door de cöordinaten van R te hebben berekend, waarbij alles netjes is gedefinieerd. De notatie die we zullen gebruiken voor optelling van P en Q is: $P + Q = R$, met $R = (x_R, y_R)$ en gedefinieerd zoals in Vergelijking (2.3).

Het vorige geval was het meest generieke geval, maar om de punten een groep te laten vormen, moet ook het geval dat $P = Q$ welgedefinieerd zijn. Als we het meetkundige beeld weer voor ons halen, dan is het geval $P = Q$ eigenlijk te vergelijken met de helling van de grafiek op één punt. Dat wil zeggen, de lijn tussen twee punten op de kromme die dicht bij elkaar liggen benadert de raaklijn en zal uiteindelijk gelijk zijn aan de raaklijn aan de kromme in het punt P (of Q). Voor de helling van deze raaklijn differentiëren we Vergelijking (2.1) impliciet waaruit volgt dat

$$2y\frac{\mathrm{d}y}{\mathrm{d}x} = 3x^2 + a \implies m = \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{3x_P^2 + a}{2y_P}.$$

Voor het geval dat $y_P = 0$ zou de raaklijn verticaal zijn en zou deze de kromme snijden in het punt ∞. Zoals net gezegd zijn de uitdrukkingen in (2.3) welgedefinieerd, dus is het hier ook mogelijk om simpelweg te substitueren, waarvoor $P + Q = 2P$ in het geval $P = Q$, $y_P \neq 0$ geldt dat

$$x_{2P} = m^2 - 2x_P \tag{2.4a}$$
$$y_{2P} = m(x_P - x_{2P}) - y_P. \tag{2.4b}$$

Net hebben we al stilletjes eigenschappen van het element ∞ benoemt en de definitie van elliptische kromme heeft daar kennelijk iets mee te maken. Daar zullen we nu verder op in gaan.

2.2.2. Het neutrale element ∞

Herinner de manier waarop we ∞ hebben geïntroduceerd, namelijk als een element toegevoegd aan de oplossingsverzameling voor de kromme E

$$E = \{(x,y) \in \mathbb{R}^2 \mid y^2 = x^3 + ax + b)\} \cup \{\infty\}$$

Dit zegt natuurlijk niets over de conceptuele betekenis achter ∞. Net claimden we al dat bij een verticale raaklijn aan de kromme, de raaklijn de kromme zou snijden in het punt ∞. Dit helpt bij het definiëren van enkele berekeningen. Daaruit zouden we dus af kunnen leiden dat ∞ ergens bovenaan de y-as zit. Vanwege de symmetrie zou dat ook betekenen dat dit punt ergens onderaan de y-as zit. Dat klinkt tegenstrijdig om te eisen, maar als we met andere (eindige) lichamen werken en niet met $\mathbb{R}$ – en dat zullen we ook doen – dan is het concept van 'bovenaan' en 'onderaan' en de ordening van getallen niet meer zinvol. Een andere eigenschap is dat elke verticale lijn het punt ∞ snijdt. Net hebben we géén voorwaarden opgelegd aan de verticale lijnen, dus dat betekent in dit geval dat parallelle lijnen elkaar zouden snijden(!). Dit impliceert weer dat we misschien ∞ niet als een punt moeten zien bovenaan/onderaan de y-as, maar als een lijn op oneindig die bovenaan/onderaan zit.

Via projectieve meetkunde zouden we dit concept goed kunnen praten. We zullen daar in dit verslag kort op ingaan, maar voor de geïnteresseerde lezer verwijzen we naar [15, 31]. Hierbij gebruiken we de definities uit Washington. Voor K een lichaam kunnen we de twee-dimensionale *projectieve ruimte* $\mathbb{P}^2_K$ bekijken, die wordt samengesteld uit equivalentieklassen van *drietallen* (x,y,z) waarvoor $x,y,z \in K$. Hierbij geldt dat minstens één van de x,y,z ongelijk is aan 0. We noemen twee drietallen equivalent indien er een $0 \neq \lambda \in K$ bestaat zodanig dat

$$(x_1,y_1,z_1) = (\lambda x_2, \lambda y_2, \lambda z_2) \qquad x_i, y_i, z_i \in K \quad i = 1,2.$$

Bijvoorbeeld: de drietallen $(1,1,1)$ en $(2,2,2)$ zijn equivalent met $2 = \lambda \in K$ (aannemende dat $2 \in K$). Omdat de equivalentieklasse voor een drietal uniek bepaald wordt door de verhouding $x : y : z$ hanteren we de notatie $(x : y : z)$. Als we nu spelen met de verhoudingen, kunnen we kijken naar de equivalentieklasse van $(x : y : 0)$. Deze is netjes gedefinieerd, omdat minstens twee variabelen ongelijk is aan 0. Elementair kunnen we dus zeggen dat de verhoudingen van x en y tot z 'oneindig' groot is, omdat

$$\lim_{z\to 0} \frac{x}{z} = \infty \quad \text{en} \quad \lim_{z\to 0} \frac{y}{z} = \infty.$$

Vandaar dat we deze equivalentieklasse kunnen identificeren als de 'punten op oneindig' in de twee-dimensionale projectieve ruimte. Voor het element ∞ dat we aan onze kromme willen toevoegen, moeten we gebruik maken van de *homogene* projectieve vorm van de Weierstrassvergelijking. Dat is de volgende vorm

$$y^2 z = x^3 + axz^2 + bz^3. \tag{2.5}$$

Homogeen in dit geval betekent dat voor elke term de som van alle exponenten gelijk is aan n, waarbij $n = 3$ in ons geval. De keuze voor $n = 3$ lag vast, omdat x^3 de term was met hoogste macht. We kunnen de punten (x,y) op E in het *affiene* vlak identificeren aan $(x : y : 1)$ voor het projectieve vlak. Om het gedrag te bekijken van punten op E die op oneindig liggen, stellen we $z = 0$ en volgt uit Vergelijking (2.5) dat $x^3 = 0$. Dat impliceert dat $x = 0$ en we verkrijgen het punt $(0 : y : 0)$. Aangezien er minstens één van de x,y,z ongelijk moest zijn aan 0, betekent dat $(0 : 0 : 0)$ geen mogelijkheid is. Als we nu schalen met y – dat mag omdat $y \in K$ – zien we dat $(0 : y : 0) = (0 : 1 : 0)$ het enige punt op oneindig ligt voor E. Bovendien ligt $(0 : 1 : 0)$ op elke verticale lijn en zien we dat wegens de definitie van equivalentie dat $(0 : 1 : 0) = (0 : -1 : 0)$, ofwel het punt op oneindig 'bovenaan' en 'onderaan' zijn identiek.

Nu we het element ∞ hebben behandeld, kunnen we de informatie van optelling van punten $P + Q = R$ samenvatten in Tabel 2.1.

Tabel 2.1: Optelling $(x_R, y_R) = R = P + Q$ van $P = (x_P, y_P)$ en $Q(x_Q, y_Q)$

	$P,Q \neq \infty$ $x_P \neq x_Q$	$P = Q \neq \infty$ $y_P \neq 0$	$P = Q \neq \infty$ $y_P = 0$	$P,Q \neq \infty$ $x_P = x_Q, y_P \neq y_Q$
m	$\frac{y_Q - y_P}{x_Q - x_P}$	$\frac{3x_P^2 + a}{2y_P}$	verticaal	verticaal
x_R	$m^2 - x_P - x_Q$	$m^2 - 2x_P$	∞	∞
y_R	$m(x_P - x_R) - y_P$	$m(x_P - x_{2P}) - y_P$	∞	∞

2.2.3. Groepsaxioma's

Nu we een oplossingsverzameling E hebben gedefinieerd met een welgedefinieerde bewerking $+$, resteert ons nu slechts om te laten zien dat $(E, +)$ daadwerkelijk een groep vormt. Hiervoor moeten we de volgende axioma's langsgaan:

1. *Geslotenheid*
   Geslotenheid wil zeggen dat de bewerking op twee elementen van de verzameling óók in de verzameling zit. Wiskundig zeggen we dus dat voor $P,Q \in E$ geldt dat $P + Q \in E$. Het mag duidelijk zijn dat dit geldt. We verwijzen naar Tabel 2.1 voor de precieze optelling en merken op dat deze allen gedefinieerd zijn.

2. *Identiteitselement*
   Het identiteitselement is het element van de verzameling die elk ander element stuurt naar zichzelf. In ons geval is dat uiteraard ∞ en dat moet betekenen dat voor $P \in E$ geldt dat $P + \infty = P$. Meetkundig is dit eenvoudig na te gaan: de lijn door P en ∞ snijdt de kromme in $-P$. Met spiegeling door x-as vinden we dus als som P. Ook volgens de formules kan dit bevestigd worden.

3. *Bestaan van inverses*
   Een inverse van een element wil zeggen dat er voor een punt $P \in E$ er een element $Q \in E$ bestaat zodanig dat $P + Q = \infty$. Ook deze eigenschap is triviaal, aangezien $Q = -P$ gemakkelijk voldoet. Meetkundig: de lijn door P en $-P$ snijdt de kromme in ∞. De spiegeling is tevens weer ∞. Bovendien bestaat $-P$ altijd vanwege de symmetrische eigenschap van elliptische krommen.

4. *Associativiteit*
   Associativiteit betekent dat we de volgorde van de herhaalde bewerking geen invloed heeft op het uiteindelijke resultaat. Dus voor $P,Q,R \in E$ geldt dat $(P + Q) + R = P + (Q + R)$. Dit geval is het lastigst te bewijzen, maar kan gedaan worden op enkele manieren. In dit verslag zullen we daar niet verder op in gaan, maar suggereren we enkele mogelijkheden om dit te doen. Om te beginnen is het met 'boerenkracht' te doen via formules van optellingen. Ook is het mogelijk dit te bewijzen door middel van een beetje projectieve meetkunde, singulariteit van krommen en partiële afgeleiden van homogene polynomen. Uiteindelijk is het nog mogelijk om dit aan te tonen met behulp van divisorgroepen (Picardgroepen) van een kromme en dit is wellicht het meest conceptuele bewijs van deze drie. Voor nu zullen we alleen deze mooie bewijzen gebruiken om te concluderen dat $(E, +)$ inderdaad associatief is.

Zelfs nu we hebben geconcludeerd dat $(E, +)$ inderdaad een groep is, kunnen we nog één mooie eigenschap van deze groep laten zien. Dat is namelijk dat de groep $(E, +)$ een *abelse* groep is. Voor deze groep geldt namelijk dat de volgorde van optelling van twee punten $P,Q \in E$ niet uitmaakt (commutativiteit):

$$P + Q = Q + P.$$

Dit is net als de eerste drie axioma's eenvoudig na te gaan. Volgens de formules is dit met de hand te bewijzen, maar we kiezen voor de meetkundige redenering. Herinner de meetkundige werkwijze:

1. Teken de lijn k door P en Q.
2. Noteer het snijpunt van deze lijn en de kromme als $-R$.
3. Spiegel het snijpunt in de x-as.
4. Verkrijg het punt R.

Het maakt in stap 1 voor de lijn niet uit of we de posities van P en Q in de zin van optelling verwisselen, omdat de lijn door deze twee punten altijd gelijk zal zijn. De vervolgstappen worden hierdoor dus ook niet beïnvloed en we concluderen dat de groep inderdaad abels is.

Zoals eerder gezegd is voor de cryptografie de optelling erg interessant indien je een punt bij zichzelf optelt. We zien de eerder genoemde termen $3Q, 10R$ en $-4S$ dan ook niet als een vermenigvuldiging, maar als volgt:

$$3Q = Q+Q+Q, \qquad 10R = \underbrace{R+\ldots+R}_{10 \text{ maal}} \qquad \text{en} \qquad -4S = -S+-S+-S+-S.$$

In het algemeen zien we een factor maal een punt nP dus als herhaaldelijk optellen van het punt bij zichzelf:

$$nP = \underbrace{P+\ldots+P}_{n \text{ maal}} \qquad n>0 \tag{2.6a}$$

$$nP = \underbrace{-P+\ldots+-P}_{|n| \text{ maal}} \qquad n<0 \tag{2.6b}$$

De associativiteit speelt een belangrijke rol. We willen namelijk (bij cryptografie) wel dat het berekenen van bijvoorbeeld $20P$ op verschillende manieren hetzelfde resultaat levert:

$$4(5P) = 20P = 5(4P).$$

Het achterhalen van n gegeven nP is waar de veiligheid van ECC op leunt. Hoe eenvoudiger dit te kraken is, hoe minder veilig het systeem. Dit zal in § 3.2 aan bod komen.

2.2.4. nP berekenen

Om het punt nP te berekenen, is het inefficiënt om het te doen als volgt

$$\begin{aligned} P &= (x,y) \\ 2P &= P+P \\ 3P &= P+P+P = P+2P \\ 4P &= P+P+P+P = P+3P \\ 5P &= P+P+P+P+P = P+4P \\ &\vdots \end{aligned}$$

Een methode die al een stuk sneller gaat heet het *double-and-add* algoritme. Het beschrijft de stappen om met verdubbelen van het punt P en optellen van P efficiënter bij het punt nP te komen. Het is eenvoudiger te begrijpen met een direct rekenvoorbeeld. Stel we zijn geïnteresseerd in het berekenen van $234P$. We berekenen eerst de binaire representatie van 234

$$\underbrace{234}_{\text{decimaal}} = \underbrace{11101010}_{\text{binair}} \Longrightarrow 234 = 2^7+2^6+2^5+2^3+2^1.$$

Dan geldt er voor $234P$ dat

$$234P = 2^7P + 2^6P + 2^5P + 2^3P + 2^1P.$$

Het idee is om P telkens te verdubbelen en de waardes van deze verdubbelingen op te slaan, om vervolgens de juiste waardes bij elkaar op te tellen. Een verdubbeling van P geeft 2^1P. 2^1P verdubbelt naar 2^2P. Deze verdubbelt naar 2^3P. Bij elke verdubbeling verhoogt de exponent met 1. We moeten voor het getal 234 dus 7 keer verdubbelen, omdat de hoogste macht van 2 in de binaire representatie gelijk is aan 2^7. Nu de verzameling $\{P, 2^1P, 2^2P, 2^3P, 2^4P, 2^5P, 2^6P, 2^7P\}$ is bepaald, kunnen we de benodigde verdubbelingen bij elkaar optellen. Zo zijn voor het getal 234 dus slechts 7 verdubbelingen (double) en 4 optellingen (add) nodig. Dat is al beter dan 233 optellingen!

2.2.5. Torsiepunten

Daar nP interessant is voor de cryptografie, loont het om meer kennis te hebben over n-voud. Aangezien dit veel algemener geldt, hoeven we ons niet te beperken tot slechts $\mathbb{R}$. We kunnen dit bijvoorbeeld zien als een afbeelding $[n]$ waarbij K het lichaam is waar E over gedefinieerd is:

$$\begin{aligned}[m] : E(K) &\to E(K)\\ P &\mapsto mP.\end{aligned}$$

Er valt iets te zeggen over deze afbeelding. Is deze afbeelding namelijk surjectief? In het algemene geval is dat niet zo. Voor een gegeven $Q \in E(K)$ is het niet zozeer noodzakelijk dat deze een n-voud is van een punt P op de kromme. Dat wordt echter verholpen als we kijken naar de volgende afbeelding:

$$\begin{aligned}[n] : E(\overline{K}) &\to E(\overline{K})\\ P &\mapsto nP.\end{aligned}$$

We nemen hier dus punten op E over de *algebraïsche afsluiting* van K. Een algebraïsche afsluiting $\overline{K}$ is een lichaamsuitbreiding van K. Vergelijkingen over K hebben lang niet altijd oplossingen in hun eigen lichaam. Laat $K = \mathbb{R}$: een hele triviale vergelijking over $\mathbb{R}$ zonder oplossingen te hebben in $\mathbb{R}$ is de volgende:

$$x^2 + 1 = 0.$$

De enige oplossingen voor deze vergelijking zijn $x = \pm i$, dat duidelijk niet in $\mathbb{R}$ ligt, maar in $\mathbb{C}$. Aangezien $\mathbb{R} \subset \mathbb{C}$, noemen we $\mathbb{C}$ de algebraïsche afsluiting van $\mathbb{R}$, ofwel $\overline{K} = \mathbb{C}$. Overigens vormt $\mathbb{C}$ haar eigen algebraïsche afsluiting: we noemen $\mathbb{C}$ ook wel algebraïsch afgesloten.

Tevens kunnen we kijken naar punten van eindige orde: torsiepunten. De orde van een punt is in dit geval het kleinste positieve gehele getal n waarvoor geldt dat $nP = \infty$. Met betrekking tot de afbeelding $[n]$ zijn dat precies die punten die naar het ∞ worden gestuurd in $E(\overline{K})$, ofwel $\ker([n])$. De n-torsie kan gedefinieerd worden als volgt:

$$E[n] = \{P \in E(\overline{K}) \mid nP = \infty\} = \ker([n]).$$

De n-torsie bestaat dus uit punten van eindige orde die een deler is van n. Het zijn dus niet alleen punten van exact orde n. Als bijvoorbeeld $4P = \infty$, dan geldt ook dat $8P = 4P + 4P = \infty + \infty = \infty$. Als we in eindige lichamen werken, dan zijn alle punten op de elliptische kromme torsiepunten. Dit komt doordat eindige lichamen – zoals de naam doet vermoeden – bestaat uit een eindig aantal punten.

2.3. Elliptische krommen over eindige lichamen

Voor de toepassingen in de cryptografie kunnen we ons niet permitteren om in $\mathbb{R}$ te blijven werken en zullen we gerichter zijn qua keuzes van lichamen. Dat doet echter niets af van het feit dat $\mathbb{R}$ een

goed idee heeft gegeven over elliptische krommen. Bovendien werken alle formules die opgesteld zijn in Tabel 2.1 door te denken in $\mathbb{R}$ nog steeds voor eindige lichamen. Zo zijn optellingen en vermenigvuldigingen goed gedefinieerd, maar zo ook aftrekken en delen zoals in elk ander lichaam. Onder delen verstaan we vermenigvuldigen met een multiplicatieve inverse.

In de praktijk worden krommen over *eindige* lichamen gebruikt, omdat deze de veiligheid van cryptosystemen gebaseerd op elliptische krommen verhoogt. Het *discrete logaritme probleem* wat ontstaat bij enkele technieken in de cryptografie is namelijk lastiger te kraken in eindige lichamen. Dat zal in § 3.2 aan bod komen. De moeilijkheidsgraad van DLP's is in de volgende lijst toenemend

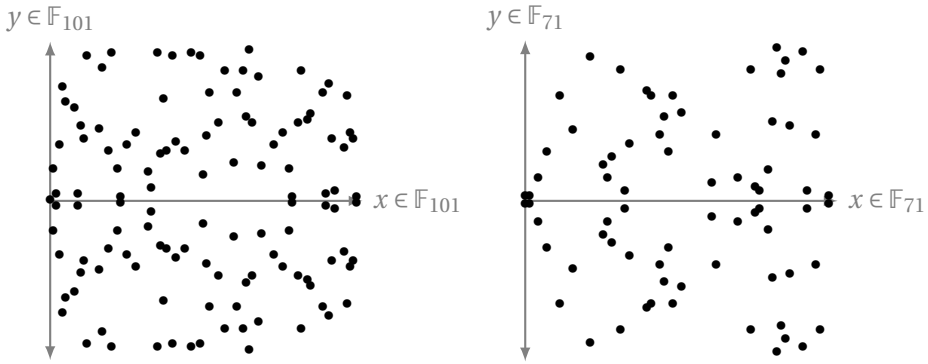
1. DLP in een additieve groep over $\mathbb{F}_q$.
2. DLP in een multiplicatieve groep over $(\mathbb{Z}/q\mathbb{Z})^\times$.
3. DLP in een groep van punten op een elliptische kromme E over $\mathbb{F}_q$, ook wel ECDLP genoemd.

Vandaar dat we nu het algemene geval gaan bespreken voor een elliptische kromme E over een eindig lichaam K. Onze definitie wordt in dat geval

$$E(K) = \{(x, y) \in K^2 \mid y^2 = x^3 + ax + b\} \cup \{\infty\}. \tag{2.7}$$

Vanzelfsprekend zijn nu $a, b \in K$. Merk op dat we in plaats van de korte Weierstrass vorm ook de algemenere vorm kunnen gebruiken; de definities voor $E(K)$ zijn vergelijkbaar. K slaat al dan niet altijd op de verzameling van gehele getallen modulo p met p priem en in dit verslag zal dat zeker zo zijn. Deze wordt normaal gesproken echter genoteerd als $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = \{0, 1, \ldots, p-1\}$. Eindige lichamen worden in het algemeen echter genoteerd als $\mathbb{F}_q$ met $q = p^k$ met p priem en $k \in \mathbb{N}$. Een lichaam van orde q bestaat dan en slechts dan als q een priemmacht is.

Vanwege de keuze van het lichaam zullen we echter geen gladde kromme kunnen plotten met de punten op $E(\mathbb{F}_q)$ zoals in Figuur 2.1. De punten op krommen over eindige lichamen zijn nog steeds te plotten, maar zullen niet veel bijdragen aan het concept. De symmetrie blijft behouden mits we met negatieve y-waarden rekenen. Doordat we rekenen modulo p zouden alle punten op krommen van Figuur 2.4 in principe ook boven de x-as kunnen liggen, wat wel ten koste zou gaan van de symmetrie:



Figuur 2.4: De twee elliptische krommen van Figuur 2.1 over eindige lichamen.

De introductie van modulo rekenen is niet zonder reden. Het is essentieel voor de veiligheid van een systeem gebaseerd op ECC, omdat een cryptosysteem aanzienlijk eenvoudiger is te kraken als er een zekere ordening bestaat in het lichaam. In het lichaam $\mathbb{F}_3 = \{0, 1, 2\}$ geldt bijvoorbeeld het volgende:

$$10 \equiv 1 \pmod 3 \quad \text{en} \quad 102 \equiv 0 \pmod 3.$$

Intuïtief denken we dat $10 < 102$, maar in het lichaam $\mathbb{F}_3$ zijn deze respectievelijk equivalent aan 1 en 0, dus zou volgens deze beredenering ook $1 < 0$, wat duidelijk tegenstrijdig is met eerder genoemde. Onze conclusie is dus dat door het gebruiken van deelgroepen modulo p de begrippen 'groot' en 'klein' lastig te zijn gebruiken. Nu we de eindige lichamen hebben geïntroduceerd kunnen we gebruik maken van eigenschappen van eindige (abelse) groepen. Eén van de eigenschappen die we vaker zullen gebruiken is een gevolg van de stelling van Lagrange: de orde n van een punt $P \in E$ is een deler van de orde van de groep $\#E(\mathbb{F}_p)$. De orde van een element in een groep in dit geval is het kleinste positieve gehele getal n zodanig dat $nP = \infty$.

Eén van de belangrijkste stellingen voor ECC gaat over het aantal punten van $E(\mathbb{F}_q)$. De stelling van Hasse zegt het volgende:

Stelling (Hasse)**.** *Laat E een elliptische kromme zijn over $\mathbb{F}_q$. Dan geldt voor de orde $\#E(\mathbb{F}_q)$ van $E(\mathbb{F}_q)$:*

$$|\#E(\mathbb{F}_q) - (q+1)| \leq 2\sqrt{q}.$$

Hasse geeft een afschatting van het aantal punten en het interval $[q+1-2\sqrt{q}, q+1+2\sqrt{q}]$ wordt dan ook wel Hasse's interval genoemd. Neem bijvoorbeeld $E : y^2 = x^3 + x + 1$ over $\mathbb{F}_{103}$: het interval geeft dat $\#E(\mathbb{F}_{103}) \in [84, 124]$. Merk op dat de grenswaarden met de wortel in het algemeen geen gehele getallen zijn, maar aangezien de orde van een groep gedefinieerd is als een geheel getal, mogen we deze logisch afronden naar beneden en boven. Deze informatie alleen is nog niet genoeg om de orde van de groep te vinden. Daarvoor is nog meer informatie nodig, bijvoorbeeld de orde van een punt in $E(\mathbb{F}_{103})$. Het punt $(0, 1)$ ligt op de kromme, waarvan we de orde in principe kunnen achterhalen met algoritmes zoals Baby-step Giant-step. De orde van dit punt is 87 en we weten dat $\#E(\mathbb{F}_{103})$ een veelvoud hiervan moet zijn, omdat dit een eigenschap is van eindige groepen. Het enige veelvoud van 87 wat in Hasse's interval is bevat is 87 zelf, dus vinden we op deze manier $\#E(\mathbb{F}_{103}) = 87$.

De manier hierboven kan gezien worden als omslachtig, omdat we eerst nog de orde van een punt nodig hebben. Het laat echter enkel een demonstratie zien van hoe Hasse's interval werkt. In de praktijk zullen we programma's zoals Sagemath en Pari gebruiken om de orde van een groep dan wel element te berekenen. Deze gebruiken al dan niet vaak een specifiek algoritme voor het achterhalen van $\#E(\mathbb{F}_q)$. Het idee van *Schoof's Algoritme* is het bepalen van het aantal punten op $E(\mathbb{F}_q)$ door te kijken naar het aantal punten $\#E(\mathbb{F}_q) \pmod{l_i}$, waarbij l_i een klein priemgetal is in de verzameling $L = \{l_1, l_2, \ldots, l_n\}$ zodanig dat

$$N = \prod_{l \in S} l > 4\sqrt{q}.$$

Laat $a_q = q + 1 - \#E(\mathbb{F}_q)$, dan is $|a_q| \leq 2\sqrt{q}$ wegens de stelling van Hasse. Als we $a \pmod{N}$ kunnen bepalen, dan kunnen we wegens de keuze van $N > 4\sqrt{q}$ de waarde van $\#E(\mathbb{F}_q)$ bepalen. Door $a \pmod{l}$ te bepalen voor alle $l \in L$ – dit kan met behulp van de torsiepunten $E[n]$ – kunnen we met de Chinese reststelling concluderen wat $a \pmod{N}$ is.

Chinese resstelling (Congruenties)**.** *Laat $p_i \in \mathbb{Z}$ met $\gcd(p_i, p_j) = 1$ voor alle $i \neq j$ en $i, j \in \{1, \ldots, n\}$. Dan geldt voor een stelsel van n congruenties van de vorm*

$$x \equiv a_i \pmod{p_i}, \quad a_i \in \mathbb{Z}$$

dat er een unieke oplossing voor x modulo $N = \prod_{i=1}^{n} a_i$ bestaat.

2.4. Afbeeldingen: pairings en de Frobenius

2.4.1. Weil pairing

Deze pairing is belangrijk gebleken voor het bewijzen van de stelling van Hasse. Bovendien kan deze gebruikt worden om een *DLP* aan te pakken: de Weil pairing wordt namelijk gebruikt om te

onderzoeken of twee punten $P,Q \in E[n]$ lineair afhankelijk van elkaar zijn, $Q = kP$ voor $k \in \mathbb{N}$, of van een ander willekeurig punt $R \in E[n]$:

Laat E een elliptische kromme zijn over K met $n \in \mathbb{N}$ zodat $p \nmid n$, voor p het karakteristiek van K. Dan kunnen we de de n-torsie identificeren als $E[n] \simeq \mathbb{Z}_n \oplus \mathbb{Z}_n$. Definieer de groep (onder vermenigvuldiging) van n-de eenheidswortels van $\overline{K}$.

$$\mu_n = \{x \in \overline{K} \mid x^n = 1\}.$$

Aangezien $p \nmid n$ zijn er geen samenvallende punten, dus zijn alle nulpunten uniek. Een voortbrenger ζ van de groep noemen we een n-de primitieve eenheidswortel.

Daaruit kunnen we concluderen dat μ_n een cyclische groep is van orde n. Definieer nu de *Weil pairing* als een afbeelding

$$\begin{aligned} e_n : E[n] \times E[n] &\to \mu_n \\ (P,Q) &\mapsto \zeta \end{aligned}$$

De Weil pairing is bilineair in vermenigvuldigingszin

$$\begin{aligned} e_n(P_1 + P_2, Q) &= e_n(P_1, Q)e_n(P_2, Q) \\ e_n(P, Q_1 + Q_2) &= e_n(P, Q_1)e_n(P, Q_2). \end{aligned}$$

Dit was te verwachten wegens de structuur van het beeldverzameling μ_n. In het bijzonder geldt dus dat

$$e_n(kP, Q) = e_n(P, kQ) = e_n(P, Q)^k.$$

Bovendien is deze pairing niet-gedegenereerd in beide variabelen, dus $e_n(\infty, Q) = e_n(P, \infty) = 1$ voor alle $P,Q \in E[n]$. Ook geldt er $e_n(P,P) = 1$. Er zijn daarnaast nog andere eigenschappen waaraan de Weil pairing voldoet, maar deze zullen op de volgende na niet verder worden besproken. We verwijzen naar Washington [31] voor deze niet-besproken eigenschappen.

De eigenschap die wel belangrijk wordt bevonden heeft betrekking tot de Frobenius afbeelding, of eigenlijk voor elk endomorfisme α. Namelijk dat

$$e_n(\alpha(P), \alpha(Q)) = e_n(P, Q)^{\deg(\alpha)}.$$

Als we over eindige lichamen gaan werken, wat we zullen doen, geldt bovenstaande eigenschap in het bijzonder voor het *Frobenius* endomorfisme.

2.4.2. Frobenius

Voor ECC wordt er gebruik gemaakt van de *Frobenius* afbeelding. Hiervoor zullen we enkele definities introduceren om uiteindelijk te bouwen naar de definitie hiervan. *Endomorfismen* werden net genoemd bij de Weil pairing. Dit is een *homomorfisme*, een afbeelding die een zekere structuur behoudt bij het afbeelden, in de context van elliptische kromme gegeven door *rationale* functies. Een endomorfisme heeft de volgende definitie in de context van elliptische krommen:

$$\begin{aligned} &\alpha : E(\overline{K}) \to E(\overline{K}) \\ &\quad (x,y) \mapsto (R_1(x,y), R_2(x,y)), \qquad R_{1,2}(x,y) = \frac{T_{1,2}(x,y)}{N_{1,2}(x,y)}, \quad T(x,y), N(x,y) \quad \text{polynomen} \end{aligned}$$

met de volgende eigenschappen

$$\begin{aligned} \alpha(P+Q) &= \alpha(P) + \alpha(Q) \\ \alpha(\infty) &= \infty \\ \alpha(-P) &= -\alpha(P). \end{aligned}$$

Het is mogelijk om het homomorfisme om te schrijven naar $\alpha(x,y) = (R_1(x), R_2(x)y)$, waar $R_i(x) = m_i(x)/n_i(x)$ met $m_i(x), n_i(x)$ polynomen voor $i = 1,2$. We definiëren de graad

$$\deg(\alpha) = \max\{\deg m(x), \deg n(x)\}.$$

Voor de nulafbeelding $\alpha \equiv 0$ laten we $\deg(\alpha) = 0$. Er is nog een eigenschap waar endomorfismen aan kunnen voldoen en die we nodig hebben als laatste definitie. Een *separabel* endomorfisme is een niet-triviaal endomorfisme ($\alpha \not\equiv 0$) waar in de context van elliptische kromme geldt dat $m'(x) \not\equiv 0 \vee n'(x) \not\equiv 0$. Als minstens één van de $m'(x), n'(x)$ ongelijk is aan 0, dan is het endomorfisme *inseparabel*. Met deze definities vinden we dat

$$\text{separabel} + \text{inseparabel} \text{ `=' separabel}$$
$$\text{inseparabel} + \text{inseparabel} \text{ `=' inseparabel}$$

De som van separabele endomorfismes kan niet zonder extra informatie over de polynomen $m(x)$ en $n(x)$ gedetermineerd worden als separabel dan wel inseparabel.

Laat E een kromme zijn over $\mathbb{F}_q$. Het *Frobenius* endomorfisme is de afbeelding

$$\begin{aligned} \phi_q : \overline{\mathbb{F}}_q &\to \overline{\mathbb{F}}_q \\ x &\mapsto x^q \end{aligned}$$

Voor elliptische krommen wordt vaak gebruik gemaakt van Frobenius op coördinaten (x,y) en zou dus gegeven worden door $\phi_q(x,y) = (x^q, y^q)$. De Frobenius kan dus eigenlijk ook gedefinieerd worden door $\phi_q : E(\overline{\mathbb{F}}_q) \to E(\overline{\mathbb{F}}_q)$. Over $\mathbb{F}_q$ is de Frobenius een inseparabel endomorfisme met $\deg(\phi_q) = q$. Bovendien is de Frobenius wel bijectief, maar de inverse kan niet geschreven worden in polynomen. Het Frobenius endomorfisme heeft ook een *duaal endomorfisme* en die noteren we als $\widehat{\phi_q}$; de *Verschiebung*. Een eigenschap die de Frobenius en de Verschiebung aan elkaar relateert is de volgende identiteit

$$\phi_q \widehat{\phi_q} = \deg(\phi_q) = q. \tag{2.8}$$

Om een volgende belangrijke vergelijking te introduceren, zullen we eerst kijken naar het endomorfisme $1 - \phi_q$. Hierbij is 1 het endomorfisme dat een punt naar zichzelf afbeeldt. De som van endomorfismen is weer een endomorfisme, dus aangezien 1 een separabel endomorfisme is en ϕ_q inseparabel, geldt dat $1 - \phi_q$ separabel is. Een eigenschap die te vinden is in [31] zegt dat

$$\#E(\mathbb{F}_q) = \deg(1 - \phi_q) = (1 - \phi_q)\widehat{(1 - \phi_q)},$$

omdat $1 - \phi_q$ separabel is. Dit zullen we verder niet bewijzen. Dit kunnen we echter wel verder uitwerken, waardoor

$$\#E(\mathbb{F}_q) = (1 - \phi_q)\widehat{(1 - \phi_q)} = (1 - \phi_q)(\widehat{1} - \widehat{\phi_q}) = 1 - \phi_q - \widehat{\phi_q} + \phi_q\widehat{\phi_q} = q + 1 - (\phi_q + \widehat{\phi_q})$$

Maar dat komt ons bekend voor! Er gold eerder namelijk al dat $\#E(\mathbb{F}_q) = q + 1 - a_q$. Er geldt dus dat $a_q = \phi_q + \widehat{\phi_q}$. Hasse's grens kunnen we bewijzen door het endomorfisme $m + n\phi_q$ te bekijken met $m, n \in \mathbb{Z}$.

$$0 \le \deg(m - n\phi_q) = (m - n\phi_q)(m - n\widehat{\phi_q}) = m^2 - mn(\phi_q + \widehat{\phi_q}) + n^2\phi_q.$$

Dus $m^2 - mn(\phi_q + \widehat{\phi_q}) + n^2\phi_q \ge 0$, dat wil zeggen, de discriminant $D = n^2(\phi_q + \widehat{\phi_q})^2 - 4n^2q \le 0$. Aangezien $n^2 \neq 0$, concluderen we dat

$$(\phi_q + \widehat{\phi_q})^2 - 4q \le 0 \iff |\phi_q + \widehat{\phi_q}| \le 2\sqrt{q} \iff |a_q| \le 2\sqrt{q}$$

We vinden dus dat $m^2 - a_q mn + n^2 q = \deg(m - n\phi_q)$. Maar als we $m = \phi_q, n = 1$ kiezen als endomorfismen vinden we

$$\phi_q^2 - a_q\phi_q + q = \deg(\phi_q - \phi_q) = \deg(0) = 0$$

Dus blijkbaar hebben we een afbeelding gevonden dat het nulendomorfisme is:

$$\phi_q^2 - a_q\phi_q + q = 0 \tag{2.9}$$

Aangezien dit endomorfisme werkt op een punt $P \in E(\mathbb{F}_q)$, geldt er dat

$$\phi_q^2(P) - a_q\phi_q(P) + qP = \infty.$$

Vergelijking (2.9) is net als de stelling van Hasse belangrijk voor het aantal punten van $E(\mathbb{F}_q)$. Het linkerlid van de vorm $X^2 - aX + q$ wordt ook wel het *karakteristieke polynoom van de Frobenius* genoemd. Dit kan ook uitgewerkt worden met behulp van Cayley-Hamilton. De meer technische details zullen niet aan bod komen, aangezien deze niet veel inzicht geeft voor het gebruik van deze identiteit.

Aangezien ϕ_q een endomorfisme is dat werkt op $E[n] \simeq (\mathbb{Z}/n\mathbb{Z}) \times (\mathbb{Z}/n\mathbb{Z})$ en er een zekere basis bestaat voor $(\mathbb{Z}/n\mathbb{Z})^2$, kunnen we deze identificeren aan een 2×2 matrix A ten opzichte van een zekere basis. Schrijf

$$A = \begin{pmatrix} \alpha_1 & \beta_1 \\ \alpha_2 & \beta_2 \end{pmatrix}.$$

Cayley-Hamilton zegt dat voor het karakteristiek polynoom $p(\lambda) = \det(\lambda I_2 - A)$ geldt dat $p(A) = 0$. Als we dit uitwerken krijgen we:

$$\begin{aligned} \det\begin{pmatrix} \lambda - a & b \\ c & \lambda - d \end{pmatrix} &= (\lambda - a)(\lambda - d) - bc \\ &= \lambda^2 - (a+d)\lambda + ad - bc \\ &= \lambda^2 - \mathrm{tr}(A)\lambda + \det(A) \implies A^2 - \mathrm{tr}(A)A + \det(A) = 0 \end{aligned}$$

A identificeren aan haar endomorfisme geeft $\phi_q^2 - \mathrm{tr}(\phi_q)\phi_q + \det(\phi_q) = 0$. Dit zou dan moeten betekenen dat

$$a_q = \mathrm{tr}(\phi_q) \quad \text{en} \quad q = \det(\phi_q),$$

wat inderdaad ook zo is. De details zijn te vinden in Washington.

Eén toepassing van deze vergelijking die wij ook daadwerkelijk zullen gebruiken is bij het construeren van een elliptische kromme via *complex multiplication* (CM).

2.4.3. Tate pairing

De *Tate pairing* werd oorspronkelijk geïntroduceerd om enkele processen te versnellen. Het is gebaseerd op de Weil pairing en toont qua definities en eigenschappen daardoor ook gelijkenissen.

Laat E een elliptische kromme zijn over $\mathbb{F}_q$. Voor $n \in \mathbb{N}$ en $n|q-1$ noteren we de n-torsie als $E[n]$ (over $\mathbb{F}_q$ en de groep van eenheidswortels μ_n de groep van n-de eenheidwortels in $\mathbb{F}_q$. Laat $P \in E[n]$, $Q \in E$ en kies $R \in E(\overline{\mathbb{F}}_q)$ zodanig dat $nR = Q$. Definieer de Tate pairing τ_n als afbeelding

$$\begin{aligned} \tau_n : E(\mathbb{F}_q)[n] \times E(\mathbb{F}_q)/nE(\mathbb{F}_q) &\to \mu_n \\ (P,Q) &\mapsto e_n(P, R - \phi_q(R)), \end{aligned}$$

die niet-gedegenereerd en bilineair is. De laatste eigenschap verschilt dit keer per coördinaat qua betekenis. Er geldt dat $\tau_n(P,Q) = 1$ voor alle $Q \in E/nE$ betekent dat $P = \infty$. Voor $\tau_n(P,Q) = 1$ voor alle $P \in E[n]$ geldt niet dat $Q = \infty$ vanwege de structuur van E/nE, maar dat $Q \in nE$. Voor de bilineairiteit geldt in principe hetzelfde als bij de Weil pairing, maar willen we benadrukken dat

$$\tau_n(kP,Q) = \tau_n(P,kQ) = \tau_n(P,Q)^k,$$

waardoor deze dezelfde eigenschap als de Weil pairing heeft. Een wezenlijk verschil tussen de Weil en Tate pairing is echter dat voor een Tate pairing niet hoeft te gelden dat $\tau_n(P,P) = 1$, maar dat het wel mogelijk is.

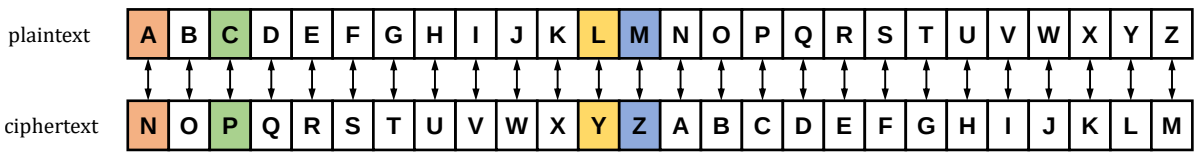
3

Cryptografie

3.1. Enkele zwakke cryptosystemen

Net zoals voor elliptische krommen, zullen we eerst een concept over cryptografie proberen te ontwikkelen alvorens deze te combineren met onze kennis over elliptische krommen. Cryptografie heeft alles te maken met informatie versleutelen. Een algemeen scenario is als volgt: twee partijen willen vertrouwelijke informatie uitwisselen, maar er bestaat een risico dat elk bericht tussen hen wordt onderschept. Het idee is dan om een bericht, de *plaintext*, zodanig te 'vervormen' dat het onleesbaar is voor alle partijen wanneer deze wordt uitgewisseld als de *ciphertext*. Het transformeren van plaintext naar ciphertext noemen we *encryptie* en gaat gepaard met coderen. Hierbij maken we onderscheid tussen *symmetrische* encryptie en *asymmetrische* encryptie. Om het bericht leesbaar te maken is er een code, *key*, nodig om deze terug te transformeren en deze wordt vaak gekozen als de inverse-transformatie van de code bij encryptie. Het is dan ook natuurlijk dat dit proces *decryptie* heet, ofwel decoderen/ontcijferen. De veiligheid van zo'n systeem wordt al dan niet bepaald door het bezitten van deze code. We gaan er namelijk vanuit dat elk informatie-uitwisseling onderschept kan worden.

Het is goed mogelijk om voor te stellen waar deze waarborging van informatie belangrijk voor is. Oorspronkelijk werden toepassingen van de cryptografie enkel gebruikt voor de geheimhouding en werd vaak toegepast voor militaire doeleinden: denk aan locaties van vijanden, gegevens van kampen en wellicht omschrijvingen van personen. Zo begon Julius Caesar lang geleden al met een cryptosysteem, wat later vernoemd werd naar het *caesarcijfer*. Deze behoort tot een veel groter cryptosysteem, namelijk de substitutiecijfer. Daar zullen we echter niet verder op ingaan. Het caesarcijfer gaat via symmetrische encryptie en kan schematisch worden weergegeven zoals in Figuur 3.1



Figuur 3.1: Voorbeeld van caesarcijfer

Een bericht dat verstuurd wordt, wordt dus letter per letter gewijzigd bijvoorbeeld zoals in Figuur 3.1. Een L uit plaintext wordt volgens de code hierboven een Y in ciphertext, een $M \mapsto Z$ enzovoorts. In deze symmetrische encryptie wordt het versleutelde bericht met dezelfde code, specifieker zijn inverse, weer hersteld. Zo kan het volgende versleutelde bericht ontcijferd worden:

> *Pnrfnepvwsre vf npugreunnyq! Urg vf rraibhqvt gr xenxra, bzqng re zrg mbtrurgra serdhragvrnanylfr rraibhqvt xna jbeqra antrtnna jryxr yrggref zrg ryxnne pbeerfcbaqrera. Obiraqvra xna qvg jbeqra trxennxg zrg oehgr-sbepr, bsjry ryxr zbtryvwxurvq gr ceborera. Pelcgbtensvr zrg ryyvcgvfpur xebzzra vf uryrznny va!*

Voor ciphertext naar plaintext geldt dan $P \mapsto C$, $n \mapsto a$... dit voortzetten geeft het volgende:

> *Caesarcijfer is achterhaald! Het is eenvoudig te kraken, omdat er met zogeheten frequentieanalyse eenvoudig kan worden nagegaan welke letters met elkaar corresponderen. Bovendien kan dit worden gekraakt met brute-force, ofwel elke mogelijkheid te proberen. Cryptografie met elliptische krommen is helemaal in!*

Vanzelfsprekend is dit niet het enige voorbeeld, maar wel een klassieke. Een ander (zwak) voorbeeld dat kinderen gezien kunnen hebben is bekend van de Donald Duck! Het is de spiegeling van tekst langs de verticale as, zoals volgt:

> De Donald Duck stopt dit soort berichten soms voorraan in hun wekelijkse blad. De uitdaging is om het bericht te lezen zonder dat je er een spiegel bij pakt. Leuk voor kinderen, maar er zijn hele duidelijke manklemmenten aan dit systeem. Ik ben ervan overtuigd dat de lezer dit bericht kan ontcijferen en de zwakheden van dit bericht kan bedenken! Cursief is deze wel verveelender te lezen vanwege het lettertype, dat vermijd ik dan ook.

Het woord is al eerder gevallen, maar we noemen deze systemen *cryptosystemen*. Het systeem bestaat namelijk uit het volgende:

- Een verzameling $\mathscr{P}$ van alle mogelijke plaintext karakters
- Een verzameling $\mathscr{C}$ van alle mogelijke ciphertext karakters
- Een verzameling $\mathscr{K}$ van alle mogelijke sleutels
- Een injectieve functie $f_k : \mathscr{P} \to \mathscr{C}$, ook wel de coderende functie afhankelijk van een sleutel k
- Een functie $f_k^{-1} : \mathscr{C} \to \mathscr{P}$, ook wel de decoderende functie

Hierbij eisen we dat f_k en f_k^{-1} elkaars inverse zijn, dat wil zeggen, voor alle $P \in \mathscr{P}$ en voor alle $C \in \mathscr{C}$ geldt dat

$$f_k(f_k^{-1}(C)) = C \quad \text{en} \quad f_k^{-1}(f_k(P)) = P.$$

Dit betekent dat elk gedecodeerde bericht $f_k^{-1}(C)$ dat we willen versleutelen weer precies het oorspronkelijke gedecodeerde bericht is, respectievelijk elk gecodeerde bericht $f_k(P)$ dat we willen decoderen weer het bericht P. Voor het caesarcijfer kunnen we bijvoorbeeld elk letter uit het alfabet schrijven als een getal uit de verzameling $\{0,\ldots,25\}$, waarbij 0 een A voorstelt, 4 een E enzovoorts. De functie f_k wordt bij eerdergenoemde caesarcijfer gegeven door $f_{13}(P) = P + 13 \pmod{26}$ met $P \in \{0,\ldots,25\}$. Vanwege de cyclische shift is het logisch als $\{0,\ldots,25\}$ de verzameling getallen zijn modulo 26. Zo wordt $27 \equiv 1 \pmod{26}$ en zou 27 dus gelijk zijn aan B. De inverse functie f_{13}^{-1} wordt gegeven door $f_{13}^{-1}(C) = C - 13 \pmod{26}$ met $C \in \{0,\ldots,25\}$. Merk op dat modulo 26 geldt dat $-13 \equiv 13$, dus ook $f_{13}^{-1}(C) = C + 13$. Dat betekent dus dat zowel de encryptie functie als decryptie functie gelijk zijn. Het caesarcijfer van net was dus een speciaal zwak cryptosysteem! Het wordt ook wel Rot(13) genoemd en is het caesarcijfer waarbij tweemaal toepassen van de encryptie dan wel decryptiefunctie het originele bericht geeft, ofwel, f_{13} en f_{13}^{-1} zijn hun eigen inverses:

$$f_{13}(f_{13}(P)) = f_{13}(P+13) = P+26 \equiv P \pmod{26}, \quad f_{13}^{-1}(f_{13}^{-1}(C)) = f_{13}^{-1}(C+13) = C+26 \equiv C \pmod{26}.$$

Laten we nu kijken naar systemen die hedendaags nog worden gebruikt en de methodiek hiervan.

3.2. DH, RSA en DSA

Zoals net te zien was, waren de cryptosystemen erg eenvoudig te kraken als de truc bekend was. Zelfs als we niet wisten wat de shift van het caesarcijfer was, hoefde een onderschepper van het bericht maar 26 – of 25 afhankelijk van hoe je het bekijkt – verschillende opties te proberen om het bericht te lezen. Dat is niets vergeleken met 'echte' veiligheid. Vanaf nu zullen termen zoals '(relatief) snel', 'efficient' en dergelijke gebruikt worden om enkele algoritmes en berekeningen te typeren. In principe noemen we deze zo, als deze in *polynomiale tijd* kunnen worden berekend. Een algoritme is van polynomiale tijd indien de totale tijd van het proces $T(n)$ uitgedrukt in de grootte van de input n geschreven kan worden als $T(n) = \mathcal{O}(n^k)$ voor k positief geheel. Een veilig cryptosysteem is dan eigenlijk ook alleen veilig, omdat men nog geen algoritmes heeft gevonden van polynomiale tijd of minder. Een gedetailleerder beschrijving van tijdscomplexiteit kan gevonden worden in het werk van Arora en Barak [3].

3.2.1. Diffie-Hellman

Een systeem dat beter werkt is wanneer er een zogenaamde *geheime sleutel* is waar twee partijen het over eens zijn, die niet eenvoudig gevonden kan worden door een onderschepper. Een manier om zo'n geheime sleutel samen te stellen is de *Diffie-Hellman key exchange* (DH). DH is een methode waarbij twee partijen publiekelijk een geheime sleutel, als in bruikbaar voor enkel deze partijen, kunnen opstellen. We nemen hierbij aan dat elk bericht tussen de twee partijen onderschept kan worden door een derde partij. In de cryptografie worden de twee partijen die de sleutel willen opstellen vaak aangeduid als Alice en Bob, waarbij de onderschepper vaak genoteerd wordt als Eve. DH is vernoemd naar Whitfield Diffie en Martin Hellman wegens hun publicatie in 1976 [10]. Laten we het principe achter het klassieke DH systeem bekijken.

Alice en Bob willen elkaar een bericht sturen en hebben daarbij beide individueel een zogeheten *geheime sleutel* in gedachte. De naamgeving duidt op het feit dat ze deze sleutel zelfs niet naar elkaar wordt verstuurd, waardoor de onderschepper Eve dus ook geen kennis heeft over deze sleutels. Deze sleutels worden gekozen in het lichaam $\mathbb{Z}/p\mathbb{Z} = \mathbb{F}_p$ waar ze beide over eens zijn met p priem. Bovendien kiezen ze een getal $G \in \mathbb{F}_p$, maar zowel G als het lichaam $\mathbb{F}_p$ zijn bekend voor Eve, omdat ze publiekelijk worden gedeeld. Noem de geheime sleutel van Alice α en die van Bob β met $\alpha, \beta \in \mathbb{F}_p$. Een ieder zullen ze nu het getal G verheffen met hun eigen geheime sleutel en rekenen ze modulo p, waardoor de volgende getallen ontstaan:

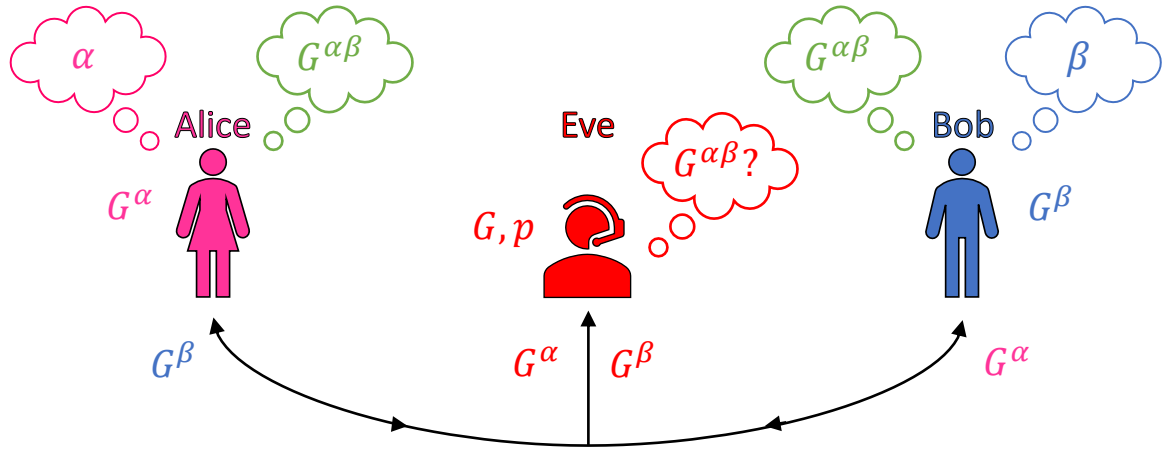
$$A = G^{\alpha} \pmod{p} \quad \text{en} \quad B = G^{\beta} \pmod{p}.$$

Vervolgens sturen ze deze A en B naar elkander, waardoor ook Eve weer deze A en B in handen krijgt. Nu komt het moment waar Eve buitenspel gezet wordt. Alice en Bob berekenen nu het getal dat gemeenschappelijk hoort te zijn: ze verheffen elkaars verstuurde getallen met hun geheime sleutel. Zo ontstaat het getal K

$$B^{\alpha} \pmod{p} \equiv (G^{\beta})^{\alpha} \pmod{p} = K = (G^{\alpha})^{\beta} \pmod{p} \equiv A^{\beta} \pmod{p}.$$

Alice en Bob bezitten nu dus beiden het gemeenschappelijke getal K, maar Eve heeft enkel de getallen p, G, A en B. In Figuur 3.2 staat de situatie schematisch weergegeven. Als Eve dus op de een of andere manier achter het getal α of β komt, heeft ze in principe ook het getal K. Het verschil tussen α, β en K is dat α, β niet voor Alice en Bob beiden bekend zijn, maar dat K dat wel is. De vraag is dan ook: kan Eve de waarde van K achterhalen?

De eenvoud van het antwoord op de gestelde vraag is waar de veiligheid van een systeem gebaseerd op DH op leunt: hoe eenvoudiger K te achterhalen, hoe minder veilig het systeem. Het probleem waar een onderschepper, Eve, in dit geval op stuit is het bepalen van (één van de) geheime sleutels α en β aan de hand van de gegeven informatie: p, G, A en B. Het eerste wat intuïtief



Figuur 3.2: Diffie-Hellman key-exchange

een aanpak zou kunnen zijn is om de logaritme te nemen van A dan wel B, om zo vervolgens α dan wel β te bepalen:

$$\log(A) \stackrel{?}{=} \alpha \log(G) \pmod{p} \quad \text{en} \quad \log(B) \stackrel{?}{=} \beta \log(G) \pmod{p}.$$

Aan de notatie te zien gaat dat echter niet zo makkelijk. Aangezien we rekenen modulo p, kunnen we niet zomaar de continue logaritme nemen zoals dat gaat in $\mathbb{R}$. In eindige lichamen $\mathbb{F}_q$ hebben we namelijk te maken met *discrete logaritmen*. Het probleem dat er ontstaat heet dan ook wel het *discrete logaritme probleem* (DLP).

Discrete Logaritme Probleem (Klassieke DH)**.** *Gegeven $G, G^n \in \mathbb{F}_q$ en $n \in \mathbb{F}_q$, bereken n.*

De moeilijkheid van het kraken van DLP's hangt voornamelijk af van de keuzes voor het lichaam $\mathbb{F}_p$ waarover gewerkt wordt. In een relatief klein eindig lichaam $\mathbb{F}_p$ is het geen moeilijke taak om een computer elke mogelijke oplossing te laten controleren, om zo de geheime sleutels en dus ook K te bepalen.

Voorbeeld van brute-force op klassieke DH

Neem als voorbeeld $p = 101, G = 10$ en laat α willekeurig in $\mathbb{F}_{101}$. Herinner dat het gemeenschappelijke getal K niet bekend is bij Eve, dus kunnen we hier ook niet naar toe werken. Een mogelijke implementatie van brute-force is het volgende:

```
sage: starttime = time.time()
....: p = 101; G = 10; F = IntegerModRing(p)
....: alpha = F.random_element(); A = G^alpha % p
....: for i in srange(1,p):
....:     if G^i % p == A:
....:         print 'Volgens de methode is de geheime sleutel alpha =', str(i)+'.'
....:         print 'In werkelijkheid is alpha =', str(alpha)+'.'
....:         print 'Het proces duurde', time.time()-starttime, 'seconden.'
....:         break
....:     else:
....:         continue
....:
```

```
Volgens de methode is de geheime sleutel alpha = 3.
In werkelijkheid is alpha = 87.
Het proces duurde 0.000443935394287 seconden.
```

Er ontstaat een probleem. Eve heeft niet precies de geheime sleutel van Alice weten te kraken, maar heeft wel een getal $\tilde{\alpha}$ gevonden waarvoor $A \equiv G^{\tilde{\alpha}} \pmod{p}$. Wegens commutativiteit in de exponent kan Eve dus zo de gemeenschappelijke sleutel vinden:

$$K = (G^{\alpha})^{\beta} \equiv (G^{\tilde{\alpha}})^{\beta} \equiv (G^{\beta})^{\tilde{\alpha}} \pmod{101}.$$

In principe is het bestaan van een *pseudo-α* geen probleem voor Eve, omdat ze uiteindelijk bij de gemeenschappelijke geheime sleutel is gekomen: dat is immers waar het om gaat bij het kraken van DH. Voor Alice en Bob is dat echter wel! Er lijken dus twee waarden – wellicht wel meer – te zijn waarvoor Eve op K had uit kunnen komen. Het is eenvoudig te bedenken dat hoe meer waarden mogelijk zijn om K te achterhalen, hoe minder veilig het systeem is dat Alice en Bob hanteren. Had dat voorkomen kunnen worden zodat er géén pseudo-α's bestaan? Het antwoord daarop vinden we in onze keuze voor G in bovenstaand voorbeeld.

Het voorval ontstaat vanwege het 'onzorgvuldig' kiezen van G: in dit voorbeeld is G geen *primitieve wortel modulo* 101. Een getal g is een primitieve wortel modulo n als g de multiplicatieve groep

$$(\mathbb{Z}/n\mathbb{Z})^{\times} = \{a \equiv g^x \in \mathbb{Z}/n\mathbb{Z} \mid \gcd(a,n) = 1, x \in \mathbb{Z}\}$$

voortbrengt. Met andere woorden, voor elk getal a dat copriem is met n bestaat er een macht g^x met $x \in \mathbb{Z}$ waarvoor $g^x \equiv a \pmod{n}$. Bovendien geldt er dat er voor elke a slechts één $x \in \mathbb{Z}$ bestaat op een cyclische shift van $n-1$ na, waardoor eerdergenoemde pseudo-α's niet voorkomen. Indien $n = p$ met p priem is elk getal $a \in \{1,\ldots,p-1\}$ copriem met p, dus geldt ook $(\mathbb{Z}/p\mathbb{Z})^{\times} = \{1,\ldots,p-1\}$. Dit stemt overeen met de formule van de cardinaliteit van $(\mathbb{Z}/p\mathbb{Z})^{\times}$, namelijk $|(\mathbb{Z}/p\mathbb{Z})^{\times}| = p-1$ met p priem.

Het kiezen van primitieve wortels zou volledige veiligheid echter niet garanderen – dat is sowieso al lastig – maar het zou aanzienlijk minder eenvoudig zijn om een correcte sleutel te vinden. Als we de code aanpassen, dan vinden we – zelfs na vaak herhalen – dat de gevonden sleutel altijd overeenkomt met de oorspronkelijke sleutel:

```
sage: starttime = time.time()
....: p = 101; G = primitive_root(p); F = IntegerModRing(p)
....: alpha = F.random_element(); A = G^alpha % p
....: for i in srange(1,p):
....:     if G^i % p == A:
....:         print 'Volgens de methode is de geheime sleutel alpha =', str(i)+'.'
....:         print 'In werkelijkheid is alpha =', str(alpha)+'.'
....:         print 'Het proces duurde', time.time()-starttime, 'seconden.'
....:         break
....:     else:
....:         continue
....:
Volgens de methode is de geheime sleutel alpha = 11.
In werkelijkheid is alpha = 11.
Het proces duurde 0.000319004058838 seconden.
```

Laten we kijken naar een voorbeeld met grotere waarde voor p. We verhogen het priemgetal aanzienlijk en kijken naar p in de orde van 2^{24}. De reden hiervoor is dat een p rond de 2^{16} binnen enkele seconden te berekenen is met brute-force, maar 2^{32} een MemoryError geeft in SageMath:

```
sage: starttime=time.time()
....: p = next_prime(2^24); G = primitive_root(p); F = IntegerModRing(p)
....: alpha = F.random_element(); A = G^alpha % p
....: for i in srange(1,p):
....:     if G^i % p == A:
....:         print 'Volgens de methode is de geheime sleutel alpha =', str(i)+'.'
....:         print 'In werkelijkheid is alpha =', str(alpha)+'.'
....:         print 'Het proces duurde', time.time()-starttime, 'seconden.'
....:         break
....:     else:
....:         continue
....:
Volgens de methode is de geheime sleutel alpha = 10177459.
In werkelijkheid is alpha = 10177459.
Het proces duurde 24345.4291892 seconden.
```

Omgerekend duurde dit proces dus > 6.5 uur. Om aan te tonen dat dit niet efficiënt is, gebruiken we een implementatie voor het berekenen van discrete logaritmen in SageMath.

```
sage: starttime = time.time()
....: alf = discrete_log(A, G, p-1, '*')
....: print 'Volgens een ingebouwde methode in SageMath is alpha = ', alf
....: print 'Het proces duurde', time.time()-starttime, 'seconden.'
....:
Volgens een ingebouwde methode in SageMath is alpha =  10177459
Het proces duurde 0.00659203529358 seconden.
```

Dit gaat inderdaad een stuk sneller dan brute-force. Afhankelijk van de waarde van α kunnen zelfs DLP's met grote waarden voor p snel worden opgelost met de brute-force implementatie, dus had deze implementatie net zo goed de sleutel kunnen vinden in veel kortere tijd. Het kan namelijk zijn dat α relatief klein gekozen wordt, waardoor α snel gevonden kan worden. Echter is de grootste waarde voor p ook niet aan te raden. Als we namelijk p zelf nemen, dan geldt er

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

Dit is de *Kleine stelling van Fermat*. Aangezien we links en rechts kunnen vermenigvuldigen met de inverse van G, volgt er dat $G^{p-1} \equiv 1 \pmod{p}$. Oftewel: als $\alpha = p-1$ gekozen wordt, dan ontvangt Eve een $A = 1$ en weet ze wegens de Kleine stelling van Fermat dat $\alpha = p-1$ wel moet gelden. Het is dus verstandig om $\alpha, \beta \in [2, p-2]$ te kiezen. Daarnaast zou het gemeenschappelijke getal K voor $\alpha = p-1$ ook gelijk zijn aan 1, immers

$$K = A^\beta \equiv 1^\beta \pmod{p}$$

voor elke waarde van β. Al met al concluderen we dat we met deze implementatie van brute-force teveel afhangen van 'geluk': hoe kleiner α gekozen wordt, hoe sneller deze gevonden kan worden. Het kiezen van grotere waarden voor $n = p$ priem zorgt er echter voor dat de tijdstappen langer duren:

```
sage: starttime=time.time()
....: p = next_prime(2^24); G = primitive_root(p); F = IntegerModRing(p)
....: alpha = 10; A = G^alpha % p
....: for i in srange(1,p):
```

```
....:     if G^i % p == A:
....:         print 'Volgens de methode is de geheime sleutel alpha =', str(i)+'.'
....:         print 'In werkelijkheid is alpha =', str(alpha)+'.'
....:         print 'Het proces duurde', time.time()-starttime, 'seconden.'
....:         break
....:     else:
....:         continue
....:
Volgens de methode is de geheime sleutel alpha = 10.
In werkelijkheid is alpha = 10.
Het proces duurde 7.07562494278 seconden.
```

Samenvattend sommen we de stappen op voor de totstandkoming van de geheime sleutel met DH:

1. Alice en Bob laten publiekelijk weten over welk lichaam $\mathbb{F}_p$ ze werken en kiezen G als een primitieve wortel modulo p.
2. Alice en Bob kiezen respectievelijk geheime getallen α en β.
3. Alice berekent $A = G^\alpha \pmod p$, Bob berekent $B = G^\beta \pmod p$.
4. Ze delen deze A en B publiekelijk met elkaar.
5. Alice berekent $B^\alpha \pmod p$ en Bob berekent $A^\beta \pmod p$.
6. Ze komen gezamenlijk uit op $K = G^{\alpha\beta} \pmod p$.

Het meest efficiënte algoritme voor DLP's heeft een looptijd van

$$\mathcal{O}\left(\exp\left((C+o(1))\sqrt{\log(k)}\sqrt{\log[\log(k)]}\right)\right)$$

met $C > 0$ constant, $k = p$ en heet de *Index Calculus*, zie § 3.4.5 .

3.2.2. Rivest-Shamir-Adleman

Waar DH zojuist meer gezien wordt als een protocol om een sleutel publiekelijk uit te wisselen, is *Rivest-Shamir-Adleman* (RSA) een publieke-sleutel cryptosysteem zoals eerder gedefinieerd. Het is vernoemd naar Ron Rivest, Adi Shamir en Leonard Adleman die het principe van dit systeem publiceerde in 1978 [26]. Net als DH maakt RSA ook gebruik van geheime sleutels en publieke sleutels. De publieke sleutel van DH van zojuist kan gezien worden als het paar van parameters (n, g). Laten we kijken naar het idee achter RSA.

Alice en Bob willen nog altijd met elkaar kunnen communiceren in het bijzijn van onderscheppers zoals Eve. Alice begint met het kiezen van twee verschillende priemgetallen p, q. Deze gebruikt ze om het getal $n = pq$ te berekenen. Vervolgens kiest ze de publieke sleutel e – als in encryptie – zodanig dat $1 < e < \varphi(n)$ geheel en $\gcd(e, \varphi(n)) = 1$. Hierbij is $\varphi(n)$ weer Euler's phi functie. Aangezien p, q priem zijn en wegens de werking van φ, geldt er dat $\varphi(n) = (p-1)(q-1)$. De publieke sleutel die Alice nu heeft gecreëerd wordt gegeven door (n, e). De geheime sleutel d – als in decryptie – is gedefinieerd als het getal zodanig dat

$$de \equiv 1 \pmod{\varphi(n)}.$$

d is dus de multiplicatieve inverse van e modulo $\varphi(n)$. Deze houdt Alice geheim, waardoor zowel Bob als Eve deze niet kennen. Het idee is nu dat Alice de publieke sleutel (n, e) naar Bob stuurt en dat Bob nu een bericht naar Alice kan sturen. Hij zet zijn plaintext bericht b om naar een getal b –

waar zometeen een voorbeeld van wordt gegeven – zodat $b < n$ en berekent met de encryptiefunctie f_n

$$f_n(b) \equiv b^e \pmod{n}.$$

$f_n(b)$ vormt de ciphertext en dus stuurt hij deze naar Alice, die Eve dus ook binnenkrijgt. Het berekenen van b^e is waar informatie wordt versleuteld, wat ook af te leiden is uit het machtsverheffen met het encryptiegetal e. Vervolgens is het dus aan Alice de taak om b te achterhalen en doet dat met haar geheime sleutel tevens decryptiecijfer d:

$$f_n^{-1}(f_n(b)) \equiv (b^e)^d \equiv b^{de} \equiv b \pmod{n}$$

Het is echter niet meteen te zien dat de laatste equivalentie geldt. Deze kunnen we bewijzen met de Kleine stelling van Fermat die eerder is genoemd en het feit dat de volgende rekenregel geldt voor modulo rekenen voor p, q priem en x, a geheel:

$$x \equiv a \pmod{p} \quad \text{en} \quad x \equiv a \pmod{q} \implies x \equiv a \pmod{pq}$$

Om aan te tonen dat $b^{de} \equiv b \pmod{n}$, kunnen we met bovenstaande implicatie dus nagaan dat $b^{de} \equiv b \pmod{p}$ en $b^{de} \equiv b \pmod{q}$. Het bewijs dat we geven zal gaan voor modulo p, maar is zonder verlies van algemeenheid (WLOG). Dat betekent dat het bewijs ook op zal gaan voor het geval modulo q. Hiervoor zullen we twee gevallen onderscheiden: $b \equiv 0 \pmod{p}$ en $b \not\equiv 0 \pmod{p}$

Geval 1: Als $b \equiv 0 \pmod{p}$, lijkt het via intuïtie triviaal te zijn dat dan ook $b \equiv b^{de} \pmod{p}$, aangezien $de \neq 0$. Formeel is het echter te bewijzen als volgt: we kunnen b schrijven als $b = kp$ met k geheel. Er volgt

$$b^{de} = (kp)^{de} = k^{de}p^{de} = k^{de}p^{de-1}p$$

Hieruit concluderen we dat zowel b als b^{de} veelvouden zijn van p, dus zijn ze identiek 0 modulo p.

Geval 2: Als $b \not\equiv 0 \pmod{p}$, dan kunnen we de Kleine stelling van Fermat gebruiken: $b^{p-1} \equiv 1 \pmod{p}$. Deze gebruiken we bij het omschrijven van b^{de}. Wegens de definitie van d, geldt er dat $de \equiv 1 \pmod{(p-1)(q-1)}$. Oftewel, het verschil $de-1$ is een veelvoud van $(p-1)(q-1)$; $de-1 = k(p-1)(q-1)$ met k geheel. Schrijf nu b^{de} om:

$$b^{de} = b^{de-1}b = b^{k(p-1)(q-1)}b = \left(b^{p-1}\right)^{k(q-1)}b$$

Rekenen modulo p zorgt voor de identiteit die we wilden:

$$\begin{aligned} b^{de} &\equiv \left(b^{p-1}\right)^{k(q-1)} b \pmod{p} \\ &\equiv 1^{k(q-1)} b \pmod{p} \\ &\equiv b \pmod{p} \end{aligned}$$

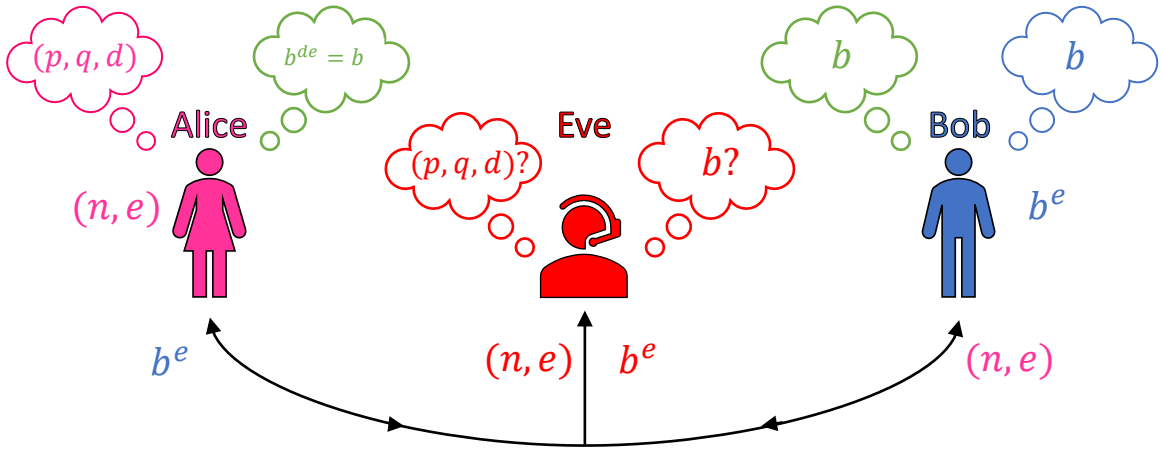
Zoals we wilden aantonen. Het blijkt te kloppen dat het decryptiecijfer inderdaad de plaintext geeft. Deze hoeft Alice dus nog enkel om te zetten. Hieronder staat de eindsituatie van het cryposysteem RSA schematisch weergegeven.

Als Eve dus op de een of andere manier achter het getal d komt, kan ze in principe ook de ciphertext kraken. De vraag is dus in dit geval: kan Eve de waarde d achterhalen?

Net zoals bij DH, is het niet-kennen van de geheime sleutel de crux van de veiligheid in het systeem: hoe eenvoudiger d te achterhalen is, hoe minder veilig het systeem. Het probleem voor Eve is dat ze gegeven n en e het decryptiecijfer moet achterhalen. Als we de definitie van d voor ons halen, namelijk

$$de \equiv 1 \pmod{\varphi(n)},$$

zien we dus dat $\varphi(n)$ bekend moet zijn. De enige manier om $\varphi(n) = (p-1)(q-1)$ te bereken is door de priemgetallen p en q te kennen. De geheime sleutel zou dus ook het paar (p, q) of een combinatie



Figuur 3.3: RSA cryptosysteem

met d kunnen zijn, omdat d direct volgt uit (p,q) met het algoritme van Euclides. Aangezien ze n kent, betekent het wegens de Hoofdstelling van de rekenkunde dat ze deze moet factoriseren om zeker op p en q uit te komen. De veiligheid van RSA leunt dus op de moeilijkheidsgraad van het factoriseren van n. Dit wordt soms ook aangeduid als het factorisatieprobleem van gehele getallen, of in het geval van RSA, priemfactorisatie probleem:

Priemfactorisatie Probleem (PFP)**.** *Gegeven een $n \in \mathbb{N}$, vind priemgetallen $p_i \in \mathbb{N}$ zodanig dat*

$$n = \prod_{i=1}^{m} p_i^{e_i}, \quad e_i \in \mathbb{N}.$$

Voor RSA geldt 'slechts' dat $m = 2$, maar afhankelijk van de grootte van n kan dit zelfs voor een product van enkel 2 priemgetallen redelijk lang duren. Een klein voorbeeld van priemgetallen p en q rond de 128 bits:

```
sage: starttime=time.time();
....: a = 128
....: p = next_prime(2^(a)); q = next_prime(2^(a-4))
....: n = p*q
....: print 'n =', str(factor(n))+'.'
....: print 'Het factoriseren van n duurde', str(time.time()-starttime), 'seconden.'
....:
n = 212676479325586539664609129644855132 83 * 340282366920938463463374607431768211507.
Het factoriseren van n duurde 321.125560045 seconden.
```

Omgerekend duurde het proces dus > 5 minuten. De keuze van p en q in het voorbeeld zorgt ervoor dat $p \neq q$. Een groter voorbeeld levert een teleurstellend resultaat op:

```
sage: starttime=time.time();
....: a = 256
....: p = next_prime(2^(a)); q = next_prime(2^(a-4))
....: n = p*q
....: print 'n =', str(factor(n))+'.'
....: print 'Het factoriseren van n duurde', str(time.time()-starttime), 'seconden.'
```

```
....:

n =  ***   Warning: MPQS: number too big to be factored with MPQS,
        giving up.
```

Als we $p = q$ zouden kiezen, dan gaat de factorisatie beduidend sneller:

```
sage: starttime=time.time();
....: a = 128
....: p = next_prime(2^(a)); q = next_prime(2^(a))
....: n = p*q
....: print 'n =', str(factor(n))+'.'
....: print 'Het factoriseren van n duurde', str(time.time()-starttime), 'seconden.'
....:
n = 340282366920938463463374607431768211507^2.
Het factoriseren van n duurde 0.0277450084686 seconden.
```

Hieruit is duidelijk te zien waarom $p \neq q$ gekozen moet worden. Bovendien wordt aangeraden dat p en q een kleine hoeveelheid van cijfers verschillen, $(p-1)$ en $(q-1)$ grote priemfactoren hebben en dat $\gcd(p-1, q-1)$ klein gekozen moet worden, om bescherming te bieden tegen gevorderde factorisatie algoritmes [26].

Voorbeeld van RSA met ASCII

Laten we een voorbeeld bekijken van encryptie en decryptie van een bericht met behulp van RSA. Ons doel is om de volgende plaintext te coderen:

> Wanneer gaan we cryptografie met elliptische krommen in werking zien?

Eerst moeten we priemgetallen p, q kiezen die ervoor zorgen dat n groot genoeg is voor een lastige factorisatie. We kiezen p, q zodanig dat n rond de 256 bits is. De volgende lijst van getallen ontstaat:

$$\begin{aligned} p &= 18446744073709551629 \\ q &= 340282366920938463463374607431768211507 \\ n &= 6277101735386680768259460193179866442067009288836208394903 \\ \varphi(n) &= 6277101735386680767919177826258927978585187937330730631768. \end{aligned}$$

Deze waarden hebben we berekend met behulp van Sagemath. We zullen deze ook gebruiken om een getal e copriem met $\varphi(n)$ te vinden. Daarnaast volgt d:

```
sage: e = ZZ.random_element(phi)
....: while gcd(e, phi) != 1:
....:     e = ZZ.random_element(phi)
....: print 'e =', e
....:
e = 215870677027309072204288373089502325043709374285176530813
sage: d = inverse_mod(e, phi); d
443598005770834035992564005928758863550915693415964428028 5
sage: mod(d*e, phi)
1
```

We hebben dus nu alle sleutels berekend:

$$\text{publiek} = \begin{cases} n &= 627710173538668076825946019317986644206700928883620839 4903 \\ e &= 215870677027309072204288373089502325043709374285176530813 \end{cases}$$

$$\text{geheim} = \begin{cases} p &= 18446744073709551629 \\ q &= 340282366920938463463374607431768211507 \\ d &= 443598005770834035992564005928758863550915693415964428028 5 \end{cases}$$

Nu alle sleutels bekend zijn, kunnen we ons bezighouden met het omzetten van ons bericht naar cijfers. Eén manier om dit te doen, is door gebruik te maken van ASCII-codering. ASCII is een code om tekst te kunnen schrijven als getallen, zoals we nu beogen. Zo zijn de ASCII-equivalenties van de hoofdletters L en Y respectievelijk de getallen 076 en 089. Om bijvoorbeeld het bericht LY om te zetten met ASCII, schrijven we het samengevoegde getal 076089, wat kortweg 76089 wordt. Om tekst af te lezen van de getallen hakken we vanaf de staart het grote getal telkens op in blokken van drie cijfers. Een voorbeeld met het getal 75082079077077069:

$$75 \mid 082 \mid 079 \mid 077 \mid 077 \mid 069 \implies K \mid R \mid O \mid M \mid M \mid E$$

Tot nu zijn enkel hoofdletters gebruikt, maar het is ook mogelijk om kleine letters en spatie te gebruiken zoals in ons bericht dat gecodeerd moet worden. Een belangrijk detail hebben we nu echter niet specifiek belicht, namelijk dat het gecodeerde bericht kleiner moet zijn dan n ($b < n$). Het kan dus zijn dat een bericht b als ASCII getal groter wordt dan n. Om alsnog te voldoen aan deze voorwaarde, schrijven we b als *concatenatie* van groepen b_i, zodanig dat $b_i < n$ voor alle i. We splitsen het grote getal op vanaf de staart in blokken van getallen die kleiner zijn dan n. De ASCII code voor onze plaintext is

$$\begin{aligned} &0870971101101011011140321030970971100321191010320991141211121161111031140 \\ &9710210510103210910111603210110810810511211610511509910410103210711411110 \\ &9109101110032105110032119101114107105110103032122105101110063 \end{aligned}$$

Dit getal is beduidend groter dan n, dus hakken we deze stukken op in de volgende blokken b_i:

$$\begin{aligned} b_1 &= 0870971101101011011140321030970971 10 \\ b_2 &= 0321191010320991141211121161111031140971021051010321091 01 \\ b_3 &= 1160321011081081051121161051150991041010321071141111091 09 \\ b_4 &= 1011100321051100321191011141071051101030321221051011100 63 \end{aligned}$$

waarbij $b = b_1 b_2 b_3 b_4$ de concatenatie. Het te versleutelen bericht b kan dan per b_i versleuteld worden. De ciphertext $f_n(b)$ wordt in dit geval dan $f_n(b) = (f_n(b_1), f_n(b_2), f_n(b_3), f_n(b_4))$ waarbij

$$\begin{aligned} f_n(b_1) &\equiv b_1^e \pmod{n} \\ f_n(b_2) &\equiv b_2^e \pmod{n} \\ f_n(b_3) &\equiv b_3^e \pmod{n} \\ f_n(b_4) &\equiv b_4^e \pmod{n} \end{aligned}$$

Met SageMath berekenen we deze waarden. Let daarbij op dat we de eerste cijfers 0 van b_1 en b_2 weglaten:

```
sage: fnb1 = power_mod(b1, e, n); print 'fnb1 =', fnb1
....: fnb2 = power_mod(b2, e, n); print 'fnb2 =', fnb2
....: fnb3 = power_mod(b3, e, n); print 'fnb3 =', fnb3
```

```
....: fnb4 = power_mod(b4, e, n); print 'fnb4 =', fnb4
....:
fnb1 = 52444440163434684652734550969133391337688602827733333959935
fnb2 = 27409376154644230557952561471713152794935740370358642486 85
fnb3 = 26538430535350506657315303728941245921059451289326804214 69
fnb4 = 21229185723728357409519232978335200891169264724980349127 13
```

Dus de ciphertext is $f_n(b) = (f_n(b_1), f_n(b_2), f_n(b_3), f_n(b_4))$. Om te bevestigen dat deze ontcijferd kan worden met de geheime sleutel, berekenen we $f_n(b)^d \pmod{n}$ coördinaatsgewijs:

```
sage: B1 = power_mod(fnb1, d, n); print 'B1 =', B1
....: B2 = power_mod(fnb2, d, n); print 'B2 =', B2
....: B3 = power_mod(fnb3, d, n); print 'B3 =', B3
....: B4 = power_mod(fnb4, d, n); print 'B4 =', B4
....:
B1 = 870971101101011011140321030970971 10
B2 = 32119101032099114121112116111103114097102105101032109101
B3 = 11603210110810810511211610511509910410103210711411110910 9
B4 = 10111003210511003211910111410710511010303212210510111006 3
```

Deze stemmen allemaal overeen met de b_i's die in het begin zijn gedefinieerd. Directe omzetting met ASCII geeft:

$$\begin{aligned} b_1 &= \text{Wanneer gaan}| \\ b_2 &= |\text{ we cryptografie me}| \\ b_3 &= |\text{t elliptische kromm}| \\ b_4 &= |\text{en in werking zien?} \end{aligned}$$

waarbij | aangeeft waar de aparte b_i's aan elkaar geplakt moeten worden. Het factoriseren van n zal met deze keuze van p en q zal enorm veel tijd kosten. Het is echter niet noodzakelijk om p en q altijd erg groot te kiezen. Het hangt compleet af van de doeleinden: een niet te grote maar zeker niet te kleine n is bruikbaar voor een paar uren. Het is dus mogelijk om telkens nieuwe waarden van p en q te kiezen rond dezelfde orde om steeds veiligheid te 'garanderen' voor een paar uur.

Een mogelijke manier om het in stukken verdelen van de ASCII code te voorkomen is het opnieuw kiezen van de waarden p en q. De enige reden dat we dit moesten doen, is omdat $b > n$ was in dit geval waardoor na modulo rekenen het ASCII getal niet meer precies het oorspronkelijke bericht weergaf. Als we dus ervoor kunnen zorgen dat $b < n$, dan vermijden we de extra moeite van het hakken in stukken. Wetende dat elk karakter in het plaintext bericht omgezet kan worden naar een combinatie van drie cijfers, is het dus mogelijk om te berekenen hoe groot p en q gekozen moeten worden. Het plaintext bericht bevatte 69 karakters inclusief spaties en vraagteken. Dat betekent dat het bericht als ASCII getal zou bestaan uit $69 \cdot 3 = 207$ cijfers. Als n een getal is van minstens 207+1 cijfers (digits), zou het ASCII getal altijd hoogstens evenveel cijfers bevatten. Een manier om dat te bewerkstelligen is om gebruik te maken van het volgende: elk geheel positief getal K kan begrensd worden door

$$10^{c-1} \leq K \leq 10^c - 1,$$

waarbij c het aantal cijfers van K. We mogen logaritmen nemen bij de ongelijkheden zonder dat het de tekens omdraait, omdat $\log(x)$ een stijgende functie is voor $x > 0$. Daaruit volgt dat

$$c-1 \leq \log_{10}(K) \leq \log_{10}(10^c - 1) \implies c-1 \leq \log_{10}(K) < c \implies c-1 = \lfloor \log_{10}(K) \rfloor,$$

omdat $\log(x)$ stijgend is. We vinden dus dat het aantal cijfers c in een getal K gelijk is aan

$$c = \lfloor \log_{10}(K) \rfloor + 1,$$

waarin $\lfloor x \rfloor$ de *floor functie* is: de floor van x is het grootste gehele getal kleiner dan x; zo is $\lfloor \pi \rfloor = 3$ en $\lfloor 2.71828182846\ldots \rfloor = 2$. Zo hebben we dus een formule gevonden voor het aantal cijfers van een getal. We kunnen dus p, q zodanig kiezen:

$$c = 208 \implies \lfloor \log_{10}(n) \rfloor = 207 \implies \lfloor n \rfloor = 10^{207}.$$

Kies p, q dusdanig dat n minstens even groot is als 10^{207}.

Samenvattend sommen we de stappen op van het RSA cryptosysteem:

1. Alice kiest p, q priem en verschillend en berekent $n = pq$.
2. Alice kiest een e zodanig dat $1 < e < \varphi(n)$ en $\gcd(e, \varphi(n)) = 1$.
3. De publieke sleutel die Alice stuurt naar Bob is (n, e).
4. Alice berekent d: de multiplicatieve inverse van e modulo $\varphi(n)$. Dit vormt – samen met p en q – haar geheime sleutel.
5. Bob versleutelt zijn bericht b door $b^e \pmod{n}$ te berekenen en stuurt deze naar Alice.
6. Alice ontcijfert b door $(b^e)^d \pmod{n}$ te berekenen.

Eén van de grote vragen in de informatica is nog steeds of elk getal ontbonden kan worden in priemfactoren binnen polynomiale tijd op een niet-kwantumcomputer. De verwachte looptijd van het best bekend algoritme voor factorisatie is van

$$L_n[1/3, C] = \mathcal{O}\left(\exp\left((C + o(1)) \sqrt[3]{\log} \sqrt[3]{(\log[\log(n)])^2}\right)\right),$$

voor n het te factoriseren getal, $C = \sqrt[3]{64/9}$ en heet de *getallenlichaamzeef*.

3.2.3. Digital Signature Algorithm

In de besproken gevallen van DH en RSA zijn we impliciet uitgegaan van het feit dat de informatiewisseling tussen Alice en Bob vlekkeloos verloopt. In de praktijk kan het echter zijn dat Eve een boodschap onderschept, aanpast en vervolgens verstuurt en laat lijken alsof dat het originele bericht was. Daarom speelt het kunnen verifiëren van de afzender ook een rol. Alice zou bijvoorbeeld een kenmerkende handtekening kunnen achterlaten in de boodschap, die Eve dan aanpast en naar Bob stuurt. De informatie wordt zo dan niet gegeven zoals het oorspronkelijk was en dat is ongunstig voor de communicatie. Dit is een reden om *digital signatures*, digitale handtekeningen, te implementeren in een bericht. Deze maakt gebruik van zogeheten *hashfuncties*.

Hashfuncties die worden gebruikt in de cryptografie voldoen aan een aantal voorwaarden, maar laten we eerst het idee achter hashfuncties proberen te bevatten. Net als alle andere functies stuurt een hashfunctie een input naar een output, maar zijn daar wel een aantal eigenschappen die belicht moeten worden. Ten eerste is de output, *hash*, van hashfuncties altijd van dezelfde grootte; de hashfunctie MD5 heeft een hash van 128 bits en hashfunctie SHA-256 heeft outputs van 256 bits. De input, *message*, ofwel bericht, kan van willekeurige grootte zijn, wat een hashfunctie erg krachtig maakt. Het aantal karakters in een bericht maakt dus geen verschil in de grootte van de hash. Je zou hashfuncties wellicht min of meer kunnen vergelijken met een denkbeeldige machine die op de vuilnisbelt een collectie troep omzet naar een kubus gemaakt van troep. Voor een grotere collectie van troep wordt de kubus massiever, maar voor een kleinere collectie wordt de groep dusdanig

uitgezet door de machine. Op deze manier blijft het eindproduct, de kubus, telkens van dezelfde grootte, maar ziet hij er anders uit afhankelijk van wat je er allemaal instopt. Een hashfunctie H die gebruikt worden in de cryptografie dienen te voldoen aan de volgende eigenschappen

- Voor een bericht b kan de hash $H(b)$ relatief snel berekend worden.
- H is een *one-way* functie: als een beeld $H(b)$ gegeven is, dan is het lastig om b te vinden.
- H is in hoge mate *collision-free*: het is lastig om verschillende berichten $b_1 \neq b_2$ te vinden zodanig dat $H(b_1) = H(b_2)$.

Merk op dat het niet zo hoeft te zijn dat H injectief is. Vanwege de willekeur in de grootte van het bericht en een output van gegeven grootte, is het niet te garanderen dat H injectief is. Collision-free impliceert dat Eve niet eenvoudig een ander bericht kan produceren met dezelde hash als het oorspronkelijke bericht. Zo is de hashfunctie SHA-1 'gekraakt' door het CWI in Amsterdam in samenwerking met Google. Zij hebben twee verschillende documenten geproduceerd met dezelfde hash, waarna ze claimen dat het mogelijk is om nu twee verschillende documenten te creëren met dezelfde hash. Als voorbeeld geven ze een huurovereenkomst die een huurder tekent, denkende dat het een contract is met lage rente, terwijl de ondertekening nagemaakt kan worden in een contract met hoge rente [30]. One-way zorgt ervoor dat Eve het bericht van Alice of Bob niet kan achterhalen. Aan de lezer de taak om deze voorwaarden te koppelen aan de denkbeeldige machine op de vuilnisbelt.

Nu we hashfuncties hebben gedefinieerd kunnen we deze gebruiken om digital signatures te bekijken. We willen dat het mogelijk is om te kunnen verifiëren of een handtekening daadwerkelijk van de oorspronkelijke afzender komt. Alice en Bob willen dus met elkaar communiceren wetende dat Eve op de loer ligt om berichten te onderscheppen en te wijzigen. Net als bij DH en RSA spelen een aantal parameters een rol in DSA. Deze zijn voornamelijk gesplitst in de publieke en geheime sleutel enerzijds en sleutels voor de handtekeningen anderzijds. De beschreven methode is een korte omschrijving van de methodiek van het National Institue of Standards and Technology (NIST) [14] . Ten eerste is er een hashfunctie H nodig. Vervolgens kiezen Alice en Bob priemgetallen p, q waarbij geldt dat $q|p-1$; dit betekent automatisch dat $q < p$. De keuze voor de grootte van p, q zijn net als in RSA een maat voor veiligheid. Het NIST heeft een aantal standaardparen van bitlengten voor p, q gekozen die zij hanteren [14]. Daarnaast hebben we een voortbrenger g met orde q modulo p nodig van de groep $\mathbb{F}_q^\times$. Een methode die gesuggereerd wordt om zo een g te vinden, is om

$$g \equiv a^{(p-1)/q} \pmod{p}$$

te stellen zodat

$$g^q \equiv 1 \pmod{p}$$

waar q dus ook het kleinst mogelijke getal moet zijn. $(p-1)/q$ is welgedefinieerd als exponent, omdat $q|p-1$ per definitie. Het drietal (p, q, g) vormt op deze manier de parameters voor de handtekeningen. Aangezien deze publiekelijk bekend zijn, zouden we dit dus ook deel van de publieke sleutel kunnen noemen. Dat zullen we echter niet doen. De publieke sleutel e volgt uit een geheime sleutel d waarbij $0 < d < q$ en

$$e \equiv g^d \pmod{p}.$$

Alice wil een bericht b ondertekenen. Ze onderneemt de volgende stappen

1. Kies k zodat $1 < k < q$ willekeurig en bereken $r \equiv (g^k \mod p) \not\equiv 0 \pmod{q}$. Indien $r \equiv 0$ moet een nieuwe waarde voor k gekozen worden.
2. Bereken $s \equiv k^{-1}(H(b) + dr) \not\equiv 0 \pmod{q}$. Indien $s \equiv 0$ moet k opnieuw gekozen worden. k^{-1} kan met uitgebreid algoritme van Euclides berekend worden.

3. Ze ondertekent haar bericht b met (r,s), mogelijk in de vorm (b,r,s).

Bob ontvangt nu $(b,\tilde{r},\tilde{s})$ en is benieuwd of $\tilde{r}=r$ en $\tilde{s}=s$. We kijken naar het proces van verifiëren. Wegens de definities van r en s kunnen we de handtekening per direct verwerpen als $0<\tilde{r},\tilde{s}<q$ niet geldt, omdat r,s modulo q zijn berekend. Bob ondergaat nu de volgende procedure:

1. Bereken $w\equiv \tilde{s}^{-1} \pmod{q}$, mogelijk met uitgebreide Euclides.
2. Bereken $x\equiv H(b)\,w \pmod{q}$.
3. Bereken $y\equiv \tilde{r}\,w \pmod{q}$.
4. Bereken $z\equiv (g^x e^y \mod p) \pmod{q}$.

De handtekening is van Alice als en slechts als $z=\tilde{r}$. Om na te gaan of dit inderdaad geldt, moeten we dus aan kunnen tonen dat daadwerkelijk geldt dat $r=z$. Dit kan uiteraard gedaan worden door aan te tonen dat

$$r=(g^x e^y \mod p) \pmod{q}.$$

Alice berekende $s\equiv k^{-1}(H(b)+dr) \pmod{q}$. We kunnen linksvermenigvuldigen met k en vervolgens rechtsvermenigvuldigen met s^{-1}. Herinner echter dat $s^{-1}\equiv w \pmod{q}$, zodat

$$k\equiv H(b)\,w+d\,r\,w \pmod{q}.$$

Met de rekenregel in modulair rekenen

$$m=n \pmod{p} \implies g^m\equiv g^n \pmod{p}, \quad g \text{ willekeurig geheel getal},$$

vinden we dus ook dat

$$g^k\equiv g^{H(b)w}g^{drw}\equiv g^{H(b)w}e^{rw}\equiv g^x e^y \pmod{p}.$$

Dus volgt er dat

$$r\equiv (g^k \mod p) \pmod{q}\equiv (g^x e^y \mod p) \pmod{q}\equiv z.$$

DSA is gebaseerd op een eerder bekende procedure om handtekening te sturen, namelijk via het systeem bedacht door ElGamal. Net als bij ElGamal geldt er dat het belangrijk is om k willekeurig te kiezen voor elk bericht k, omdat anders de geheime sleutel x berekend kan worden. Als k bijvoorbeel twee keer als dezelfde waarde wordt gekozen voor verschillende berichten b_1 en b_2, krijgen we waarden s_1 en s_2:

$$\begin{aligned} s_1&\equiv k^{-1}[H(b_1)+dr] \pmod{q}\\ s_2&\equiv k^{-1}[H(b_2)+dr] \pmod{q}\end{aligned}$$

Dit vormt een stelsel van congruenties. We kunnen de tweede vergelijking aftrekken van de eerste en krijgen

$$\begin{aligned} s_1-s_2&\equiv k^{-1}[H(b_1)+dr-H(b_2)-dr] \pmod{q}\\ s_1-s_2&\equiv k^{-1}[H(b_1)-H(b_2)] \pmod{q}\\ k&\equiv [H(b_1)-H(b_2)]\cdot(s_1-s_2)^{-1} \pmod{q}\end{aligned}$$

Merk op dat k dus berekend kan worden door $H(b_i), s_i$ te kennen voor $i = 1,2$. Deze zijn echter bekend voor iedereen, dus ook voor Eve! Dat betekent dus dat Eve k kan achterhalen. Deze kan weer berekend worden als volgt

$$\begin{aligned} s &\equiv k^{-1}[H(b) + dr] \pmod{q} \\ ks &\equiv H(b) + dr \pmod{q} \\ dr &\equiv ks - H(b) \pmod{q} \\ d &\equiv [ks - H(b)]r^{-1} \pmod{q} \end{aligned}$$

$k, s, H(b)$ zijn allen weer bekend bij Eve en dus is x makkelijk te berekenen. Wegens de definitie van r, kan eenvoudig worden nagegaan of k hetzelfde is gekozen. Als er namelijk twee ondertekende berichten $(b_1, r_1, s_1), (b_2, r_2, s_2)$ zijn met dezelfde waarde voor k, dan geldt $r_1 = r_2$. Bij een overeenkomende waarde voor r en verschillende berichten $b_1 \neq b_2$ kan de handtekening van Alice worden gekopieerd door Eve. Het is zelfs mogelijk gebleken om x te berekenen als enkele bits van k bekend zijn bij verschillende handtekeningen. Daar zullen we niet verder op ingaan in dit verslag, maar we verwijzen naar [13, 23] voor de geïnteresseerde lezer.

3.3. ECC

Nu we een idee hebben over cryptografie en cryptosystemen, kunnen we de informatie combineren tot *elliptic curve cryptography* (ECC), ofwel cryptografie met elliptische krommen.

3.3.1. ECDH

De meest voorkomende toepassing van ECC is *Elliptic Curve Diffie-Hellman* (ECDH). Zoals de naam doet vermoeden, combineert deze het principe van DH key-exchange met de algebraïsche eigenschappen van een elliptische kromme E. Als we de stappen van DH weer erbij halen, dan is ECDH enkel een kwestie van enkele regels anders definiëren:

1. Alice en Bob laten publiekelijk weten over welk lichaam $\mathbb{F}_q$ en welke kromme E ze werken en kiezen $P \in E(\mathbb{F}_q)$.

2. Alice en Bob kiezen respectievelijk geheime getallen α en β in $\mathbb{F}_q$

3. Alice berekent $A = \alpha P \pmod{q}$, Bob berekent $B = \beta P \pmod{q}$.

4. Ze delen deze A en B publiekelijk met elkaar.

5. Alice berekent $\alpha B \pmod{q}$ en Bob berekent $\beta A \pmod{q}$.

6. Ze komen gezamenlijk uit op $K = \alpha\beta P \pmod{q}$.

Merk op dat in ECDH de sleutels A, B geen getallen zijn, maar coördinaten op $E(\mathbb{F}_q)$. De keuze van het lichaam $\mathbb{F}_q$ en het punt P is van belang voor de veiligheid van het systeem. Het DLP voor ECDH kan geformuleerd worden als het volgende probleem:

Discrete Logaritme Probleem (ECDLP)**.** *Gegeven zijn een elliptische kromme $E, \mathbb{F}_q$ en $P, nP, \in E(\mathbb{F}_q)$ en $\alpha \in \mathbb{F}_q$, bereken n.*

In § 3.4 zullen we kijken naar algoritmen om DLP's te kraken. Deze zijn gebaseerd op specifieke situaties en zullen dus ieder beter werken in een bijzonder scenario. Laten we nu een voorbeeld van ECDH bekijken.

Voorbeeld van ECDH

Reinier Bröker schreef in 2007 een artikel voor *Kennislink* over het gebruik van elliptische krommen ter bescherming van persoonsgegevens [8]. Hij schreef over de aanwezigheid van elliptische krommen in mobiele telefoons. Normaliter zijn de artikelen die geplaatst worden op Kennislink erg toegankelijk. Zo heeft Bröker het principe van ECDH uitgelegd op een zeer begaanbare wijze, zelfs zonder de naam te noemen! Als een voorbeeld van de toepassing van ECDH, zullen wij voortborduren op de informatie die Bröker publiceerde. In het artikel staan geen parameters gegeven, maar die kunnen we zelf bedenken of gebruiken van bekende krommen. Dat laatste zullen we doen: we gebruiken de kromme genaamd *Curve25519* gemaakt door Daniel Bernstein waarover hij zei: "An attacker who spends a billion dollars on special-purpose chips to attack Curve25519, using the best attacks available today, has about 1 chance in 10000000000000000000000000000 of breaking Curve25519 after a year of computation" [5].

In principe is Curve25519 een *Montgomery kromme*, maar vanwege haar specifieke parameters kan deze ook geschreven worden in Weierstrass vorm

$$y^2 = x^3 + 486662x^2 + x \quad \text{over} \quad \mathbb{F}_q,$$

met q priem en

$$\begin{aligned} q &= 57896044618658097711785492504343953926634992332820282019728792003956564819949 \\ &= 2^{255} - 19. \end{aligned}$$

We nemen het punt P als

$$(9, 14781619447589544791020593568409986887264606134616475288964881837755586237401)$$

Bröker zegt dat elke mobiele telefoon een elliptische kromme heeft en dat deze kenbaar is voor de mobiel zelf en de provider. Bovendien is het punt P ook bekend bij beiden. De bescherming – in dit geval van kosten voor een telefoongesprek – komt door de aanwezigheid van een unieke identificatiecode c. De identificatiecode c wordt bij het bellen verstuurd naar de provider, die dan weet wie er gebeld heeft en wie dus moet betalen. De uitwisseling van c gaat via ECDH. Ze gebruiken namelijk de gemeenschappelijke geheime sleutel K die ontstaat om de versleutelde identificatiecode te sturen. Daartoe berekent de mobiel αP en de provider βP. Voor onze demonstratie kiezen we $\alpha, \beta \in [2, k-2]$ willekeurig met $kP = \infty$ in SageMath.

$$\begin{aligned} \alpha &= 18295981418567853541299959981264436918409303688586051599074289913076725973334 \\ \beta &= 33159953887268969072570100757700001133344253073794180014983089178220382083444 \end{aligned}$$

De mobiele telefoon berekent nu αP en de provider berekent βP. Vervolgens sturen ze deze naar elkaar en vermenigvuldigen de mobiele telefoon en de provider elkaars gestuurde punt met α respectievelijk β, zodat $K = \alpha\beta P$. Dan

$$\begin{aligned} K &= (x_K, y_K) \\ x_K &= 977408522374678787265308695909037140612956319611956105638865474859313180427 \\ y_K &= 123265322731633206680165172022020556686303525995864972380399330920699094249 49 \end{aligned}$$

Er zijn nu een aantal dingen die we kunnen doen met K om de unieke identificatiecode te versleutelen. Bröker benoemt als voorbeeld het optellen van de x_K bij c [8]. Zo wordt het bericht b dat de mobiele telefoon naar de provider verstuurt

$$b = x_K + c.$$

Als de provider c wil achterhalen, hoeft deze enkel $b - x_K$ te berekenen. Uiteraard kent de provider b, maar ook K is bekend wegens ECDH. We kunnen enorm veel variëren met het versleutelen van c. We noemen een aantal mogelijkheden

- $b = (x_K \cdot y_K \pmod q) + c$
- $b = (|x_K - y_K| \pmod q) + c$
- $b = (x_K \cdot y_K^{-1} \pmod q) + c$
- $b = (16x_K + 7y_K + 2018 \pmod q) + c$

Merk op dat we voor alle berekeningen met x_K en y_K modulo q rekenen. Zo voorkomen we dat we informatie lekken door middel van de grootte van de waarde. Uiteraard zijn er oneindig veel mogelijkheden. Het belangrijkste is dat de telefoon en de provider dezelfde versleuteling hanteren. Aangezien ECDH symmetrische encryptie gebruikt, gaat decryptie op precies dezelfde (inverse) wijze.

3.3.2. ECDSA

Net als bij ECDH, is *Elliptic Curve Digital Signature Algorithm* een combinatie van de eigenschappen van punten op een elliptische kromme en het DSA. Een toepassing van ECDSA is op dit moment erg populair, namelijk in de *Bitcoin*. Deze gebruikt de kromme genaamd *secp256k1* gegeven door

$$\begin{aligned} y^2 &= x^3 + 0x + 7 \quad \text{over} \quad \mathbb{F}_p \\ q &= 2^{256} - 2^{32} - 977 \\ &= 115792089237316195423570985008687907853269984665640564039457584007908834671663 \end{aligned}$$

Bitcoin is een type *cryptocurrency*, cryptogeld: het is geld in digitale vorm. Dat maakt het conceptueel al lastiger om te begrijpen, omdat de meest abstracte vorm voor de gemiddelde burger voorheen de pinpas was. Het bestaan van transacties bij online bankieren is bij de Bitcoin hetzelfde. We zullen niet gedetailleerd ingaan op Bitcoins en de werking, maar zullen één deelonderwerp kort belichten. Dat gaat namelijk over het uitwisselen van Bitcoins.

Bitcoins zijn in te wisselen voor geld, dus je kan er in principe ook mee betalen. Als je zelf een betaling in de vorm van Bitcoins zou willen, heb je daar een zogeheten Bitcoin-adres voor nodig. Dit dient in principe dus als een manier om geld te storten op iemands digitale Bitcoin portemonnee. Bij grote en kleine – maar vooral grote – transacties is het daarom natuurlijk gewenst dat je geen geld stuurt naar de verkeerde persoon. Je hebt daarbij in termen van Bitcoins een *verzendadres* en een *ontvangstadres*. Deze kan je in principe één op één relateren aan de publieke sleutel (ontvangst) en geheime sleutel (verzend). Je kan er als volgt over denken: het is niet erg als je geld ontvangt (publiekelijk kenbaar), maar je wilt er wel zeker van zijn dat jij de enige bent die Bitcoins kan verzenden vanaf jouw eigen portemonnee (geheim). Om dit te bewerkstelligen, maakt men bij transacties met Bitcoins gebruik van ECDSA.

Het idee van ECDSA is hetzelfde als eerder beschreven. We gebruiken nu echter termen die aansluiten op elliptische krommen. Alice wil nog steeds een bericht b ondertekenen. Zij en Bob zijn het eens over een elliptische kromme E over $\mathbb{F}_q$ en een punt G op de kromme met priemorde n. De geheime sleutel d kiest ze zodat $1 \leq d < n$. Ze berekent vervolgens $P = dG$. De publieke sleutel wordt dan $(E(\mathbb{F}_q), n, G, Q)$ en de geheime sleutel uiteraard d.

Om het bericht b te ondertekenen volgt ze de volgende procedure:

1. Kies een willekeurig getal k zodanig dat $1 \leq k < n$ en berekent $R = kG$. Merk op dat G een punt op de kromme is.
2. Bereken $s \equiv k^{-1}(b + d\,x_G) \pmod n$.
3. Verzend het ondertekende bericht (m, R, s)

De notatie is vergelijkbaar met die van klassieke DSA, maar R is in dit geval een punt op de kromme. Bob krijgt nu $(m, \tilde{R}, \tilde{s})$ binnen en vraagt zich weer af of deze van Alice zijn. Hij wil dus nagaan dat $\tilde{R} = R$ en $\tilde{s} = s$. Hij ondergaat de procedure:

1. Bereken $w_1 = \tilde{s}^{-1} b \pmod{n}$
2. Bereken $w_1 = \tilde{s}^{-1} x_G$
3. Bereken $Z = w_1 G + w_2 P$

De handtekening is van Alice als en slechts als $Z = R$. Om na te gaan dat dit zo is, moeten we dus aan kunnen tonen dat $w_1 G + w_2 P = R$. Met de rekenregel voor modulo rekenen

$$a \equiv b^{-1} \pmod{n} \implies a^{-1} \equiv b \pmod{n}$$

vinden we met $s \equiv k^{-1}(b + d x_G) \pmod{n}$ dat

$$\begin{aligned} Z &= w_1 G + w_2 P = s^{-1} bG + s^{-1} x_G P \\ &= s^{-1}(bG + x_G P) = s^{-1}(bG + x_G dG) \\ &= s^{-1}(b + x_G d)G \\ &= kG \\ &= R \end{aligned}$$

3.3.3. ECC tegenover RSA

De oplettende lezer heeft wellicht opgemerkt dat RSA niet is besproken in combinatie met elliptische krommen. Een verklaring hiervoor is dat dit ook niet gebruikt wordt. Zoals gezegd leunt RSA op de moeilijkheidsgraad van het factoriseren van grote priemgetallen. Er is geen bekende manier om elliptische kromme hierin te verwerken. Daarentegen zijn DH en DSA wel besproken in combinatie met elliptische krommen, omdat deze leunen op de moeilijkheidsgraad van DLP's. Het loont dus om meer te weten te komen over het mogelijke kraken van DLP's. Tot nu toe hebben we enkel DLP's proberen op te lossen met brute-force. Het mag duidelijk zijn dat dit algoritme uiterst onvoordelig is vanwege eerder genoemde redenen, waaronder de benodigde tijd.

Er zijn een aantal redenen om ECC te gebruiken ten faveure van RSA, maar voornamelijk ligt het aan de bitgrootte. Voor dezelfde mate van veiligheid kan de volgende tabel worden gemaakt.

n-bit veiligheid	RSA sleutels	ECC sleutels
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Tabel 3.1: Benodigde minimale bitgrootte van sleutels van RSA en ECC volgens NIST [18, 19]

Veiligheid van n-bits betekent dat er 2^n opties moeten worden nagegaan waarna het systeem niet meer veilig is. Volgens Tabel 3.1 zijn er voor 128-bit veiligheid in RSA sleutels nodig van 3072 bits, maar in ECC sleutels van 'slechts' 256 bits. Er is dus op te merken dat RSA sleutels in het algemeen veel langere sleutels nodig hebben vergeleken met ECC sleutels voor dezelfde mate van veiligheid. Het is erg gunstig wanneer de sleutels kleiner gekozen kunnen worden, omdat er veel minder rekenkracht nodig is. Voor meer details over Tabel 3.1 verwijzen we naar [18, 19].

Tegenwoordig wordt RSA nog steeds wereldwijd gebruikt en vaker dan ECC sinds het publiceren van PKCS#1 (Public-Key Cryptography Standard) in 1993. Sindsdien is deze bijgehouden en de laatste versie komt uit 2012 [27]. Veel van de patenten op RSA zijn verlopen in 2000, waardoor deze wereldwijd gebruikt kon worden. Veel patenten op ECC zijn nog steeds actief. We verwijzen naar [1] voor een lijst van een aantal patenten. Een andere mogelijke verklaring voor het gebruik van RSA in

plaats van ECC is het algemende idee dat RSA eenvoudiger is te begrijpen dan ECC. Voor RSA is enkel redelijke basale kennis nodig over modulo rekenen en enkele stellingen uit de getaltheorie. Voor elliptische krommen is de precieze achtergrondkennis lastig te bepalen, maar er zijn enorm veel onderdelen uit de abstracte algebra die toepasselijk zijn op elliptische krommen. Als laatste kan een verklaring zijn dat RSA simpelweg eerder werd gepubliceerd dan ECC. "Never change a winning horse", blijken meeste instanties ook te denken: zolang RSA nog goed werkt – zolang er geen efficiëntere algoritmes worden gevonden voor factorisatie – is er in principe geen grote reden om over te stappen. Het garandeert echter nooit complete veiligheid, omdat er zomaar een nieuw uiterst efficiënt algoritme bedacht kan worden.

3.4. Algoritmes voor DLP's

Voor een onderschepper is het probleem duidelijk:

Discrete Logaritme Probleem (ECDH)**.** *Gegeven zijn een elliptische kromme E over $\mathbb{F}_q$ en $\alpha P, \beta P \in E(\mathbb{F}_q)$ en $\alpha\beta \in \mathbb{F}_q$. Bereken $\alpha\beta P$.*

Merk op dat het doel niet zozeer is om α of β te berekenen, maar om achter de geheime gemeenschappelijke sleutel te komen. Met de kennis van tegenwoordig kennen we echter geen andere manier om dat te doen, behalve door α of β te berekenen. Dus zolang we geen efficiënt algoritme vinden, is het voor men onvermijdelijk om deze α dan wel β te moeten achterhalen. Dit is eigenlijk een specifiek geval van ECDLP, namelijk voor een onderschepper in een DH situatie. Algemener kijken we naar dit probleem wat op hetzelfde neerkomt

Discrete Logaritme Probleem (ECDLP)**.** *Gegeven zijn een elliptische kromme E over $\mathbb{F}_q$ en $P, Q \in E(\mathbb{F}_q)$ met $Q = nP$ en $n \in \mathbb{F}_q$. Bereken n.*

We zullen een aantal algoritmes bespreken die in de loop der jaren zijn bedacht om DLP's te kraken. De meeste algoritmes werken voor elke eindige groep, dus in het bijzonder voor $E(\mathbb{F}_q)$ en voor het ECDLP.

3.4.1. Brute-force

We hebben bij klassieke DH en bij RSA laten zien hoelang brute-force kan duren en bij ECC wordt dat er niet korter op. Er zijn echter wel verschillende methodes van brute-force. De meest *naïeve* methode die bekend is, hebben we eigenlijk al laten zien bij klassieke DH. Daar werkten we niet met elliptische krommen, maar ging het om modulo rekenen na exponentiatie:

$$g^k \equiv P \pmod{p}. \tag{3.1}$$

We berekenden g^k om vervolgens de rest modulo p te berekenen. Voor hele grote waardes van k zullen de meeste computers een runtime error geven wegens gebrek aan geheugen bijvoorbeeld. Dat kan voorkomen worden door de methode die we toegepast hebben bij het RSA voorbeeld. Veel programma's hebben een optie om de uitdrukking in (3.1) efficiënt te berekenen door middel van *herhaald kwadrateren*. Vergelijk dit met haar analoog double-and-add voor elliptische krommen besproken in § 2.2.4. SageMath heeft op zichzelf echter al ingebouwde algoritmes om dit voor elliptische krommen zo redelijk snel te berekenen. Een mogelijke implementatie voor brute-force op ECDLP in SageMath is als volgt:

```
sage: def BF(P,Q):
....:     starttijd = time.time()
....:     for i in srange(order(P)):
....:         if i*P == Q:
....:             print 'Het getal n zodat nP = Q is n = ', str(i)+'.'
```

```
....:             print 'Brute-force duurde', time.time()-starttijd, 'seconden.'
....:             break
....:         else:
....:             continue
```

Brute-force werkt in het algemeen voor alle welgedefinieerde groepen. Het is dus niet zo dat deze methode alleen werkt voor ECDLP's in het bijzonder. De verwachte looptijd is $\mathcal{O}(k)$ met k de orde van P, omdat er maximaal k mogelijheden nagegaan worden.

3.4.2. Baby-step Giant-step

De methode *Baby-step Giant-step* en is gepuclibeerd door Daniel Shanks [28] en heeft veel weg van de brute-force methode. Gegeven is $E(\mathbb{F}_q)$ en een punt $P, Q \in E(\mathbb{F}_q)$ en de orde k van P. Het algoritme gaat als volgt

1. Kies $m = \lfloor \sqrt{k} \rfloor$ en bereken mP.
2. Bereken iP en sla deze op in een lijst voor $0 \leq i < m$. Dit zijn de Baby-steps die worden afgelegd, omdat er telkens P opgeteld wordt.
3. Bereken $Q - jmP$ voor $j = 0, \ldots, m-1$ en stop zodra er een waarde overeenkomt met een waarde uit de lijst van stap 2. Dit zijn de Giant-steps, omdat er telkens mP wordt afgetrokken, wat dus m keer zoveel is als in de baby-steps.
4. Als deze overeenkommen geldt er $iP = Q - jmP$ en is $Q = nP$ met $n \equiv i + jm \mod k$.

De juistheid van dit algoritme kan als volgt worden nagegaan. Wegens de eerste stap geldt er dat $0 \leq n < m^2$. Als we n schrijven als $n = n_0 + mn_1$ dan is $n \equiv n_0 \pmod{m}$. Dat wil zeggen $0 \leq n_0 < m$. Laat $n_1 = (n - n_0)/m$, dan volgt $0 \leq n_1 < m$. Nu geldt er dat

$$Q - n_1 mP = nP - n_1 mP = n_0 P$$

voor $i = n_0$ en $j = n_1$. Zoals te zien is gebruikt deze methode meer geheugen, omdat de waardes opgeslagen moeten worden. Het voordeel is echter wel dat er tijd wordt bespaard. Het idee van een methode als deze is niet nieuw. Het wordt een *meet-in-the-middle* methode genoemd. Een implementatie van Baby-step Giant-step is er al in SageMath. Deze kan gebruikt worden door de functie `bsgs()`. Baby-step Giant-step werkt overigens ook in elke welgedefinieerde groep. Voor een groep met een multiplicatieve bewerking hanteren we gewoonweg andere notatie. De verwachtte looptijd is wegens de implementatie $\mathcal{O}(\sqrt{k})$.

3.4.3. Pollard's rho

Pollard's ρ is gepubliceerd door John Pollard in 1978 [25]. Als het punt $P \in E(\mathbb{F}_q)$ priemorde k heeft, dan is het idee van Pollard's ρ om verschillende getallenparen (a, b) en $(\tilde{a}, \tilde{b})$ met te vinden zodanig dat

$$aP + bQ = \tilde{a}P + \tilde{b}Q.$$

We kunnen deze namelijk omschrijven op deze manier:

$$\begin{aligned} aP + bnP &= \tilde{a}P + \tilde{b}nP \\ P(a + bn) &= P(\tilde{a} + \tilde{b}n) \\ a + bn &= \tilde{a} + \tilde{b}n \qquad (P \neq \infty) \\ (b - \tilde{b})n &= \tilde{a} - a \\ n &\equiv (\tilde{a} - a) \cdot (b - \tilde{b})^{-1} \pmod{k} \end{aligned}$$

De notie dat k een priemorde is, was nodig om de laatste congruentie welgedefinieerd te laten zijn. Het verschil $b - \tilde{b}$ heeft voor k niet priem niet zozeer een multiplicatieve inverse. De vraag resteert hoe deze $a, \tilde{a}, b, \tilde{b}$ gevonden worden.

Eén manier [12] om dat te doen is om een functie $f : \langle P \rangle \to \langle P \rangle$ te definiëren. Hierbij is $\langle P \rangle$ de ondergroep van $E(\mathbb{F}_q)$ voortgebracht door P. Deze f dient voor elk punt $X = aP + bQ \in \langle P \rangle$ en $0 \leq a, b < k$ de waarde $f(X) = \tilde{X}$ snel te kunnen berekenen. Hierbij is $\tilde{X} = \tilde{a}P + \tilde{b}Q$ en $0 < \tilde{a}, \tilde{b} < k$. Dat kan gedaan worden door een partitie te maken van $\langle P \rangle$ van L verzamelingen $\{V_1, \ldots, V_L\}$ waarbij $|V_i| \approx |V_j|$ voor $i \neq j$. Als we nu de partitie functie $h : \langle P \rangle \to \{1, \ldots, L\}$ gegeven door $h(X) = j\mathbf{1}_{V_j}(X)$ en $0 \leq a_j, b_j < n$ met $j \in \{1, \ldots, L\}$ definiëren, vinden we eindelijk dat

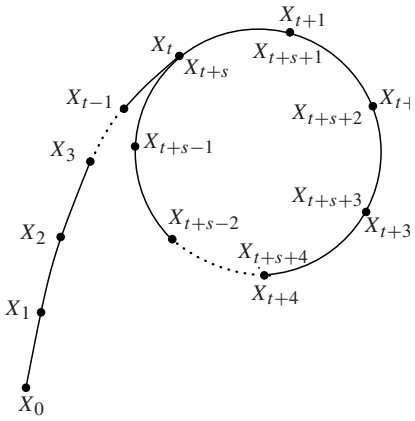
$$f(X) = X + a_j P + b_j Q.$$

Het berekenen van $f(X)$ levert nu het volgende op

$$\begin{aligned} f(X) = aP + bP + a_j P + b_j Q &= (a + a_j)P + (b + b_j)Q \\ &= \tilde{a}P + \tilde{b}Q \end{aligned}$$

en dus is $f(X) = \tilde{X}$ met $\tilde{a} \equiv a + a_j \pmod{k}$ en $\tilde{b} \equiv b + b_j \pmod{k}$ zoals we wilden.

De functie f is iteratief; hij stuurt elk punt in de cyclische groep $\langle P \rangle$ naar een rij van punten $\{X_i\}_i \geq 0$ waarvoor $f(X_i) = X_{i+1}$, maar de rij zal na $s + t$ stappen zich blijven herhalen. Hierin is X_t de eerste waarde die herhaald wordt, namelijk zo dat $X_t = X_{t+s}$. De herhaling komt vanwege de orde van de groep: $E(\mathbb{F}_q)$ is eindig en $\langle P \rangle$ heeft orde k die $\#E(\mathbb{F}_q)$ deelt. De naam van deze methode is afgeleid van de vorm die ontstaat na deze herhaling, namelijk die lijkt op ρ. Pollard's ρ heeft onge-



Figuur 3.4: Pollard's ρ waar t het aantal stappen van de staart weergeft en s de stappen van de herhaling [12].

veer dezelfde looptijd als Baby-step Giant-step, maar heeft minder geheugen nodig. De verwachte looptijd is $\mathcal{O}(\sqrt{n})$ [31].

3.4.4. Pohlig-Hellman

De methode van *Pohlig-Hellman* werd gepubliceerd in 1978 [25] door Stephen Pohlig en Martin Hellman. Het idee van Pohlig-Hellman is vergelijkbaar met het idee van Schoof's algoritme. Als $P, Q \in E(\mathbb{F}_q)$ en $Q = nP$ en k de orde is van P, dan willen we ten eerste de priemfactorisatie van k weten, dus alle p_i priem zodat

$$\prod_{i=1}^{m} p_i^{e_i} = k.$$

Hierbij zijn e_i dus de exponenten van elk priemgetal; voor $8 = 2^3$ is $p_1 = 2$ en $e_1 = 3$. Schoof's algoritme werkte enkel met kleine priemfactoren, maar hier willen we dus de waardes n_i berekenen

zodanig dat

$$n \equiv n_i \pmod{p_i^{e_i}}.$$

De Chinese reststelling zegt dan dat al deze congruenties een unieke waarde $n \pmod{k}$ opleveren. De methode werkt en dat kunnen we zien door te kijken naar $p_i^{e_i}$. We willen n_i berekenen en dat kan als volgt [12].

Schrijf n_i als

$$n_i \equiv c_0 + c_1 p_i + c_2 p_i^2 + \cdots + c_{e_i-1} p_i^{e_i-1}. \pmod{p_i^{e_1}} \tag{3.2}$$

Hierin zijn de coëfficiënten $c_j \in \{0, p_i - 1\}$. De coëfficiënten kunnen in het algemene geval berekend worden op de volgende wijze. Laten we beginnen met c_0 voor p_i. Laat $P_0 = (k/p_i)P$ en $Q_0 = (k/p_i)Q$.

$$\begin{aligned} Q_0 = \left(\frac{k}{p_i}\right) Q &= \left(\frac{k}{p_i}\right)\left[c_0 + c_1 p_i + \cdots + c_{e_i-1} p_i^{e_i-1}\right] P \\ &= c_o \left(\frac{k}{p_i}\right) P + \left(c_1 + c_2 p_i + \cdots + c_{e_i-1} p_i^{e_i-2}\right) \underbrace{kP}_{=\infty} \\ &= c_0 \left(\frac{k}{p_1}\right) P = c_0 P_0 \implies c_0 P_0 = Q_0 \end{aligned}$$

Er onstaat bij het achterhalen van c_0 dus eigenlijk een 'nieuw' ECDLP, maar in een kleinere groep. Preciezer, de ECDLP gaat nu over $\langle P_0 \rangle$, de cyclische groep voortgebracht door P_0. Laat $Q_1 = (k/p_i^2)(Q - c_0 P)$ voor c_1. Er volgt

$$\begin{aligned} Q_1 = \left(\frac{k}{p_i^2}\right)(Q - c_0 P) &= \left(\frac{k}{p_i^2}\right)(n - c_0) P \\ &= \left(\frac{k}{p_i^2}\right)\left(c_1 p_i + c_2 p_i^2 + \cdots + c_{e_i-1} p_i^{e_i-1}\right) P \\ &= c_1 \left(\frac{k}{p_i}\right) P + \left(c_2 p_i + c_3 p_i^2 + \cdots + c_{e_i-1} p_i^{e_1-2}\right) \underbrace{kP}_{=\infty} \\ &= c_1 \left(\frac{k}{p_i}\right) P \implies c_1 P_0 = Q_1 \end{aligned}$$

Het algemene geval kunnen we dus opschrijven als

$$c_s P_0 = Q_s \quad \text{waar} \quad Q_s = \frac{k}{p_i^{s+1}} \left(Q - \sum_{l=0}^{s-1} c_l p^l P \right), \quad s \geq 1.$$

De Pohlig-Hellman methode reduceert dus in principe het grotere ECDLP in ECDLP over kleinere groepen ($\langle P_0 \rangle$). Deze zijn relatief snel op te lossen, zelfs met brute-force.

Pohlig-Hellman kan dus erg krachtig zijn om ECDLP's op te lossen, maar dan moet de orde k wel te ontbinden zijn in relatief kleine priemfactoren. Het minst gunstige geval is wanneer k zelf ook priem is. Dan kunnen we deze methode als ongeschikt beschouwen. Voor de veiligheid betekent dit dat de orde van de groep gekozen moet worden zodanig dat het een grote priemfactor bevat. De verwachte looptijd is $\mathcal{O}(\sum_{i=1}^{m} e_i(\log(n) + \sqrt{p_i}))$ volgens Menezes et al. [20], maar aangezien de kleine priemfactoren relatief snel gaan, kunnen we zeggen dat het $\mathcal{O}(\sqrt{p_m})$ is. Een implementatie van Pohlig-Hellman staat in principe al in SageMath.

3.4.5. Index Calculus

De methode van *Index Calculus* is het meest efficiënt bij het oplossen van DLP's. Deze heeft namelijk een verwachte looptijd van $L_q[1/2, C] = O\left(\exp\left((C + o(1))\sqrt{\log(q)}\sqrt{\log[\log(q)]}\right)\right)$ [12, 21], waarbij $C > 0$ een constante. De enige grote kanttekening hierbij is dat deze niet werkt over groepen in het

algemeen: we kunnen deze methode niet zomaar gebruiken voor onder andere ECDLP's in $E(\mathbb{F}_q)$. Via een andere methode, zoals *MOV* en *Frey-Rück*, is het echter wel mogelijk om ECDLP's uiteindelijk op te lossen. Dat is te lezen in § 3.5.

Een groep waar Index Calculus wel werkt naar behoren is de multiplicatieve groep $(\mathbb{Z}/q\mathbb{Z})^\times$. De methode werkt in het algemeen ook voor een priemmacht q, maar we bespreken het voor $(\mathbb{Z}/p\mathbb{Z})^\times$ met p priem. Laat g met orde $p-1$ de voortbrenger zijn van $(\mathbb{Z}/p\mathbb{Z})^\times$. We willen het volgende DLP oplossen

$$g^n \equiv a \pmod{p}$$

voor $a \in (\mathbb{Z}/p\mathbb{Z})^\times$. Noteer voor n de logaritme met grondtal g van a als $n = \log_g(a) \pmod{p-1}$. De orde van g is namelijk $p-1$. Voor deze notatie geldt net als bij de continue logaritme dat

$$\log_g(a_1 a_2) \equiv \log_g(a_1) + \log_g(a_2) \pmod{p-1} \quad \forall a_1, a_2 \in (\mathbb{Z}/p\mathbb{Z})^\times. \tag{3.3}$$

Kies een basis grondtallen $\mathscr{B} = \{p_1, \ldots, p_m\}$ met p_i priem voor $i = 1, \ldots, m$. Het idee is nu om getallen te vinden die machten zijn van g en zodat deze te schrijven zijn als product van p_i's, dus

$$g^j = \prod_{i=1}^{m} p_i^{e_i(j)}, \quad x \in [1, p-1] \quad \text{en} \quad e_i \geq 0.$$

Met deze uitdrukking kunnen we echter ook x noteren als logaritmes met grondtal g wegens Vergelijking (3.3).

$$\begin{aligned} j &\equiv \log_g\left(\prod_{i=1}^{n} p_i^{e_i(j)}\right) \pmod{p-1} \\ j &\equiv \sum_{i=1}^{m} \log_g\left(p_i^{e_i(j)}\right) \pmod{p-1} \\ j &\equiv \sum_{i=1}^{m} e_i(j) \log_g(p_i) \pmod{p-1} \end{aligned} \tag{3.4}$$

Om de waarde van alle $\log_g(p_i)$'s te achterhalen, kunnen we een stelsel van n lineair onafhankelijke vergelijkingen samenstellen van de vorm van Vergelijking (3.4). Het stelsel dat deze $\log_g(p_i)$'s opleveren moeten voldoende zijn om $\log_g(a)$ te berekenen. De belangrijkste stap van Index Calculus is dan ook om een zeer geschikte $\mathscr{B}$ te vinden. Deze moet ten eerste namelijk zó gekozen worden dat a gefactoriseerd kan worden door p_i's. Uiteraard willen we het liefst dat n zo klein mogelijk gekozen wordt, omdat het scheelt in rekenwerk. Voor een te kleine n kunnen we echter nooit het DLP oplossen [12, 21]. Dat is de afweging de gemaakt moet worden bij Index Calculus. Voor de uiteindelijke waarde van n berekenen we $g^x a$ voor x geheel totdat deze gefactoriseerd kan worden door $p_i \in \mathscr{B}$, dus

$$g^x a = \prod_{i=1}^{n} p_i^{e_i}.$$

Om n te berekenen nemen de de $\log_g$ aan beide kanten en vinden we

$$\begin{aligned} \log_g(g^x a) &\equiv \log_g\left(\prod_{i=1}^{n} p_i^{e_i}\right) \pmod{p} \\ \underbrace{\log_g(g^x)}_{=x} + \log_g(a) &\equiv \sum_{i=1}^{m} \log_g\left(p_i^{e_i}\right) \pmod{p} \\ n = \log_g(a) &\equiv \sum_{i=1}^{m} e_i \log_g\left(p_i\right) - x \pmod{p}. \end{aligned}$$

Index Calculus voor $(\mathbb{Z}/p\mathbb{Z})^\times$ maakt dus gebruik van het feit dat elementen uit de groep gefactoriseerd kunnen worden in producten van priemgetallen. Voor $E(\mathbb{F}_q)$ kan dat echter niet zomaar worden gedaan. Victor Miller beweert zelfs dat 'het zeer onwaarschijnlijk is dat "index calculus" ooit zal werken op elliptische krommen.' [22].

Dat concludeert de methode van Index Calculus en de algoritmes voor dit verslag. We vatten de informatie over hun looptijd samen in de volgende tabel.

Methode	*Looptijd*	*Opmerking*
Brute-force	$\mathcal{O}(k)$	kost in het algemeen veel tijd
Baby-step Giant-step	$\mathcal{O}(\sqrt{k})$	gebruikt relatief veel geheugen
Pollard's ρ	$\mathcal{O}(\sqrt{k})$	gebruikt minder geheugen dan Baby-step Giant-step
Pohlig-Hellman	$\mathcal{O}(\sqrt{p_m})$	ongeschikt indien $k = \prod_{i=1}^{m} p_i^{e_i}$ priem waarbij p_i groot zijn
Index Calculus	$L_q[1/2, C]$	werkt niet direct over $E(\mathbb{F}_q)$

Tabel 3.2: Looptijden van verschillende methodes voor ECDLP's, waar k de orde is van punt $P \in E(\mathbb{F}_q)$

3.5. Speciale methoden voor ECDLP

De methoden die eerder zijn genoemd zijn generiek; het ging niet per se om ECDLP's. Methodes die echter wel concreet voor elliptische krommen bedacht zijn, gelden voor een aantal type krommen in het bijzonder. Bovendien zijn er methodes ontworpen die gebruikt maakt van pairings, zoals we die kennen van § 2.4.

3.5.1. MOV

De MOV methode werd gepubliceerd door Albert Menezes, Tasuaki Okamoto en Scott Vanstone in 1993 [21]. Het idee van de MOV methode is om het ECDLP te transformeren naar DLP in eindige lichamen $\mathbb{F}_{q^m}^\times$ met behulp van pairings. Herinner namelijk dat het DLP in de multiplicatieve groep $\mathbb{F}_q^\times$ relatief eenvoudiger is op te lossen. Daarin bestaat namelijk de besproken Index Calculus, om DLP's op te lossen in subexponentiele tijd [12, 20, 31]. Het getal m wordt ook wel de *embedding graad* van $E(\mathbb{F}_q)$ genoemd. m is precies zo gedefinieerd als het kleinste positieve gehele getal is zodanig dat

$$\#E(\mathbb{F}_q) \mid q^m - 1. \tag{3.5}$$

Hierin komt $q^m - 1$ van de orde van $\mathbb{F}_q^\times$. Het is mogelijk om m telkens met 1 te verhogen, totdat (3.5) juist is. Het is echter verstandiger om te kijken naar alle mogelijke waarden voor m. Via (3.5) vinden we

$$q^m \equiv 1 \pmod{\#E(\mathbb{F}_q)}.$$

Herinner de Kleine stelling van Fermat, die zegt dat

$$q^{\#E(\mathbb{F}_q)-1} \equiv 1 \pmod{\#E(\mathbb{F}_q)}.$$

Dat betekent dat $m \mid \#E(\mathbb{F}_q) - 1$ en dus dat de priemfactorisatie van $\#E(\mathbb{F}_q) - 1$ alle mogelijkheden voor m geeft, omdat m de multiplicatieve orde is van q modulo $\#E(\mathbb{F}_q)$. Laten we nu kijken naar het algoritme. Laat $P, Q \in E(\mathbb{F}_q)$ met k de orde van P en $nP = Q$. We nemen aan dat q en k copriem zijn zodat $E[k] \cong (\mathbb{Z}/k\mathbb{Z})^2$. Kies m zodanig dat de k-torsie $E[k] \subseteq E(\mathbb{F}_{q^m})$. Zoals Washington [31] opmerkt zijn alle punten in de k-torsie bevat in de algebraïsche afsluiting $\overline{\mathbb{F}}_q = \bigcup_{i \geq 1} \mathbb{F}_{q^i}$. Dan is ook het codomein van de Weil pairing, $\mu_k = \{x \in \overline{\mathbb{F}}_q \mid x^k = 1\}$, bevat in $\mathbb{F}_{q^m}$. Dus

$$E[k] \subseteq E(\mathbb{F}_{q^m}) \quad \text{en} \quad \mu_k \subseteq \mathbb{F}_{q^m}. \tag{3.6}$$

Het algoritme gaat als volgt

1. Kies een willekeurig punt $T \in E(\mathbb{F}_{q^m})$.
2. Ga na of $T \notin E(\mathbb{F}_q)$ zit. Zoniet, kies $T \in E(\mathbb{F}_{q^m})$ opnieuw willekeurig.
3. Bereken de Weil pairingen $\xi_1 = e_k(P,T)$ en $\xi_2 = e_k(Q,T)$, maar $\xi_2 = \xi_1^n$ want
$$e_k(Q,T) = e_k(nP,T) = e_k(P,T)^n = \xi_1^n.$$
4. Er geldt dat $\xi_1, \xi_2 \in \mathbb{F}_{q^m}^\times$ zitten wegens (3.6) en $\xi_1, \xi_2 \in \mu_t$ met t de orde van T. Los dus nu het DLP probleem
$$\xi_2 \equiv \xi_1^n$$
op in $\mathbb{F}_{q^m}^\times$. Dat zou kunnen met Index Calculus.

Een elliptische kromme die goed bestand is tegen de MOV methode heeft dat m groot gekozen moet worden; krommen die niet goed bestand zijn hebben kleine m. Eén type kromme is daar uiterst ongeschikt voor. Dat zijn namelijke de *superinguliere* elliptische krommen. Een elliptische kromme $E(\mathbb{F}_q)$ met $q = p^r$ en p priem is supersingulier als

$$\#E(\mathbb{F}_q) = q + 1 - b, \qquad b \equiv 0 \pmod{p}.$$

In Hoofdstuk 4 zullen we over twee supersinguliere kromme het ECDLP proberen op te lossen. Met het karakteristieke polynoom van de Frobenius is te bewijzen dat $m = 2$ voldoet voor supersinguliere elliptische krommen [31]. De verwachte looptijd is gelijk aan die van Index Calculus, namelijk $L_q[1/2, C]$

3.5.2. Frey-Rück

MOV gebruikte de Weil pairing, maar ook de Tate pairing kan gebruikt worden bij het kraken van DLP's. Sterker nog, de Tate pairing kan gebruikt worden in situaties waarbij de Weil pairing dat niet kan. De Frey-Rück methode werd gepubliceerd door Gerhard Frey en Hans-Georg Rück in 1994 [11].

Herinner dat ook de Tate pairing τ_k afbeeldt naar μ_k. Dat geeft al een subtiele hint naar het feit dat het idee van de Frey-Rück methode op hetzelfde idee is gebaseerd: ECDLP reduceren naar DLP over $\mathbb{F}_{q^m}^\times$. Washington [31] zegt dat voor de Tate pairing ook $\mu_k \subseteq \mathbb{F}_q^\times$, maar in tegenstelling tot de Weil pairing is er enkel één punt $P \in E(\mathbb{F}_q)$ nodig van orde k. De Weil pairing had namelijk ook als voorwaarde dat de k-torsie $E[k] \subseteq E(\mathbb{F}_q)$. Tevens zou volgens Barreto et al. de Tate pairing ook sneller te berekenen zijn [4].

3.5.3. Smart

De Smart methode werd gepubliceerd door Nigel Smart in 1999 [29]. De boodschap die hij wilde geven, was dat *abnormale* krommen van de vorm

$$\#E(\mathbb{F}_q) = q$$

niet geschikt zijn voor cryptografisch gebruik. Hij ontwikkelde namelijk een manier om deze sneller op te lossen dan MOV op supersinguliere krommen [31]. Het idee van de methode was om ECDLP weer te reduceren tot een eenvoudiger probleem. We schrijven voor nu $\mathbb{F}_q = \mathbb{F}_p$. Een homomorfisme van $E(\mathbb{F}_p)$ naar de additieve groep $\mathbb{F}_p$ zou bijvoorbeeld een DLP in $\mathbb{F}_p$ opleveren. Er bestaat echter geen homomorfisme die rechtstreeks dat doel bereikt. Wel kunnen we daar min of meer komen met een zogeheten *lift*. Bekijk de volgende afbeelding

$$f : E(\mathbb{F}_p) \to E(\mathbb{Q}_p)$$

Hierin schrijven we voor $\mathbb{Q}_p$ de *p-adische getallen.* Een p-adisch getal kan worden uitgedrukt als oneindige som van de vorm

$$\sum_{i=-m}^{\infty} c_i p^i = c_{-m}p^{-m} + \cdots + c_0 + \cdots + c_m p^m + \ldots$$

Voor de getallen waarvoor geldt dat $c_i = 0$ voor $i < 0$ – getallen zonder negatieve machten van p – worden ook wel de p-adische gehele getallen, waarbij we de notatie $\mathbb{Z}_p$ hanteren. De afbeelding f lift dus als het ware punten van E over $\mathbb{F}_q$ naar $\tilde{E}$ over $E(\mathbb{Q}_p)$, maar definiëren we wel zodanig dat $\tilde{E}$ modulo p weer E wordt. Het idee [29] van de lifts naar $E(\mathbb{Q}_p)$ is dat een 'log-afbeelding' niet bestaat in $\mathbb{F}_p$, maar wel bestaat in $\mathbb{Q}_p$. Laat $P, Q \in E(\mathbb{F}_q)$ zodat $Q = nP$. Noem de lifts van P en Q respectievelijk $\tilde{P} \in E(\mathbb{Q}_p)$ en $\tilde{Q} \in E(\mathbb{Q}_p)$ zodanig dat

$$\tilde{Q} = n\tilde{P} + R, \quad R \in E_1(\mathbb{Q}_p).$$

Waarbij $E_1(\mathbb{Q}_p)$ de kern is van de modulo p afbeelding $g : E(\mathbb{Q}_p) \to E(\mathbb{F}_p)$. We definiëren dus $E_1(\mathbb{Q}_p) = \ker(g)$. De kern van g bestaat uit alle punten die na modulo p gelijk zijn aan ∞ in $E(\mathbb{F}_q)$. $\tilde{Q} = n\tilde{P}$ na modulo p, dus moet R wel naar het punt ∞ in $E(\mathbb{F}_q)$ worden gestuurd door g. Bovendien is $E_1(\mathbb{Q}_p) \cong \mathbb{Z}_p$ als $p > 2$, wat te zien is met de definitie van p-adische getallen. Dit isomorfisme noemen we de p-adische logaritme ψ_p gegeven door

$$\psi_p((x,y)) \equiv \frac{-x}{y} \pmod{p^2}$$

en doet precies wat we zouden willen en verwachten van een log functie. We moeten het probleem dus eerst nog op de een of andere manier in $E_1(\mathbb{Q}_p)$ zien te krijgen. Als we nu kijken naar

$$\tilde{Q} - n\tilde{P} = R \in E_1(\mathbb{Q}_p), \tag{3.7}$$

kunnen we het feit gebruiken dat $\#E(\mathbb{F}_p) = p$ om verder te komen voor het berekenen van n. Aangezien $p\tilde{S} = \infty$ voor elke $\tilde{S} \in E(\mathbb{Q}_p)$, kunnen we het linker- en rechterlid van Vergelijking (3.7) vermenigvuldigen met p en zeggen dat

$$\underbrace{p\tilde{Q}}_{\in E_1(\mathbb{Q}_p)} - n \underbrace{p\tilde{P}}_{\in E_1(\mathbb{Q}_p)} = pR \in E_2(\mathbb{Q}_p).$$

Nu hebben we precies wat we willen. Aangezien $p\tilde{Q}, p\tilde{P} \in E_1(\mathbb{Q}_p)$, kunnen we ψ_p toepassen zodat

$$\psi_p(p\tilde{Q}) - n\psi(p\tilde{P}) = \psi_p(pR) \equiv 0 \pmod{p^2}.$$

Door nu op te lossen voor n vinden we voor $\psi_p(p\tilde{P}) \not\equiv 0 \pmod{p}$ dat

$$n \equiv \frac{\psi_p(p\tilde{Q})}{\psi_p(p\tilde{P})} \pmod{p}$$

Dit concludeert de Smart methode. De verwachte looptijd is van $\mathcal{O}(\log p)$ [29]. We vatten de informatie over deze methoden speciaal voor ECDLP samen in de volgende tabel.

Methode	*Looptijd*
MOV/Frey-Rück	$L_q[1/2, C]$
Smart	$\mathcal{O}(\log p)$

Tabel 3.3: Looptijden van de besproken methodes voor ECDLP's

4

Voorbeelden van het kraken van ECDLP's

We zullen proberen om een aantal krommen te bouwen die geschikt zijn om ECDLP's te kraken met enkele methoden genoemd in Hoofdstuk 3. Deze typen krommen moeten dus vermeden worden bij het opstellen van een kromme die in de praktijk wordt gebruikt. Specifieker zullen we kijken naar krommen van de volgende types

- $\#E(\mathbb{F}_q)$ is een *glad* getal. Dat wil zeggen dat de priemfactorisatie van $\#E(\mathbb{F}_q)$ enkel bestaat uit kleine priemgetallen. Pohlig-Hellman is zeer efficiënt indien dit het geval is.
- $\#E(\mathbb{F}_q) = q+1$: supersinguliere krommen. ECDLP's kunnen redelijk snel worden gekraakt als we MOV of Frey-Rück toepassen.
- $\#E(\mathbb{F}_q) = q$: abnormale krommen. De Smart methode kan ECDLP snel kraken op deze krommen.

4.1. Complexe vermenigvuldiging (CM)

De methode van *complex multiplication* (CM), complexe vermenigvuldiging, kan ons helpen om krommen te genereren in de vorm die we zoeken. We zullen niet uitgebreid ingaan op CM, maar we zullen de benodigde theorie benoemen en verwijzen de geïnteresseerde lezer naar [15, 31].

Als $\Lambda = \omega_1\mathbb{Z} + \omega_2\mathbb{Z}$ een rooster is, noemen we de verzameling

$$\{\alpha : \mathbb{C} : \alpha\Lambda \subset \Lambda\}$$

de endomorfismenring van Λ en noteren deze als $\mathrm{End}(\mathbb{C}/\Lambda)$. Er geldt bijvoorbeeld dat de verzameling van gehele getallen $\mathbb{Z} \subset \mathrm{End}(\mathbb{C}/\Lambda)$. Voor $\Lambda \in \mathbb{C}$ geldt er dat Λ een CM-rooster is als

$$\mathrm{End}(\mathbb{C}/\Lambda) \setminus \mathbb{Z} \neq \emptyset.$$

Een elliptische kromme E/Λ noemen we een CM-kromme als $\mathrm{End}(\mathbb{C}/\Lambda) \supsetneq \mathbb{Z}$. Een CM-kromme E heeft bovendien een endomorfismering $\mathrm{End}(E)$ isomorf aan een *deelring van gehelen* in $\mathbb{Q}(\sqrt{-d})$,

$$\mathrm{End}(E) \cong \mathbb{Z}\left[\frac{1+\sqrt{-d}}{2}\right] \qquad \text{als} \quad d \equiv 3 \pmod 4$$
$$\mathrm{End}(E) \cong \mathbb{Z}\left[\sqrt{-d}\right] \qquad \text{als} \quad d \equiv 1,2 \pmod 4$$

met $d \in \mathbb{Z}$. We zullen in principe werken met $d \equiv 3 \pmod 4$. Een *geheel* getal in deze context zijn getallen in $\mathbb{Q}(\sqrt{-d}) = \{a + b\sqrt{-d} \mid a, b \in \mathbb{Q}\}$ die oplossingen zijn van

$$1 \cdot x^n + a_{n-1}x^{n-1} + \cdots + a_0 = 0, \tag{4.1}$$

met de coëfficiënten $a_i \in \mathbb{Z}$. Het ismorfisme impliceert een kromme E van algemene Weierstrass-vergelijking met vaste coëfficiënten

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

Met de karakteristieke vergelijking van de Frobenius en de stelling van Hasse

$$\phi_q^2 - a_q\phi_q + q = 0, \qquad \#E(\mathbb{F}_q) = q + 1 - a_q,$$

kunnen we a_q zó krijgen dat we gewenste waarden voor $\#E(\mathbb{F}_q)$ afdwingen. Als we bijvoorbeeld $\#E(\mathbb{F}_q) = q + 1$ willen hebben, moeten we ervoor zorgen dat $a_q = 0$. De Frobenius is een endomorfisme dat werkt op E, dus deze zit ook in $\mathrm{End}(E)$. Dat betekent dat

$$\phi_q \in \mathbb{Z}\left[\frac{1+\sqrt{-d}}{2}\right] \implies \phi_q = m + n\frac{1+\sqrt{-d}}{2}$$

Herinner dat

$$q = \phi_q\widehat{\phi_q}, \tag{4.2}$$
$$a_q = \phi_q + \widehat{\phi_q}. \tag{4.3}$$

In dit geval is $\widehat{\phi_q} = \overline{\phi_q}$. Vanwege de complex-geconjugeerde is het zo dat

$$q = \phi_q\overline{\phi_q} = (-\phi_q)(-\overline{\phi_q}) \implies a_q = \phi_q + \widehat{\phi_q} \vee a_q = -(\phi_q + \widehat{\phi_q})$$

Als q echter geen factorisatie heeft van de vorm $q = \alpha\overline{\alpha}$ in $\mathbb{Z}[(1+\sqrt{-d})/2]$, dan is E supersingulier over $\mathbb{F}_q$. Met (4.2) en (4.3) vinden we

$$q = \left(m + n\frac{1+\sqrt{-d}}{2}\right)\left(m + n\frac{1-\sqrt{-d}}{2}\right) = m^2 + mn + \frac{1}{4}(d+1)n^2 \tag{4.4}$$
$$a_q = \left(m + n\frac{1+\sqrt{-d}}{2}\right) + \left(m + n\frac{1-\sqrt{-d}}{2}\right) = 2m + n \quad \text{of} \quad a_q = -(2m+n). \tag{4.5}$$

Merk op dat wegens $d \equiv 3 \pmod 4$, dat $d + 1 \equiv 0 \pmod 4$. Dat betekent dat $(d+1)/4$ een geheel getal is. Zo vinden we een stelsel vergelijkingen voor m, n en een priemgetal q. Uiteindelijk willen we q weten zodat het lichaam $\mathbb{F}_q$ waar we over werken duidelijk wordt.

$$q = m^2 + mn + \frac{1}{4}(d+1)n^2 \tag{4.6}$$
$$a_q = 2m + n = 0 \tag{4.7}$$

Laat bijvoorbeeld m variëren en bereken $n = -2m$ totdat er een q bestaat die voldoet. Voor onze voorbeelden willen we graag werken rond de 128 bits. Het is wegens beperkingen in SageMath echter niet gelukt om dit in alle gevallen zo te krijgen. Merk op dat voor $a_q \neq 0$ het zo kan zijn dat we een q vinden waarvoor a_q met factor -1 verschilt van de gewenste waarde. Om dat probleem op te lossen is het mogelijk te kijken naar de *kwadratische twist* E^d van E. Het idee is dat we kijken naar de kromme E^d met haar eigen Weierstrassvergelijking, waarvoor a_{q_d} de a_q is van E^d. De relatie tussen krommen E en E^d kan samengevat worden door

$$a_{q_d} = -a_q.$$

Dit concludeert de methode van het vinden een gewenste $\#E(\mathbb{F}_q)$ voor een CM-kromme voor dit verslag. Voor de lezer die geïnteresseerd is in meer details over het construeren van krommen met een specifiek aantal punten, verwijzen we naar het werk van Reinier Bröker. Zo heeft hij in 2006 een PhD thesis getiteld "Constructing elliptic curves of prescribed order" geschreven [7], maar ook een aantal andere gerelateerde artikelen gepubliceerd. Hier komt de theorie achter complexe vermenigvuldiging ook zeker aan bod, wat veel meer diepgaand is dan in dit verslag tentoongesteld.

4.2. ECDLP: Pohlig-Hellman

Om te beginnen zullen we ECDLP proberen te kraken via een generieke methode. SageMath heeft in de functie `discrete_log()` al Pohlig-Hellman geïmplementeerd. Peter Novotney heeft in 2010 [24] echter een illustratief stuk code geschreven, die wij hier ook zullen laten zien.

```
sage: def PohligHellman(P,Q):
....:     start = time.time()
....:     zList = list()
....:     conjList = list()
....:     rootList = list()
....:     n = P.order()
....:     factorList = n.factor()
....:     for facTuple in factorList:
....:         P0 = (ZZ(n/facTuple[0]))*P
....:         conjList.append(0)
....:         rootList.append(facTuple[0]^facTuple[1])
....:         for i in range(facTuple[1]):
....:             Qpart = Q
....:             for j in range(1, i+1):
....:                 Qpart = Qpart - (zList[j-1]*(facTuple[0]^(j-1))*P)
....:             Qi = (ZZ(n/(facTuple[0]^(i+1))))*Qpart
....:         zList.insert(i, discrete_log(Qi, P0, operation='+'))
....:         conjList[-1] = conjList[-1] + zList[i]*(facTuple[0]^i)
....:     print 'Het duurde', time.time()-start, 'seconden.'
....:     return crt(conjList, rootList)
```

Nu moeten we nog een vergelijking voor E en een lichaam $\mathbb{F}_q$ vinden waar we E over willen definiëren. Pohlig-Hellman is uiterst efficiënt indien $\#E(\mathbb{F}_q)$ ontbonden kan worden in enkel kleine priemgetallen. We spelen met de kennis over supersinguliere krommen en kiezen E zodanig dat $\#E(\mathbb{F}_q) = q+1$. Volgens Washington [31] is E met vergelijking

$$y^2 = x^3 + 1$$

supersingulier als en slechts als $q \equiv 2 \pmod 3$ en $q \geq 5$ priem. Nu we een vergelijking hebben en een voorwaarde, zoeken we een priemgetal q zodanig dat $q = 2p-1$ met p een glad getal. Zo volgt er dat $\#E(\mathbb{F}_q) = q+1 = 2p-1+1 = 2p$. We schrijven een stuk code die een q bepaalt met deze voorwaarden. Kies één relatief grotere priem 1607 – de dag van de eindvoordracht, min of meer zoals Arjen Lenstra deed [31] – en werk als volgt.

```
sage: def vindq(n):
....:     p = 1607
....:     for i in range(2,n):
....:         if Mod(p, next_prime(i)) == 0:
....:             continue
....:         else:
....:             p = p*next_prime(i)
....:             q = 2*p - 1
....:             if is_prime(q) == 1 and Mod(q, 3) == 2:
....:                 print 'p =', factor(p)
....:                 print '  =', p
....:                 print '2p - 1 = '+str(q)+' is een priemgetal van '
```

```
                                        +str(ceil(log(q, 2))+1)+' bits.'
....:                 print '2p - 1 = 2 (mod 3).'
....:                 print
....:             else:
....:                 continue
sage: vindq(300)
p = 3 * 5 * 7 * 11 * 13 * 17 * 19 * 23 * 29 * 31 * 37 * 41 * 43 * 47 * 53 *
    59 * 61 * 67 * 71 * 73 * 79 * 83 * 89 * 97 * 101 * 103 * 107 * 109 *
    113 * 127 * 131 * 137 * 139 * 149 * 151 * 157 * 163 * 167 * 173 * 179 *
    181 * 191 * 193 * 197 * 199 * 211 * 223 * 227 * 229 * 233 * 239 * 241 *1607
  = 20572907128501831137417396576009459818841525867074899888364806215179945745 2
    55834654603186771788389105 5
2p - 1 = 41145814257003662274834793152018919637683051734149799776729612430359 89
         14905116693092063735435767782109 is een priemgetal van 339 bits.
2p - 1 = 2 (mod 3).
```

De eerste resultaten laten we weg, omdat we een voorbeeld willen hebben van minstens 128-bit veiligheid. Zo krijgen we $\#E(\mathbb{F}_q) = 2p$. Om het ECDLP te simuleren, kunnen we gewoonweg een willekeurig punt P op de kromme en nemen we $Q = nP$ met $1 < n < k-1$ willekeurig. Het is echter aan te raden dat het interval $[2, k-2]$ zo groot mogelijk wordt gekozen. Dat kan gedaan worden door een eventuele voortbrenger van E te nemen in plaats van een willekeurig punt P. Achteraf verifiëren we of het algoritme daadwerkelijk die waarde voor n heeft gevonden die we zochten. Hierbij is k de orde van het punt P. We voorspellen dat dit proces erg snel zal gaan wegens de keuze van q.

```
sage: q = 41145814257003662274834793152018919637683051734149799776729612430359 89
          14905116693092063735435767782109
sage: E = EllipticCurve(GF(q), [0,1])
sage: P = E.gens()[0]; P
(256858941257406503337425163140538196025033439909751100322221020818004786803773 8
14730999390143100274419 : 34300310404210354248329215286009226479094298593518425 8
05266248834166526843166251929337449123003990105 7 : 1)
sage: k = P.order()
sage: n = ZZ.random_element(2, k-1)
sage: Q = n*P; Q
(898808926168562161518235065890721709141798767958798198149185082940772936757208 6
39818079655668761425 25 : 28648187209833138781989241541352810911954510894648197 63
11498292671057386530073792945260610615394466025 25 : 1)
sage: PohligHellman(P, Q)
Het duurde 4.27943301201 seconden.
208276713725918685525435311121463344474298415214848334224650199327690567255298 90
828671269928257095667
sage: n
208276713725918685525435311121463344474298415214848334224650199327690567255298 90
828671269928257095667
```

Deze kromme zou enkel afgaand op haar lichaam dus rond de 170-bit veiligheid moeten voorstellen, maar het ECDLP is wel erg snel opgelost, overigens wel zoals verwacht.

4.3. ECDLP: MOV

Voor de MOV methode zullen we twee voorbeelden laten zien. Herinner dat het ECDLP eenvoudig te kraken is met MOV indien E supersingulier is. Dat komt doordat $m = 2$ vaak al voldoende is voor

supersinugliere krommen:

$$q^2-1=(q-1)(q+1)\equiv 0 \pmod{q+1=\#E(\mathbb{F}_q)} \implies q+1 | q^2-1.$$

De implementatie van de MOV methode kan als volgt gedaan worden:

```
sage: def MOV(P,Q,E1,E2):
....:     start = time.time()
....:     q = E1.order()-1
....:     F1.<a> = GF(q)
....:     F2.<b> = GF(q^2)
....:     aa = F1.modulus().roots(F2)[0][0]
....:     phi = Hom(F1, F2)(aa)
....:     P2 = E2(phi(P.xy()[0]), phi(P.xy()[1]))
....:     Q2 = E2(phi(Q.xy()[0]), phi(Q.xy()[1]))
....:     T = E2.random_point()
....:     zeta1 = P2.weil_pairing(T, q+1)
....:     while zeta1.multiplicative_order() != P.order():
....:         T = E2.random_point()
....:         zeta1 = P2.weil_pairing(T, q+1)
....:     zeta2 = Q2.weil_pairing(T, q+1)
....:     print 'n = ', zeta2.log(zeta1)
....:     print 'Het proces duurde', time.time()-start, 'seconden.'
```

Over $\mathbb{F}_q$ met $q=2^n$

Washington [31] heeft laten zien dat een kromme E met vergelijking

$$y^2+y=x^3 \quad \text{over} \quad \mathbb{F}_2$$

supersingulier is met karakteristieke vergelijking voor de Frobenius

$$\phi_2^2+2=0. \tag{4.8}$$

Er geldt namelijk dat $E(\mathbb{F}_2)=\{(0,0),(0,1),\infty\}$ en dus dat $\#E(\mathbb{F}_2)=3$. Daaruit volgt met Hasse dat $a_2=2+1-3=0$ en dus is $E(\mathbb{F}_q)$ inderdaad supersingulier. Bovendien geeft Washington een formule voor een elliptische kromme E over een lichaamsuitbreiding van $\mathbb{F}_q$, namelijk $E(\mathbb{F}_{q^n})$

$$\#E(\mathbb{F}_{q^n})=q^n+1-(\alpha_q^n+\beta_q^n) \implies \#E(\mathbb{F}_{2^n})=2^n+1-(\alpha_2^n+\beta_2^n).$$

Hierin zijn α_2 en β_2 de nulpunten van Vergelijking (4.8). Schrijf

$$\alpha=\sqrt{-2}=\sqrt{2}i \quad \text{en} \quad \beta=-\sqrt{2}i.$$

Om E over $\mathbb{F}_{q^n}$ supersingulier te krijgen willen we dat $(\sqrt{2}i)^n+(-\sqrt{2}i)^n=0$. Dit gebeurt precies voor het geval dat n oneven is. Schrijf $n=2k+1$ met $k\in\mathbb{Z}$, dan

$$\begin{aligned}(\sqrt{2}i)^{2k+1}+(-\sqrt{2}i)^{2k+1} &= (\sqrt{2})^{2k+1}i^{2k+1}+(-\sqrt{2})^{2k+1}i^{2k+1}\\ &= \sqrt{2}(2)^k i^{2k+1}-\sqrt{2}(2)^k i^{2k+1}=0.\end{aligned}$$

We kunnen nu dus kijken naar lichaamsuitbreidingen van $\mathbb{F}_2$ en behouden het feit dat deze ook supersingulier zijn over de gegeven Weierstrassvergelijking, mits we de macht van 2 oneven kiezen. Voor SageMath gaat $q=2^{63}$ erg snel.

```
sage: q = 2^63
sage: F1.<a> = GF(q)
sage: F2.<b> = GF(q^2)
sage: E1 = EllipticCurve(F1, [0,0,1,0,0])
sage: E2 = EllipticCurve(F2, [0,0,1,0,0])
sage: P = E1.gens()[0]
sage: k = P.order()
sage: n = ZZ.random_element(2, k-1)
sage: Q = n*P
sage: MOV(P,Q,E1,E2)
n =  8312461621001202276
Het proces duurde 2.18109798431 seconden.
sage: n
8312461621001202276
```

Vanaf ongeveer 2^{87} kan SageMath de discrete logaritmen niet meer zo soepel berekenen. Voor 2^{85} werkt het echter nog wel.

```
sage: q = 2^85
sage: F1.<a> = GF(q)
sage: F2.<b> = GF(q^2)
sage: E1 = EllipticCurve(F1, [0,0,1,0,0])
sage: E2 = EllipticCurve(F2, [0,0,1,0,0])
sage: P = E1.gens()[0]
sage: k = P.order()
sage: n = ZZ.random_element(2, k-1)
sage: Q = n*P
sage: MOV(P,Q,E1,E2)
n =  430893604541551860572406
Het proces duurde 0.915363073349 seconden.
sage: n
430893604541551860572406
```

We zien dat het proces bij een groter uitbreidingslichaam sneller is gegaan. Dit kan op zich gebeuren, aangezien de implementatie afhangt van de keuze van n zodat $Q = nP$ en het kiezen van een willekeurig punt T dat voldoet aan onze voorwaarde.

Over $\mathbb{F}_q$ met $q = p$ priem

Washington [31] heeft ook uitgewerkt dat de kromme E met vergelijking

$$y^2 = x^3 + x \quad \text{over} \quad \mathbb{F}_q$$

supersingulier is als en slechts als $q = p \equiv 3 \pmod 4$ met p priem. We zoeken dus nog een p die voldoet aan deze voorwaarden en willen ter demonstratie het liefst een zo groot mogelijke p laten zien. We schrijven de volgende code.

```
sage: def vindp(p):
....:     for i in range(p, p+100000):
....:         if is_prime(i)==1 and Mod(i,4)==3:
....:             print 'p =', i
....:             break
....:
```

```
sage: vindp(2^84)
p = 19342813113834066795298819
```

Hierbij houden we rekening met het vermogen van SageMath om tot ongeveer 2^{84} snel te kunnen werken met discrete logaritmen. We simuleren een ECDLP met de gegeven vergelijking en over het lichaam $\mathbb{F}_p$.

```
sage: p = 19342813113834066795298819
sage: F1.<a> = GF(p)
sage: F2.<b> = GF(p^2)
sage: E1 = EllipticCurve(F1, [1,0])
sage: E2 = EllipticCurve(F2, [1,0])
sage: P = E1.gens()[0]
sage: k = P.order()
sage: n = ZZ.random_element(2, k-1)
sage: Q = n*P
sage: MOV(P,Q,E1,E2)
n =  21659710943863447627 4304
Het proces duurde 0.134943962097 seconden.
sage: n
21659710943863447627 4304
```

Voor een kromme die puur via haar parameters 42-bit veiligheid zou moeten geven, is de MOV methode erg efficiënt gebleken om het ECDLP in zeer korte tijd te kraken.

4.4. ECDLP: Smart

Een mogelijke implementatie van de Smart methode is wederom gemaakt door Novotney [24]. Novotney heeft een implementatie voor de lift van punten P en Q, maar ook de methode zelf.

```
sage: def HenselLift(P,p,prec):
....:     E = P.curve()
....:     Eq = E.change_ring(QQ)
....:     Ep = Eq.change_ring(Qp(p, prec))
....:     x_P, y_P = P.xy()
....:     x_lift = ZZ(x_P)
....:     y_lift = ZZ(y_P)
....:     x, y, a1, a2, a3, a4, a6 = var('x, y, a1, a2, a3, a4, a6')
....:     f(a1, a2, a3, a4, a6, x, y) = y^2 + a1*x*y + a3*y - x^3 -
          a2*x^2 -a4*x - a6
....:     g(y) = f(ZZ(Eq.a1()), ZZ(Eq.a2()), ZZ(Eq.a3()), ZZ(Eq.a4()),
          ZZ(Eq.a6()), ZZ(x_P), y)
....:     gDiff = g.diff()
....:     for i in range(1, prec):
....:         uInv = ZZ(gDiff(y = y_lift))
....:         u = uInv.inverse_mod(p^i)
....:         y_lift = y_life - u*g(y_lift)
....:         y_lift = ZZ(Mod(y_lift, p^(i+1)))
....:         y_lift = y_lift + O(p^prec)
....:     return Ep([x_lift, y_lift])

sage: def Smart(P, Q, p, prec):
```

```
....:     start = time.time()
....:     E = P.curve()
....:     Eqq = E.change_ring(QQ)
....:     Eqp = Eqq.change_ring(Qp(p, prec))
....:     P_Qp = HenselLift(P, p, prec)
....:     Q_Qp = HenselLift(Q, p, prec)
....:     p_times_P = p*P_Qp
....:     p_times_Q = p*Q_Qp
....:     x_P, y_P = p_times_P.xy()
....:     x_Q, y_Q = p_times_Q.xy()
....:     psi_P = -(x_P/y_P)
....:     psi_Q = -(x_Q/y_Q)
....:     k = psi_Q/psi_P
....:     k = Mod(k, p)
....:     print 'Het proces duurde', time.time()-start, 'seconden.'
....:     return k
```

Kies een CM-kromme E. We willen voor deze methode dat $a_q = 1$ zodat $\#E(\mathbb{F}_q) = q$, maar houden in gedachte dat we wellicht een kwadratische twist nodig hebben. Onze keuze gaat uit voor

$$\mathrm{End}(E) \cong \mathbb{Z}\left[\frac{1+\sqrt{-11}}{2}\right].$$

De vergelijking van E wordt dan gegeven door

$$y^2 + y = x^3 - x^2 - 7x + 10$$

Via Vergelijking (4.6) en Vergelijking (4.7) krijgen we

$$q = m^2 + mn + 3n^2$$
$$a_q = 2m + n = 1$$

We schrijven een kort stuk code en vinden een geschikte q.

```
sage: def vindenq(a):
....:     for m in range(a,a+1000000):
....:         n = 1 - 2*m
....:         q = m^2 + m*n + 5*n^2
....:         if is_prime(q) == 1:
....:             print 'q=', str(q)+' voldoet en is van ',
                        ceil(log(q,2))+1, ' bits.'
....:             break
....:         else:
....:             continue
....:
vindenq(2^125)
q=34375776492328245516372636174454222711017688276902037059431689742336519077843
voldoet en is van 256 bits.
```

We moeten echter nog wel controleren of we de goede a_q te pakken hebben. Dat kan snel gedaan worden in SageMath. We zien dat we er een factor -1 naast zitten en lossen dat op door naar de kwadratische twist te kijken.

```
sage: q = 64653649714978308587278262886762997122 73
sage: E = EllipticCurve(GF(q), [0,0,1,-38,90])
sage: E.cardinality() == q; E.cardinality()
False
34375776492328245516372636174454222711017688276902037059431689742336519077845
sage: Ed = E.quadratic_twist()
sage: Ed.cardinality() == q; Ed.cardinality()
True
34375776492328245516372636174454222711017688276902037059431689742336519077843
```

Tevens vinden we dat E^d wordt gegeven door de vergelijking

$$y^2 = x^3 + a_4 x + a_6$$
$$a_4 = 153720179180105502453687971416465100143543092134260558634933785983771075716 90$$
$$a_6 = 111010288456678177688281561401559491527006153193473454004497978618449167689 77.$$

Vervolgens kiezen we een punt met zo groot mogelijke orde, om een zo moeilijk mogelijk ECDLP te simuleren. Merk op dat $\#E(\mathbb{F}_q) = q$ met q priem, dus de groep is cyclisch en heeft daardoor een punt met orde q. We verwachten dus één voortbrenger van de groep.

```
sage: Ed.gens()
((2340488223664932436594157492770963047275321503287936546964351063715559508 1583 :
65322739786049076486329255604392424632375073234688587791100301972428870727 65 : 1),)
sage: P = Ed.gens()[0]
sage: k = P.order()
sage: n = ZZ.random_element(2, k-1)
sage: Q = n*P; Q
(2870721267078828114216083125720293233669690972917692955327218944200148749 41 :
1276032267839518151815478209637753670406826803588823219421231855715024179077 : 1)
sage: Smart(P, Q, q, 2)
Het proces duurde 0.243947029114 seconden.
14895177415554804061326795327375540214421499211725966834318965540945696572273
sage: n
14895177415554804061326795327375540214421499211725966834318965540945696572273
```

Zo kunnen we zien dat de Smart methode uiterst effectief is op abnormale krommen.

5

Conclusie en toepassingen in de praktijk

In het verslag hebben we kunnen zien hoe de punten op een elliptische kromme E een abelse groep vormen. De optelling van deze punten is van belang geweest bij het gebruiken van elliptische krommen in de cryptografie. Uiteindelijk hebben we gekeken naar drie voorbeelden van krommen die vermeden moeten worden voor het gebruik van ECC. We vatten de informatie kort samen.

$\#E(\mathbb{F}_q) =$	Methode	Opmerking ter preventie
glad	Pohlig-Hellman	Neem $E(\mathbb{F}_q)$ priem, of minstens een grote priemdeler p van $E(\mathbb{F}_q)$.
$q+1$	MOV	Kies $\mathbb{F}_q$ met kleinste m groot voor $\#E(\mathbb{F}_q) \mid q^m - 1$ of kies andere E
q	Smart	Kies q zodanig dat $a_q \neq 1$

Tabel 5.1: Krommen die vermeden moeten worden

Dit zijn niet de enige typen krommen die vermeden moeten worden, maar krommen met de meest bekende methoden om ECDLP's te kraken zijn genoemd. Voor meer manieren verwijzen we naar [12, 21, 31]. We zullen een lijst geven van voorwaarden waar een elliptische kromme minstens aan moet voldoen, wil deze gebruikt kunnen worden voor de cryptografie: we beginnen met onze waarnemingen uit Tabel 5.1 en vullen deze aan met criteria van *SafeCurves* [6]:

1. $\#E(\mathbb{F}_q)$ heeft minstens één grote priemdeler
2. $E(\mathbb{F}_q)$ heeft een hoge embedding graad m
3. $\#E(\mathbb{F}_q) \neq q$
4. $D \geq 2^{100}$. Hierin is
$$D = \begin{cases} \frac{a_q^2 - 4q}{s^2} & \text{als } \frac{a_q^2 - 4q}{s^2} \equiv 1 \pmod 4 \\ 4\left(\frac{a_q^2 - 4q}{s^2}\right) & \text{elders} \end{cases}$$
en s^2 het grootste kwadraat dat $a_q^2 - 4q$ deelt. Voor kleine waarden van D zou Pollard's ρ efficiënt kunnen werken,
5. De kromme is *robuust*. De keuzes voor de kromme en haar parameters moeten uitgelegd kunnen worden met veiligheidsargumenten.

Daarnaast wordt er ook onderscheid gemaakt tussen ECDLP en ECC veiligheid. ECC is interactief in tegenstelling tot ECDLP. Zo zou een onderschepper bijvoorbeeld ook kunnen bekijken hoeveel tijd

of stroom een berekening die gemaakt wordt door één van de communicerende partijen kost. Dat kan zelfs al genoeg zijn om informatie te geven over de geheime informatie. Er mag dus geen enkele vorm van informatie gelekt worden, wil ECC 'echt' veilig zijn. Dat terwijl je bij ECDLP gewoonweg een wiskundig probleem probeert op te lossen. Daarentegen is veiligheid natuurlijk nooit gegarandeerd. De veiligheid is namelijk gebaseerd op onze huidige kennis, maar ook op onze onkunde. We hebben voorlopig namelijk geen efficiënte algoritmen gevonden om ECDLP's in allerlei situaties snel op te lossen. Zodra een algoritme daarvoor gepubliceerd wordt, zullen de meeste 'veilige' krommen helaas niet meer zo veilig zijn. Dat is op dit moment echter nog niet aan de orde.

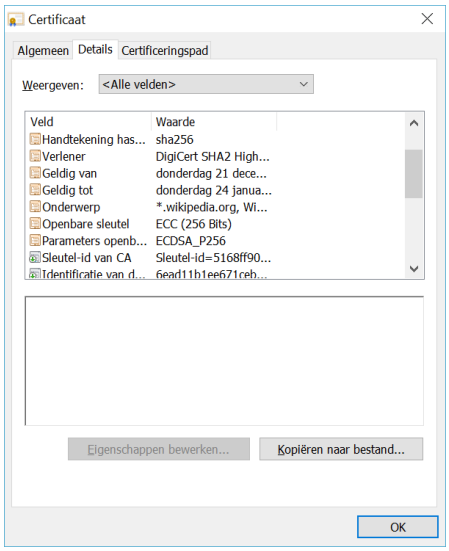
Uiteraard kan men zelf een kromme proberen te maken waarover het ECDLP lastig is op te lossen. In de praktijk is het echter aan te raden om een kromme te kiezen die wordt aangeraden door het NIST, SECG of Brainpool [9, 14, 17]. Voor een overzicht van de veiligheid van krommen raden we aan om te kijken op de website *SafeCurves* [6]. SafeCurves houdt zich bezig met het testen en verifiëren van niet enkel ECDLP veiligheid, maar ook ECC veiligheid. SafeCurves heeft een aantal tests gevoerd op diverse krommen die worden aangeraden voor toepassingen in de praktijk. Ze spreken daar ook over meer methoden om ECDLP's te kraken dan wij hier doen. Ook de kromme Curve25519 is op deze site getest en heeft alles doorstaan. De keuze voor de kromme Curve25519 in ons eerdere voorbeeld was niet met willekeur. Curve25519 wordt namelijk voor een indrukwekkende lijst aan protocollen en software gebruikt en de meeste bekende toepassing waarvan mensen dat waarschijnlijk niet weten, is bij de encryptie van berichten over *WhatsApp*. Deze wordt voornamelijk gebruikt bij het genereren van publieke sleutels. Bijvoorbeeld bij het vormen van een nieuwe groepsgesprek, genereert WhatsApp bij het eerste verstuurde bericht een publieke sleutel met Curve25519. Alle berichten die hierna worden verstuurd, zijn ondertekend met deze publieke sleutel – dit betekent dus dat de andere gebruikers kunnen verifiëren of een bericht gestuurd door gebruiker A daadwerkelijk van A komt. Naast WhatsApp gebruikt ook *Google Chrome* ECC met Curve25519. Chrome hanteert TLS protcollen, die gebruik maken van symmetrische en asymmetrische cryptografie. In het eerste geval kunnen gebruikers nagaan of hij of zij verbonden is met de 'echte' server. Zo wordt gevoelige informatie zoals op sites waar je kan internetbankieren ingevoerd op sites die daar echt een certificaat voor hebben. Wellicht heeft de lezer bij het gebruik van Google Chrome opgemerkt dat er op een aantal sites 'Beveiligd' staat, zoals in Figuur 5.1.

Figuur 5.1: Indicatie van een server met corresponderende certificaat

Door verder te klikken vinden we extra informatie over bijvoorbeeld de publieke en geheime sleutels en hoe deze tot stand zijn gekomen. Zie Figuur 5.2.

We verwijzen naar [2] voor een lijst van toepassingen van de kromme en [32] voor het artikel betreft WhatsApp. Een reden dat Curve25519 zo vaak wordt gebruikt is dat de kromme zeer efficiënt is met het berekenen van n-vouden van een punt P op de kromme, via een Montgomery ladder die deze vermenigvuldiging binnen een vaste tijd kan berekenen. Zo kan de tijd van een vermenigvuldiging geen informatie lekken over de waarde van n.

Ook de eerdergenoemde secp256k1 die voor de Bitcoin wordt gebruikt maakt gebruik van dit soort certificaten. ECDSA is hier erg belangrijk voor de waarborging van transacties. Al met al concluderen we dat er op SafeCurves voldoende krommen staan waar mee gewerkt kan worden. Voor de lezer die toch echt een eigen kromme wil maken, verwijzen we weer naar het werk van Bröker [7].



Figuur 5.2: ECC in een certificaat

Bibliografie

[1] (2017). ECC patents. *Wikipedia*. Geraadpleegd op: 25-06-2018.

[2] (2018). Things that use Curve25519. https://ianix.com/pub/curve25519-deployment.html. Geraadpleegd op: 06-07-2018.

[3] Arora, S. & Barak, B. (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press.

[4] Barreto, P. S. L. M., Galbraith, S. D., hÉigeartaigh, C. O., & Scott, M. (2007). Efficient pairing computation on supersingular Abelian varieties. *Designs, Codes and Cryptography*, *42*(3), 239–271.

[5] Bernstein, D. (2009). Cryptography in DNSCurve. https://dnscurve.org/crypto.html. Geraadpleegd op: 04-07-2018.

[6] Bernstein, D. & Lange, T. (2014). SafeCurves: Choosing safe curves for elliptic-curve cryptography. https://safecurves.cr.yp.to. Geraadpleegd op: 01-07-2018.

[7] Bröker, R. M. (2006). *Constructing Elliptic Curves of Prescribed Order*. PhD thesis, s.n.], S.l.

[8] Bröker, R. M. (2007). Bekijk: De elliptische kromme in je telefoon. https://www.nemokennislink.nl/publicaties/de-elliptische-kromme-in-je-telefoon/. Geraadpleegd: 06-07-2018.

[9] Brown, D. R. (2010). SEC 2: Recommended Elliptic Curve Domain Parameters.

[10] Diffie, W. & Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, *22*(6), 644–654.

[11] Frey, G. & Rück, H.-G. (1994). A Remark Concerning m-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves. *Mathematics of Computation*, *62*(206), 865–874.

[12] Hankerson, D., Menezes, A. J., & Vanstone, S. (2006). *Guide to Elliptic Curve Cryptography*. Springer Science & Business Media.

[13] Howgrave-Graham, N. A. & Smart, N. P. (1999). Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography*, *23*, 283–290.

[14] Information Technology Laboratory (2013). Digital Signature Standard (DSS). Technical Report NIST FIPS 186-4, National Institute of Standards and Technology.

[15] Knapp, A. W. (2018). *Elliptic Curves. (MN-40)*. Princeton University Press.

[16] Koblitz, N. (2012). *A Course in Number Theory and Cryptography*. Springer Science & Business Media.

[17] Lochter, M. & Merkle, J. (2010). Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation. Technical Report RFC5639, RFC Editor.

[18] Mahto, D. & Yadav, D. K. (2017). RSA and ECC: A Comparative Analysis. *12*(19), 9.

[19] Maletsky, K. (2005). RSA vs ECC Comparison for Embedded Systems, 5.

[20] Menezes, A. J., Katz, J., van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. CRC Press.

[21] Menezes, A. J., Okamoto, T., & Vanstone, S. A. (1993). Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, *39*(5), 1639–1646.

[22] Miller, V. S. (1985). Use of Elliptic Curves in Cryptography. In *Advances in Cryptology — CRYPTO '85 Proceedings*, Lecture Notes in Computer Science, (pp. 417–426). Springer, Berlin, Heidelberg.

[23] Nguyen, P. Q. & Shparlinski, I. E. (2002). The Insecurity of the Digital Signature Algorithm with Partially Known Nonces. *Journal of Cryptology*, *15*(3), 151–176.

[24] Novotney, P. (2010). Weak Curves In Elliptic Curve Cryptography, 9.

[25] Pollard, J. M. (1978). Monte Carlo Methods for Index Computation (mod p). *Mathematics of Computation*, *32*(143), 918.

[26] Rivest, R. L., Shamir, A., & Adleman, L. (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, 15.

[27] RSA Labratories (2012). PKCS #1 v2.2: RSA Cryptography Standard, 63.

[28] Shanks, D. (1971). Class number, a theory of factorization, and genera. In *1969 Number Theory Institute (Proc. Sympos. Pure Math., Vol. XX, State Univ. New York, Stony Brook, N.Y., 1969)* (pp. 415–440).

[29] Smart, N. P. (1999). The Discrete Logarithm Problem on Elliptic Curves of Trace One. *Journal of Cryptology*, *12*(3), 193–196.

[30] Stevens, M. & Bursztein, E. (2017). SHAttered. https://shattered.io/. Geraadpleegd: 26-06-2018.

[31] Washington, L. C. (2008). *Elliptic Curves: Number Theory and Cryptography, Second Edition*. CRC Press.

[32] WhatsApp (2017). WhatsApp Encryption Overview. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf. Geraadpleegd op: 03-07-2018.