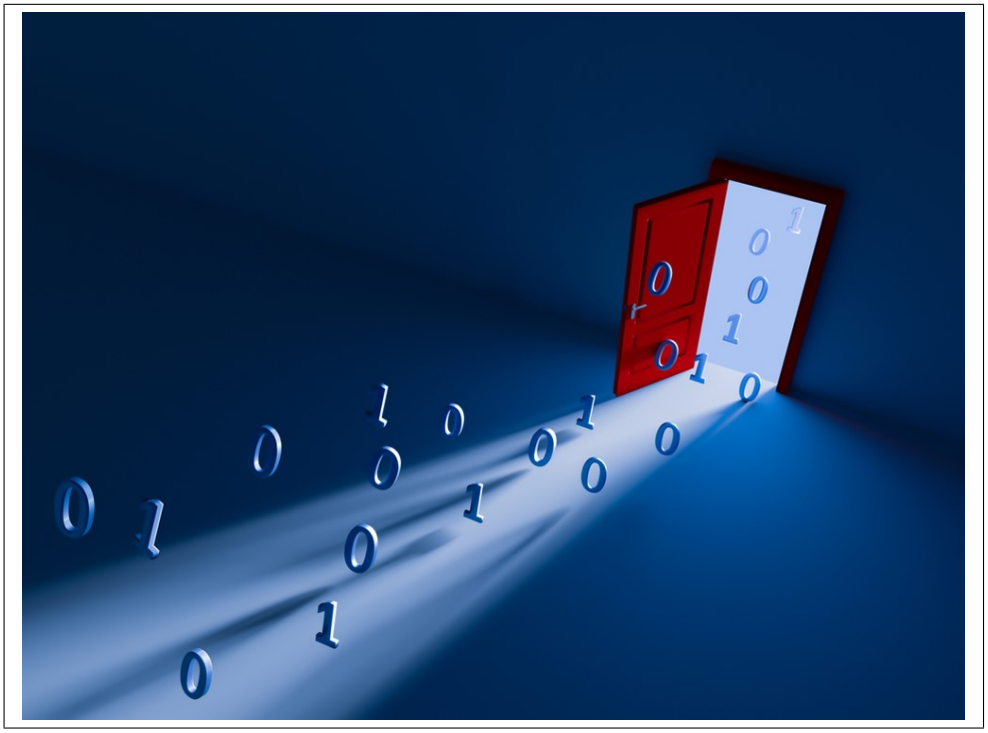
Universiteit Utrecht

Faculteit Bètawetenschappen

Spionage met beveiligingssoftware

Bachelor Scriptie

Paulien van den Noort
Wiskunde



Supervisors:

Dr. G. Tel
Department of Information and Computing Sciences

Prof. Dr. F. Beukers
Mathematisch Instituut

15 juni 2018

Inhoudsopgave

1 Inleiding

"De Rijksoverheid stopt met het gebruik van de antivirussoftware van het Russische bedrijf Kaspersky Lab", meldde nu.nl op 14 mei [1]. Kaspersky Lab is een bekend cybersecurity bedrijf in Rusland dat antivirussoftware maakt. Al eerder besloot de overheid van de Verenigde Staten om de antivirussoftware van Kaspersky Lab niet meer te gebruiken. De reden dat de Verenigde Staten en Nederland stoppen met het gebruik van deze software is de angst voor spionage. Men is bang dat Rusland de antivirussoftware van dit bedrijf gebruikt om te spioneren. Kaspersky Lab is een van de beste bedrijven wat antivirussoftware betreft en heeft altijd gezegd geen banden met het Kremlin te hebben, maar omdat antivirussoftware toegang heeft tot vrijwel elk programma of bestand op de computer, is het geen rare gedachte dat deze software een backdoor zou kunnen bevatten die geheime informatie doorsluist. Patrick Wardle, hoofdonderzoeker bij Digita security, een beveiligingsbedrijf, heeft onderzocht of spionage met de antivirussoftware van Kaspersky Lab mogelijk is. Het blijkt inderdaad mogelijk te zijn om de antivirussoftware zo aan te passen dat spionage mogelijk is. Dit is nog geen bewijs dat Kaspersky Lab meegewerkt heeft aan spionage, maar de mogelijkheid om te spioneren met antivirussoftware bestaat [2].

Antivirussoftware is echter niet de enige software dat bedoeld is voor veiligheid maar gebruikt zou kunnen worden voor spionage. Ook bij het versleutelen van berichten en het zetten van digitale handtekeningen bestaat de mogelijkheid dat er een backdoor ingebouwd is en dat de fabrikant van de software op deze manier persoonlijke informatie van de gebruiker kan aftappen. Denk bijvoorbeeld aan de end-to-end encryptie van WhatsApp. Het idee van end-to-end encryptie is dat elk bericht dat verstuurd wordt eerst versleuteld wordt door de verzendende telefoon en pas weer ontsleuteld wordt door de ontvangende telefoon, dit in tegenstelling tot 'normale' encryptie waar de berichten na verzenden door de berichtenservice worden ontsleuteld en vervolgens weer versleuteld naar de ontvanger worden doorgestuurd. Stel dat iemand het bericht onderschept, dan kan deze persoon het bericht niet lezen. Om het extra veilig te maken, wordt er gebruik gemaakt van een publieke en geheime sleutel per gebruiker. Dit sleutelpaar wordt gemaakt door de telefoon van de gebruiker en alleen de publieke sleutel wordt opgestuurd naar WhatsApp, de geheime sleutel blijft veilig opgeslagen in de telefoon van de gebruiker. Zonder de geheime sleutel kan WhatsApp met de end-to-end encryptie de verstuurde berichten niet lezen. Maar wat nu als WhatsApp het algoritme om de sleutels te maken zo aanpast dat ze met behulp van je publieke sleutel, je geheime sleutel kunnen achterhalen?

We zeggen dat een algoritme waarmee de fabrikant bij gebruik geheime informatie over de gebruiker kan achterhalen, een backdoor bevat, letterlijk: een achterdeurtje. In deze scriptie doe ik onderzoek naar dit soort backdoors. Ik be-

schrijf hoe backdoors eruit kunnen zien in encryptiealgoritmes en in algoritmes om digitale handtekeningen te zetten. Er bestaan verschillende soorten algoritmes hiervoor, ik heb ervoor gekozen om me te richten op de algoritmes die rekenen aan de hand van elliptische krommen. Deze algoritmes hebben een kortere sleutel nodig voor hetzelfde veiligheidniveau als de algoritmes die geen elliptische krommen gebruiken. Een elliptische kromme is gedefinieerd op een lichaam. Hoofdstuk 2 gaat dan ook over lichamen, in dit hoofdstuk wordt verteld wat een lichaam is en hoe men berekeningen kan doen met de elementen uit een lichaam. Hoofdstuk 3 gaat over elliptische krommen, de definitie en de rekenregels voor elliptische krommen worden gegeven en er wordt uitgelegd waar deze regels vandaan komen. Hoofdstuk 4 beschrijft de cryptografische algoritmes voor sleuteluitwisseling, encryptie en digitale handtekeningen. In hoofdstuk 5 beschrijf ik hoe er een backdoor in deze algoritmes geïmplementeerd kan worden. In hoofdstuk 6 beschrijf ik hoe backdoors in de praktijk gebruikt zouden kunnen worden of ooit gebruikt zijn en wat de eventuele gevolgen zouden kunnen zijn. Tenslotte geef ik in hoofdstuk 7 een korte conclusie.

2 Eindige lichamen van orde 2^m

De cryptografische algoritmes die we verderop beschrijven, rekenen aan de hand van elliptische krommen. Een elliptische kromme is gedefinieerd op een lichaam. In dit hoofdstuk wordt allereerst de definitie van een lichaam gegeven en vervolgens uitgelegd hoe we in een lichaam optellen, aftrekken, vermenigvuldigen en delen.

2.1 Eigenschappen van een lichaam

De formele definitie van een lichaam is als volgt:

Definitie 2.1 (Lichaam)**.** *Een lichaam is een verzameling L met twee operaties $+$ en $\cdot$, die voldoet aan de volgende axioma's [3]:*

(L1). ***Associatief*** *Voor alle $a, b, c \in L$ geldt $a + (b + c) = (a + b) + c$ en $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.*

(L2). ***Commutatief*** *Voor alle $a, b \in L$ geldt $a + b = b + a$ en $a \cdot b = b \cdot a$.*

(L3). ***Identiteit*** *Er zijn $0, 1 \in L$ met $0 \neq 1$ zodat $a + 0 = a$ en $a \cdot 1 = a$ voor alle $a \in L$.*

(L4). ***Inverse optelling*** *Voor elke $a \in L$ is er een $-a \in L$ zodat $a + (-a) = 0$.*

(L5). ***Inverse vermenigvuldiging*** *Voor elke $a \in L$ met $a \neq 0$ is er een $a^{-1} \in L$ zodat $a \cdot a^{-1} = 1$.*

(L6). ***Distributief*** *Voor alle $a, b, c \in L$ geldt $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.*

De operatie $+$ wordt optelling en de operatie $\cdot$ wordt vermenigvuldiging genoemd. Uit axioma $(L4)$ volgt dat ook aftrekking $-$ is gedefinieerd in een lichaam, namelijk als $a - b = a + (-b)$ voor alle $a, b \in L$. Het bijzondere aan een lichaam is dat we in een lichaam kunnen delen, dit volgt uit het bestaan van een inverse voor de vermenigvuldiging, zoals gedefinieerd in $(L5)$ en de extra eis dat a niet 0 mag zijn. Deze extra eis is nodig, zodat de situatie waarin door 0 gedeeld moet worden, niet voor kan komen. We noteren deze deeloperatie voor alle $a, b \in L$ als a/b, met $a/b = a \cdot b^{-1}$.

Wanneer we spreken over een *eindig lichaam*, bedoelen we een lichaam met een eindig aantal elementen. Zo'n eindig lichaam heeft een orde.

Definitie 2.2 (Orde van een lichaam)**.** *Het aantal elementen in een eindig lichaam wordt de orde van het lichaam genoemd.*

Elke twee lichamen met dezelfde orde zijn isomorf, daarom mogen we voor elk lichaam met dezelfde orde, dezelfde notatie gebruiken. We noteren een lichaam van orde q dan ook vanaf nu als $\mathbb{F}_q$. De $\mathbb{F}$ komt van *field*, de Engelse vertaling van lichaam. Het meest simpele voorbeeld van een eindig lichaam is $\mathbb{F}_2$, waarin de elementen zich gedragen als booleans. $\mathbb{F}_2$ bevat alleen de elementen 0 en 1, waarbij $0 \neq 1$. De optelling in $\mathbb{F}_2$ werkt precies zo als de *exclusive OR* operatie (*XOR*) uit de informatica en de vermenigvuldiging is gedefinieerd als de *AND* operatie [4].

De orde van een eindig lichaam is altijd van de vorm p^m met p priem en $m \in \mathbb{N}_{>0}$. Deze p wordt de *karakteristiek* en m de *uitbreidingsgraad* van het lichaam genoemd. Voor $m > 1$ heet het lichaam een *uitbreidingslichaam*. Wij zijn vooral geïnteresseerd in eindige lichamen met karakteristiek twee, oftewel: orde 2^m. Berekeningen in binaire lichamen gaan namelijk sneller dan berekeningen in priemlichamen en deze berekeningen kosten minder energie [5].

Omdat elke twee lichamen van gelijke orde isomorf zijn, kunnen we zelf een representatie kiezen van de elementen. De meest gangbare representatie is die van polynomen. Polynomen kunnen we namelijk bij elkaar optellen of van elkaar aftrekken, maar ook met elkaar vermenigvuldigen en al deze operaties voldoen aan $(L1)$ tot en met $(L6)$ van de definitie van een lichaam. Daarom kan in een lichaam $\mathbb{F}_{p^m}$ elk element worden gezien als een polynoom van graad kleiner dan m en coëfficiënten in $\mathbb{F}_p$, het lichaam met orde p. Merk op dat er precies p^m van zulke polynomen bestaan en dat dus elk element van het lichaam op een unieke manier als polynoom geschreven kan worden.

Zodra we polynomen met elkaar gaan vermenigvuldigen, kunnen er polynomen ontstaan van graad m of hoger. Dit wordt voorkomen door modulo een *reductiepolynoom* te rekenen. Dit reductiepolynoom is een irreducibele polynoom van graad m.

Definitie 2.3 (Irreducibele polynoom)**.** *Een irreducibele polynoom is een polynoom die niet geschreven kan worden als product van twee polynomen met lagere graad [6].*

De reductiepolynoom moet irreducibel zijn om nuldelers te voorkomen.

Stelling 2.1. *Een lichaam kan geen nuldelers bevatten.*

Bewijs. We bewijzen deze stelling met een bewijs uit het ongerijmde. Stel het lichaam bevat wel nuldelers, dan bestaan er $x, y \in \mathbb{F}$ ongelijk aan nul, zodat $xy = 0$. Maar als x in $\mathbb{F}$ zit, heeft deze dus volgens $(L5)$ een inverse $x^{-1} \in \mathbb{F}$. Vermenigvuldigen we nu in $xy = 0$ beide kanten met x^{-1}, dan zien we dat $y = 0$. Dit is in tegenspraak met de aanname dat y ongelijk is aan nul. □

Zou de reductiepolynoom reducibel zijn, dan kan deze geschreven worden als product van twee polynomen van lagere graad. Deze twee polynomen zijn dan

beide nuldelers en dan zou er dus niet meer gesproken kunnen worden van een lichaam.

In lichamen met karakteristiek 2 worden de elementen ook wel vaak in de vorm van binaire getallen weergegeven. Zo wordt bijvoorbeeld met de polynoom $x^5 + x^3 + x + 1$ hetzelfde element bedoeld als met het binaire getal 101011. Deze notaties zullen we dan ook beide gebruiken.

2.2 Berekeningen in $\mathbb{F}_{2^m}$

Optellen en aftrekken in $\mathbb{F}_{2^m}$ is relatief eenvoudig, dit gaat namelijk beide precies zo als de *XOR*-operatie. Zo is bijvoorbeeld $110101 + 100010 = 010111$, maar ook $110101 - 100010 = 010111$. Met polynomen gaat dit op dezelfde manier, we tellen ze op zoals we normaal gesproken polynomen optellen, maar zodra we een 2 als coëfficiënt krijgen, wordt deze gelijk aan 0. Bovenstaand voorbeeld ziet er dan met polynomen als volgt uit: $(x^5 + x^4 + x^2 + 1) + (x^5 + x) = x^4 + x^2 + x + 1$.

Vermenigvuldigen in $\mathbb{F}_{2^m}$ gaat, zoals eerder genoemd, met behulp van een reductiepolynoom. Allereerst worden de polynomen met elkaar vermenigvuldigd en daarna worden alle coëfficiënten modulo 2 genomen. Deze polynoom wordt vervolgens gedeeld door de reductiepolynoom met behulp van een staartdeling. In de restterm die overblijft na de staartdeling, worden alle coëfficiënten weer modulo 2 genomen en dan is deze restterm is het gewenste antwoord. In het geval van binaire getallen gaat de vermenigvuldiging zoals men zou verwachten, waarbij rekening gehouden moet worden met de volgende vier regels: $0 * 0 = 0$, $1 * 0 = 0$, $0 * 1 = 0$ en $1 * 1 = 1$. Vervolgens wordt er ook hier een staartdeling toegepast, waarbij we in plaats van aftrekken de *XOR*-operatie $\oplus$ gebruiken. Ook in dit geval is de restterm het antwoord [4]. We geven hieronder een voorbeeld waarbij we 1010011 willen vermenigvuldigen met 10111010 en waarbij de reductiepolynoom 100011011 is. Links is de vermenigvuldiging te zien en rechts de reductie met behulp van een staartdeling. We zien dus dat $(1010011 \cdot 10111010) \bmod 100011011 = 11001111$.

```
      10111010
       1010011 ·
-----------------
      10111010
     10111010
    10111010
   10111010      ⊕
-----------------
10010011101110
```

```
10010011101110
100011011      ⊕
-----------------
 11110001110
  100011011    ⊕
-----------------
  1111100010
   100011011   ⊕
-----------------
   111010100
    100011011  ⊕
-----------------
    11001111
```

Omdat delen is gedefinieerd als $a/b = a \cdot b^{-1}$, moeten we de inverse voor de vermenigvuldiging van b vinden en dan kunnen we daarna de vermenigvuldiging, zoals bescheven, toepassen op a en b^{-1}. Het vinden van de inverse voor vermenigvuldiging kan op verschillende manieren. De meest efficiënte methode is het uitgebreide algoritme van Euclides. Stel dat we de inverse van $b \in \mathbb{F}_{2^m}$ willen bepalen en f is de reductiepolynoom. Het doel van het algoritme is een $c \in \mathbb{F}_{2^m}$ en een $e \in \mathbb{F}_{2^m}$ vinden zodat $cb + ef = 1$, want dan is $c \equiv b^{-1} \bmod f(x)$. We weten dat $1 \cdot b + 0 \cdot f = b$ en dat $0 \cdot b + 1 \cdot f = f$. Het uitgebreide algoritme van Euclides begint dan ook met de twee vergelijkingen $cb + ef = u$ en $db + gf = v$. In de initialisatie wordt c gelijk aan 1, d gelijk aan 0, u gelijk aan b en v gelijk aan f. Vervolgens kijken we telkens welke van de polynomen u en v de hoogste graad heeft en vermenigvuldigen we de polynoom met de laagste graad een aantal keer met x totdat de polynomen gelijke graad hebben trekken deze vervolgens van de polynoom met de hoogste graad af. Merk op dat dit eigenlijk een partiële deling is van u op v of andersom. Op deze manier wordt de graad van de polynoom met de hoogste graad in elke stap minstens met 1 verlaagd. We stoppen wanneer de graad van u gelijk is aan 0. Omdat f irreducibel is, betekent dit dat $u = 1$ en dus dat c de inverse is van b. Het uitgebreide algoritme van Euclides berekent de inverse van b dus als volgt [7, blz. 8]:

1. $c := 1, d := 0, u := b, v := f$.
2. Zolang de graad van de polynoom u ongelijk is aan 0, doe de volgende drie stappen:
   2.1. $j :=$ graad$(u) -$ graad(v).
   2.2. Als $j < 0$: Verwissel u en v, verwissel c en d en $j := -j$.
   2.3. $u := u + x^j v$, $c := c + x^j d$.
3. Return c.

3 Wat zijn elliptische krommen?

In de 18^e eeuw werd onderzoek gedaan naar een methode om de booglengte van een ellips te bepalen. Uiteindelijk was het Euler die hier een formule voor vond. Andere wiskundigen deden verder onderzoek naar deze en soortgelijke formules om de achtergrond beter te kunnen begrijpen. Na verloop van tijd ging men de inversen van deze functies bekijken en deze definiëren over bepaalde ruimten en zo ontstonden de elliptische krommen zoals we die kennen in de wiskunde. Deze elliptische krommen bleken veel toepassingen in de wiskunde te hebben, het gebruik van deze krommen in de cryptografie is er hier een van [8, 9]. We willen nu eerst de formele definitie van een elliptische kromme geven:

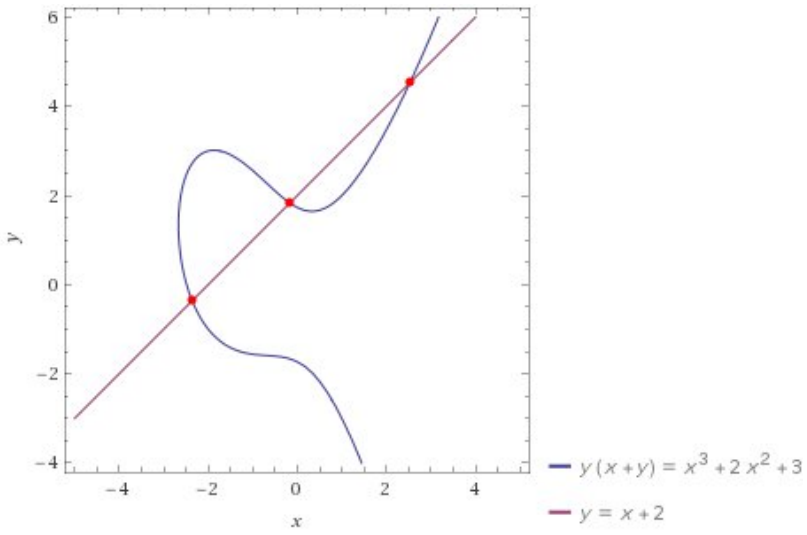
Definitie 3.1 (Elliptische kromme over $\mathbb{F}_{2^m}$)**.** *Een elliptische kromme is de verzameling van punten $P(x, y)$, met $x, y \in \mathbb{F}_{2^m}$, die voldoen aan de vergelijking:*

$$E : y^2 + xy = x^3 + ax^2 + b, \tag{1}$$

samen met een punt op oneindig: ∞. Hierin zijn $a, b \in \mathbb{F}_{2^m}$ en $b \neq 0$ [10].

Een elliptische kromme kunnen we opvatten als een groep onder optelling, waarbij het punt ∞ het identiteitselement is. De optelling bestaat uit een aantal regels die gedefinieerd zijn volgens een geometrische interpretatie. Wanneer men de verzameling punten $P(x, y)$ plot over $\mathbb{R}$, vormen deze namelijk een kromme als in figuur 1. In $\mathbb{F}_{2^m}$ vormen de punten niet zo'n mooie kromme, maar zien deze er meer uit als een wolk van punten die niet verbonden zijn. De geometrische interpretatie zoals we die voor de kromme in $\mathbb{R}$ gebruiken, kunnen we dan ook niet zomaar doortrekken naar $E(\mathbb{F}_{2^m})$, maar de formules die hieruit afgeleid worden, kunnen we wel gebruiken [11, par. 3.2]. Om te weten waar deze formules vandaan komen, leggen we nu dan ook eerst de geometrische interpretatie van de optelling uit.

De optelling van twee verschillende punten P en Q gaat als volgt: we trekken een lijn door de twee punten en het derde snijpunt dat deze lijn met de kromme heeft noemen we R, vervolgens definiëren we de som van P en Q als de inverse van R. De drie snijpunten van een lijn met de elliptische kromme hebben dus som ∞, het identiteitselement. Om een punt P te verdubbelen, wordt gebruik gemaakt van een raaklijn aan de kromme in P. Deze raaklijn snijdt de kromme in een ander punt, de inverse van dit punt is $2P$. Om de inverse van een punt te bepalen, wordt een verticale lijn door dit punt getrokken. Nu heeft zo'n lijn, op één uitzondering na, altijd twee snijpunten met de kromme, deze twee punten zijn elkaars inverse.



Figuur 1: Een elliptische kromme met $a = 2$ en $b = 3$ over $\mathbb{R}$, doorsneden met een lijn

We willen nu met behulp van deze geometrische interpretatie laten zien hoe de regel voor het bepalen van de inverse van een willekeurig punt (s, r) is afgeleid. Allereerst trekken we een denkbeeldige verticale lijn door $(s, r) \in E(\mathbb{F}_{2^m})$. We noemen het tweede snijpunt van deze lijn met de elliptische kromme (s, t) en vullen beide punten in in de elliptische kromme:

$$r^2 + sr = s^3 + as^2 + b$$
$$t^2 + st = s^3 + as^2 + b.$$

Vervolgens trekken we deze vergelijkingen van elkaar af:

$$r^2 + sr - t^2 - st = 0.$$

We weten al dat $t = r$ een oplossing is, dus we halen $t - r$ buiten haakjes:

$$(t - r)(-t - r - s) = 0.$$

De andere oplossing is dus $t = -r - s$. Merk op dat $r, s \in \mathbb{F}_{2^m}$ en dat optellen en aftrekken in $\mathbb{F}_{2^m}$ hetzelfde zijn, dus $-r - s = r + s$. Volgens de commutatieve eigenschap van het lichaam geldt dat $r + s = s + r$. Dus de inverse van (s, r) is $(s, s + r)$ [12].

De regel voor het optellen van twee verschillende punten en de regel voor het verdubbelen van een punt zijn op een zelfde soort manier afgeleid. Al de regels voor de optelling zijn als volgt [10, 13]:

- $\infty + \infty = \infty$
- $(x, y) + \infty = \infty + (x, y) = (x, y)$ voor alle $(x, y) \in E(\mathbb{F}_{2^m})$
- De inverse van $(x, y) \in E(\mathbb{F}_{2^m})$ is $(x, x + y) \in E(\mathbb{F}_{2^m})$ en de inverse van ∞ is het punt ∞ zelf.
- Voor $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ geldt $2P = P + P = (x_3, y_3)$ met $x_3 = \lambda^2 + \lambda + a$ en $y_3 = x_1^2 + \lambda x_3 + x_3$, waarbij $\lambda = \frac{x_1 + y_1}{x_1}$.
- Voor $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ en $Q = (x_2, y_2) \in E(\mathbb{F}_{2^m})$ met $P \neq Q$ geldt dat $P + Q = (x_3, y_3)$ met $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$ en $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$, waarbij $\lambda = \frac{y_2 + y_1}{x_2 + x_1}$.

Merk op dat een punt $(x, y) \in E(\mathbb{F}_{2^m})$ een vector is met x en y in $\mathbb{F}_{2^m}$. De operaties met de coördinaten gebeuren dan ook in $\mathbb{F}_{2^m}$ zoals beschreven staat in 2.2.

Het is niet moeilijk om aan te tonen dat de elliptische kromme $E(\mathbb{F}_{2^m})$ met bovenstaande regels voor optelling inderdaad een groep is, zoals bijvoorbeeld te zien is in het boek *The arithmetic of elliptic curves* [14].

4 Hoe wordt dit gebruikt in de cryptografie?

4.1 Cryptografische algoritmen met elliptische krommen

Elliptische krommen worden op verschillende manieren toegepast in de cryptografie. In deze paragraaf beschrijven we de meest bekende: ECDH voor sleuteluitwisseling, ECIES voor encryptie en ECDSA voor digitale handtekeningen.

4.1.1 Domeinparameters

Om een cryptografisch algoritme te kunnen gebruiken, moeten de verzendende en de ontvangende partij van te voren de zogeheten domeinparameters afspreken, zoals welke kromme er gebruikt wordt en in welk lichaam de berekeneningen gedaan worden. Voor cryptografie met elliptische krommen over $\mathbb{F}_{2^m}$ worden de volgende 7 domeinparameters gebruikt: m, $f(x)$, a, b, G, n en h [15]. De m is de m uit $\mathbb{F}_{2^m}$, dit moet een positief geheel getal zijn. $f(x)$ is de reductiepolynoom, deze heeft graad m en is irreducibel. a en b zijn dezelfde a en b als in definitie 3.1, deze bepalen dus welke kromme er gebruikt wordt. G is de generator en n is de orde van G. De *generator* is een punt op de elliptische kromme en de *orde* van G is het aantal keren dat G bij zichzelf opgeteld moet worden om het identiteitselement ∞ te krijgen. h is de *cofactor*. De cofactor wordt berekend door het aantal punten in $E(\mathbb{F}_{2^m})$ te delen door n [16]. In paragraaf 4.2 staat beschreven hoe al deze parameters worden gekozen.

4.1.2 Discrete logaritme

De cryptografie die gebruik maakt van elliptische krommen (Elliptic Curve Cryptography/ECC) is gebaseerd op de moeilijkheid om de discrete logaritme te berekenen, ook wel het discrete logaritme probleem genoemd (ECDLP). Dit probleem heeft te maken met scalaire vermenigvuldiging in elliptische krommen. ECDLP is kortgezegd het volgende: gegeven de punten P en Q op de elliptische kromme, vind de k waarvoor $Q = kP$ [11, par. 5.2]. We weten dat $2P = P + P$. Wanneer we $3P$ definiëren als $3P = 2P + P = (P + P) + P$, kunnen we met behulp van inductie kP definiëren door P k keer bij zichzelf op te tellen. Wanneer k en P bekend zijn, zijn er verschillende algoritmen waarmee we Q relatief snel kunnen berekenen, maar in het geval van ECDLP zijn Q en P bekend en moeten we k juist berekenen. Als de orde n groot is, zijn er veel mogelijkheden voor k. Voor n groot genoeg (minstens 160 bits) zijn er nog geen snelle algoritmen gevonden om de discrete logaritme te kunnen berekenen. De meest snelle algoritmen om ECDLP op te lossen hebben complexiteit $\mathcal{O}(\sqrt{n})$ [17].

4.1.3 ECDH

ECDH (Elliptic Curve Diffie-Hellman) is een asymmetrisch cryptosysteem waarmee een sleutel gegenereerd wordt. Deze sleutel kan vervolgens gebruikt worden in een symmetrisch encryptie algoritme. Om een sleutel te genereren met ECDH nemen Alice en Bob beide een willekeurig geheel getal d met $0 < d < n$, dit is de geheime sleutel. We noemen de geheime sleutel van Alice d_A en de geheime sleutel van Bob d_B. De publieke sleutels van Alice en Bob worden nu als volgt berekend: $K_A = d_A G$ en $K_B = d_B G$. Alice stuurt vervolgens haar publieke sleutel K_A op naar Bob en Bob stuurt zijn publieke sleutel K_B op naar Alice. Alice berekent $S_A = d_A K_B$ en Bob berekent $S_B = d_B K_A$. Merk op dat $S_A = d_A K_B = d_A d_B G = d_B d_A G = d_B K_A = S_B$. Dus Alice en Bob hebben nu hetzelfde punt $S := S_A = S_B$ op de elliptische kromme. De geheime sleutel van Alice en Bob is nu de x-coördinaat x_S van S. Oscar kan nu alleen K_A en K_B te weten komen, maar om hieruit S te berekenen, heeft hij d_A of d_B nodig en om d_A of d_B te berekenen zou hij een discrete logaritme moeten oplossen [15, 18].

4.1.4 ECIES

ECIES (Elliptic Curve Integrated Encryption Scheme) is een vorm van *hybride* cryptografie, wat betekent dat er een asymmetrisch algoritme wordt gebruikt om een sleutel te genereren en dat vervolgens met een symmetrisch algoritme het bericht versleuteld wordt met behulp van de gegenereerde sleutel. In het geval van ECIES is het asymmetrische cryptosysteem waarmee de sleutel gegenereerd wordt ECDH. Stel dat Alice met behulp van ECIES een bericht naar Bob wil sturen. Alice en Bob berekenen eerst de geheime sleutel x_S met ECDH. Met deze sleutel kan in principe het bericht gelijk versleuteld worden met behulp van een symmetisch algoritme, maar soms wordt er voor extra veiligheid nog gebruik gemaakt van een label waarmee Bob kan controleren of het bericht inderdaad van Alice komt. Bob en Alice spreken dan van te voren nog twee extra variabelen s_1 en s_2 af. Alice berekent met behulp van x_S en s_1 en een KDF (Key Derivation Function) de sleutels k_E en k_M. Om het bericht te versleutelen wordt nu k_E gebruikt en met behulp van k_M en een MAC-functie (Message Authentication Code) wordt de label van het versleutelde bericht en van s_2 berekend. Vervolgens wordt het versleutelde bericht c samen met de label naar Bob gestuurd. Bob weet x_S en s_1 en kan dus op dezelfde manier als Alice k_E en k_M berekenen. Vervolgens berekent Bob de label met de MAC-functie en controleert of deze hetzelfde is als de label die Alice gestuurd heeft. Als de labels inderdaad hetzelfde zijn, kan Bob het bericht c ontsleutelen volgens het symmetrische algoritme met behulp van de sleutel k_E [10, 19].

4.1.5 ECDSA

Als Alice een bericht naar Bob stuurt, is het belangrijk voor Bob om te weten dat het bericht wat hij ontvangt ook echt van Alice afkomstig is. Daarom ondertekent Alice haar bericht met een digitale handtekening, een stukje extra informatie waaraan Bob kan zien dat het bericht inderdaad van Alice is. Deze handtekening kan bijvoorbeeld berekend worden met behulp van ECDSA. ECDSA (Elliptic Curve Digital Signature Algorithm) is een algoritme voor het maken van digitale handtekeningen dat gebruik maakt van elliptische krommen. Om een handtekening te maken met ECDSA berekent Alice eerst $e = HASH(m)$, ze hasht dus haar bericht. e moet bitlengte n hebben, dus de hash-functie moet zo gekozen worden dat de output van deze functie altijd bitlengte n heeft. Vervolgens wordt er een willekeurig getal k met $0 < k < n$ genomen en wordt deze k vermenigvuldigd met de generator G. Van het punt kG neemt Alice de x-coördinaat modulo n en noemt het getal wat zo ontstaat r. Er moet gelden dat $r \neq 0$, wanneer dit niet het geval is, neemt Alice een nieuwe k en berekent daarmee opnieuw r. Vervolgens wordt $s = k^{-1}(e + d_A r) \bmod n$ berekend, waarbij d_A de geheime sleutel van Alice is. Ook voor s geldt dat s ongelijk aan 0 moet zijn, anders moet er een nieuw willekeurig getal k genomen worden en moeten r en s opnieuw berekend worden voor deze k. Zodra Alice een paar r en s ongelijk aan 0 heeft gevonden kan ze (r, s) als handtekening met haar bericht meesturen.

Voor het berekenen van s hebben we de inverse van k modulo n nodig. Om met zekerheid te kunnen zeggen dat k ook daadwerkelijk een inverse heeft, moet n een priemgetal zijn.

Bob ontvangt nu het bericht m met de handtekening (r, s). Hij kijkt eerst of $0 < r, s < n$, want alleen dan is de handtekening geldig. Met behulp van de publieke sleutel K_A van Alice kan hij vervolgens controleren of het bericht inderdaad van Alice afkomstig is. Bob berekent hiertoe eerst $e = HASH(m)$ en daarna $u_1 = s^{-1}e \bmod n$ en $u_2 = s^{-1}r \bmod n$. Met deze u_1 en u_2 bepaalt hij het punt $P = u_1 G + u_2 K_A$. Alleen als de x-coördinaat p_x van P modulo n gelijk is aan r, is het bericht van Alice.

Dat Bob op deze manier inderdaad kan controleren dat het bericht van Alice afkomstig is, is in te zien door de publieke sleutel $K_A = d_A G$ van Alice te substitueren in $P = u_1 G + u_2 K_A$. We zien dan dat $P = u_1 G + u_2 d_A G = (u_1 + u_2 d_A)G$. Gebruiken we vervolgens dat $u_1 = s^{-1}e \bmod n$ en $u_2 = s^{-1}r \bmod n$, dan zien we dat $P = (s^{-1}e + s^{-1}r d_A)G = s^{-1}(e + r d_A)G$. In deze vergelijking hoeven we niet meer mee te nemen dat u_1 en u_2 modulo n berekend worden, want we vermenigvuldigen met G en n is de orde van G, er zal dus dezelfde uitkomst voor P uitkomen. Nu is s^{-1} gedefinieerd als $s^{-1} = (k^{-1}(e + d_A r))^{-1} = k(e + d_A r)^{-1} \bmod n$. We kunnen P dus schrijven als: $P = k(e + d_A r)^{-1}(e + r d_A)G = kG \bmod n$. Alice had van $kG \bmod n$ de x-coördinaat r genoemd, dus de x-coördinaat van het

punt P van Bob is inderdaad gelijk aan r als de handtekening afkomstig is van Alice [16, 20, 21].

4.2 Hoe de juiste domeinparameters worden gekozen

Bij het kiezen van de juiste parameters is het van belang om eerst te bedenken hoe veilig het systeem moet zijn. Een veiligheidsniveau van i bits betekent dat de aanvaller maximaal 2^i wiskundige operaties moet uitvoeren om de geheime sleutel te achterhalen. Om een veiligheidsniveau i te bereiken moet de sleutellengte $2i$ zijn. Een veiligheidsniveau van 80 bits was veilig genoeg tot 2010. Doordat er elk jaar betere computers komen, gaat dit veiligheidsniveau elk jaar omhoog. Tot het jaar 2030 zou een veiligheidsniveau van 112 bits genoeg moeten zijn [22, blz. 73] [21]. Omdat de sleutels punten op de elliptische kromme $E(\mathbb{F}_{2^m})$ zijn, bepaalt de keuze van m de lengte van de sleutel en dus ook de veiligheid van het systeem. We willen m groot genoeg om een veilig systeem te hebben, maar niet te groot om de berekeningen snel te houden. Tot 2010 was $m = 160$ genoeg, nu zal m een iets hoger getal moeten zijn. SECG (Standards for Efficient Cryptography Group) [22, blz. 5] raadt aan om een $m \in \{163, 233, 239, 283, 409, 571\}$ te kiezen. Bij het selecteren van deze waarden voor m is met verschillende dingen rekening gehouden, zo is het belangrijk dat m een priemgetal is, anders zijn de krommen kwetsbaar voor bepaalde aanvallen en is er gekeken naar de orde, oftewel het aantal elementen, van bepaalde elliptische krommen over het lichaam $\mathbb{F}_{2^m}$ [17].

Verder geeft SECG bij elke m een geschikte reductiepolynoom. Voor $m = 163$ is dit bijvoorbeeld $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ en voor $m = 233$ is dit $f(x) = x^{233} + x^{74} + 1$ [22, blz. 5]. De laatste term van de reductiepolynoom moet altijd 1 zijn, anders kunnen we x buiten haakjes halen en is de polynoom niet irreducibel. Verder moet de polynoom een oneven aantal termen hebben, anders kan de polynoom ontbonden worden in factoren met één van de factoren gelijk aan $(x + 1)$ [10]. Nu is het belangrijk dat er zo weinig mogelijk termen zijn, zodat de binaire representatie van het polynoom veel nullen bevat, dit versimpelt de berekeningen en maakt het algoritme dus sneller. De gebruikte reductiepolynomen hebben daarom ook eigenlijk altijd 3 of 5 termen. Verder moet er altijd minstens één term zijn met een oneven macht, anders kan het polynoom geschreven worden als kwadraat van twee polynomen met graad $\frac{m}{2}$ [10]. Het is voor de efficiëntie van de berekeningen van belang om de machten in de reductiepolynoom zo laag mogelijk te houden [10]. De macht van de eerste term moet gelijk zijn aan m, de macht van de tweede term wordt dan zo laag mogelijk gekozen, gegeven de eerste term. Voor een reductiepolynoom met vijf termen wordt de macht van de derde term zo laag mogelijk gekozen, gegeven de eerste twee termen en de macht van de vierde term zo laag mogelijk, gegeven de eerste drie termen. De laatste term is altijd 1 [22, blz. 6]. In het geval van een reductiepolynoom met drie termen kan

bewezen worden dat de macht van de tweede term op deze manier altijd kleiner is dan $\frac{m}{2}$ [10].

Wat betreft het kiezen van de generator G is het van belang om te kijken naar de orde n van G en de cofactor h. We weten uit 4.1.5 dat n een priemgetal moet zijn, verder moet n groot genoeg zijn om te voorkomen dat de discrete logaritme, zoals beschreven in 4.1.2, te makkelijk uitgerekend kan worden. De generator G genereert een subgroep van $E(\mathbb{F}_{2^m})$ met n elementen door middel van scalaire vermenigvuldigingen. Volgens de stelling van Lagrange is n een deler van het aantal elementen in $E(\mathbb{F}_{2^m})$. Hieruit volgt dat de cofactor $h = \frac{\#E(\mathbb{F}_{2^m})}{n}$ een geheel getal is. Om n zo groot mogelijk te krijgen, moet h zo klein mogelijk zijn. In het beste geval is h gelijk aan 1 [17]. Krommen waarvoor geldt $\#E(\mathbb{F}_{2^m}) = 2^m$ kunnen beter niet gebruikt worden, omdat $\#E(\mathbb{F}_{2^m})$ dan geen grote priemfactor bevat.

Voor een efficiënte implementatie kunnen de parameters a en b gekozen worden als $b = 1$ en $a \in \{0, 1\}$, dit soort krommen heten Koblitz krommen. Voor extra veiligheid zijn willekeurige a en b soms beter. SECG [23] geeft een aantal voorbeelden van relatief veilige elliptische krommen met bijbehorende parameters.

5 Wat zijn backdoors en hoe ziet de implementatie eruit?

Wanneer een cryptografisch algoritme een *backdoor* bevat, wordt hiermee bedoeld dat het algoritme iets aangepast is, waardoor de aanvaller extra informatie heeft. Een backdoor geeft niet direct de geheime sleutel of het geheime bericht van de gebruiker aan de aanvaller, maar met de extra informatie, zijn eigen geheime sleutel en de output van het algoritme heeft de aanvaller bijvoorbeeld de mogelijkheid om de geheime sleutel van de gebruiker bij een volgend gebruik te achterhalen [24]. Een *kleptografische backdoor* is hetzelfde als een asymmetrische backdoor, oftewel: een backdoor in een asymmetrisch algoritme [25]. Adam Young en Moti Yung hebben in 1996 de basis gelegd voor de kleptografie. *Kleptografie* is het onderzoeksgebied waarin men zich bezig houdt met het veilig (gezien vanuit het oogpunt van de aanvaller) stelen van informatie met behulp van cryptografie [26]. Young en Yung hebben onder andere gedefinieerd waar een goede backdoor, door hen een SETUP genoemd, aan moet voldoen. We geven deze definitie, zoals beschreven in hun artikel *The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems* [27], hieronder en leggen deze vervolgens uit.

Definitie 5.1 (SETUP (Secretly Embedded Trapdoor with Universal Protection))**.** *Neem aan dat C een black-box cryptosysteem is (de precieze implementatie is niet bekend) met algemeen bekende specificaties. Een SETUP meganisme is een aanpassing van het algoritme C naar C' zo dat:*

1. *De input van C' overeen komt met de publieke specificaties voor de input van C.*

2. *C', die de publieke encryptie functie E van de aanvaller (en mogelijk ook andere functies) bevat, efficiënt rekent.*

3. *De geheime decryptie functie D van de aanvaller niet bevat is in C' en alleen bekend is bij de aanvaller.*

4. *De output van C' overeen komt met de publieke specificaties voor de output van C. Tegelijkertijd bevat de output gepubliceerde bits (van de geheime sleutel van de gebruiker) die makkelijk af te leiden zijn door de aanvaller (de output kan gegenereerd worden tijdens de sleutel-generatie of tijdens systeemoperaties zoals het zenden van berichten).*

5. *Verder zijn de output van C en C' in polynomiale tijd niet te onderscheiden voor iedereen behalve de aanvaller.*

6. *Na het ontdekken van de specifieke details van het setup-algoritme en van zijn aanwezigheid in de implementatie, kunnen de gebruikers (behalve de aanvaller) geen eerdere (of mogelijk volgende) sleutels achterhalen.*

Stel dat Alice een bericht naar Bob stuurt en dat bericht ondertekent met ECDSA. Ze gebruikt zonder het te weten een versie van ECDSA met een door de uitgever van het algoritme ingebouwde SETUP. Ze kan niet precies zien hoe het algoritme werkt, maar eventuele input en output en de rekensnelheid van het algoritme kan Alice wel zien. De aanvaller (uitgever van het algoritme) wil natuurlijk niet dat Alice kan zien dat het algoritme een SETUP bevat, daarom moet dit algoritme met dezelfde input werken als een ECDSA algoritme zonder SETUP. Dit komt overeen met de eerste eis van de bovenstaande definitie. De output van het algoritme met SETUP is wel anders dan de output van een algoritme zonder SETUP. Uit de output moet de aanvaller namelijk bepaalde gegevens van Alice kunnen afleiden. Wel moet de output van ECDSA met een SETUP een digitale handtekening (r, s) zijn, waaruit Bob op dezelfde manier als bij een handtekening die gemaakt is met een versie van ECDSA zonder SETUP kan afleiden dat het bericht inderdaad van Alice afkomstig is. Zou dit niet op dezelfde manier gaan, dan hebben Alice en Bob namelijk al snel door dat het algoritme een SETUP bevat. Deze eis staat beschreven in punt 4 van definitie 5.1. Zoals in punt 5 beschreven staat, moet het voor iedereen behalve de aanvaller niet gemakkelijk zijn om te zien dat de output anders is dan de output van een normaal ECDSA algoritme. Verder is het, zoals punt 2 zegt, van belang dat het algoritme ondanks de aanpassingen nog wel efficiënt de berekeningen uitvoert. Ook is het het beste dat de decryptiefunctie van de aanvaller alleen bekend is bij de aanvaller en dat deze dus niet verwerkt zit in het algoritme, zodat niemand anders dan alleen de aanvaller zelf de extra informatie kan afleiden, dit komt overeen met punt 3 van de definitie. Tenslotte, stel dat Alice en Bob toch ontdekken dat er een SETUP in het ECDSA algoritme zit en stel dat ze ontdekken hoe de SETUP precies werkt, dan wil de aanvaller dat Alice en Bob de extra informatie alsnog niet kunnen ontcijferen. Dit staat beschreven in punt 6 van definitie 5.1.

Een algoritme met een SETUP moet over het algemeen meerdere keren gebruikt worden voordat de aanvaller de gewenste informatie kan afleiden. In de algoritmes die beschreven staan in de paragrafen 5.1, 5.2 en 5.3 kan de aanvaller bij de tweede keer dat Alice het algoritme gebruikt haar geheime sleutel afleiden. Voor al deze algoritmes geldt dat de aanvaller Alice's sleutel (of $l - 1$ sleutels in het geval van ECDH) kan achterhalen en dus ook $l - 1$ berichten van haar kan lezen, wanneer zij het algoritme l keer gebruikt heeft.

Omdat alle cryptografische algoritmes die we beschrijven gebaseerd zijn op het discrete logaritme probleem, beschrijven we in paragraaf 5.1 eerst een mogelijke SETUP waarmee de discrete logaritme zonder al te veel moeite door de aanval-

ler berekend kan worden. Vervolgens beschrijven we in paragraaf 5.2 en 5.3 hoe deze SETUP toegepast kan worden bij een sleuteluitwisseling met ECDH en bij een digitale handtekening met ECDSA. Tenslotte beschrijven we in paragraaf 5.4 waarom bepaalde stappen nodig zijn in de SETUP algoritmes en hoe de bijbehorende constanten gekozen moeten worden.

5.1 Discrete log SETUP

Het algoritme wat we in deze paragraaf beschrijven [28, blz. 3], berekent voor een willekeurige k het punt $Q = kG$ op de elliptische kromme. De aanvaller wil de discrete logaritme van Q, oftewel het getal k weten. De gebruiker van het algoritme moet het algoritme twee keer gebruiken voordat de aanvaller k kan afleiden. In ronde 2 wordt k_2 afhankelijk gekozen van k_1 uit de eerste ronde, waardoor er stiekem een verband wordt aangebracht tussen Q_1 en Q_2. De aanvaller weet dit verband en kan vervolgens, met zijn geheime sleutel en de kennis van Q_1 en Q_2, achterhalen wat k_2 is. We laten hieronder zien hoe het algoritme werkt in de eerste twee keer dat het gebruikt wordt en hoe de aanvaller vervolgens k kan bepalen. ω, α en β zijn constante gehele getallen, met ω oneven. $H()$ is een hashfunctie die als output waarden kleiner dan n geeft. x is de geheime sleutel van de aanvaller en $Y = xG$ de publieke sleutel van de aanvaller. Merk op dat ronde 1 nog gewoon hetzelfde gaat als het algoritme zonder SETUP, behalve dat k_1 wordt opgeslagen op de computer van de gebruiker van het algoritme.
Ronde 1:

1. Neem willekeurige $k_1 < n$.
2. Sla k_1 op in het permanente geheugen van de computer.
3. Bereken $Q_1 = k_1 G$.
4. Output Q_1.

Ronde 2:

1. (a) Kies $t \in \{0, 1\}$ willekeurig.
   (b) Bereken $Z = (k_1 - \omega t)G + (-\alpha k_1 - \beta)Y$.
   (c) Bereken $k_2 = H(Z)$.
2. Sla k_2 op in het permanente geheugen van de computer.
3. Bereken $Q_2 = k_2 G$.
4. output Q_2.

Aanvaller:

1. Bereken $R = \alpha Q_1 + \beta G$.
2. (a) Bereken $Z_1 = Q_1 - xR$.
   (b) Als $Q_2 = H(Z_1)G$, output $k_2 = H(Z_1)$.
3. (a) Bereken $Z_2 = Z_1 - \omega G$.
   (b) Als $Q_2 = H(Z_2)G$, output $k_2 = H(Z_2)$.

Om te laten zien dat de aanvaller op deze manier inderdaad k_2 vindt, willen we aantonen dat Z_1 of Z_2 gelijk is aan Z uit ronde 2. We substitueren hiervoor $Q_1 = k_1 G$ en $R = \alpha Q_1 + \beta G$ in de formule voor Z_1 en vereenvoudigen deze:

$$\begin{aligned} Z_1 &= Q_1 - xR \\ &= k_1 G - x(\alpha Q_1 + \beta G) \\ &= k_1 G - x\alpha Q_1 - x\beta G. \end{aligned}$$

Vervolgens gebruiken we dat $Q_1 = k_1 G$ en dat $Y = xG$:

$$\begin{aligned} Z_1 &= k_1 G - x\alpha Q_1 - x\beta G \\ &= k_1 G - x\alpha k_1 G - x\beta G \\ &= k_1 G - \alpha k_1 xG - \beta xG \\ &= k_1 G - \alpha k_1 Y - \beta Y \\ &= k_1 G + (-\alpha k_1 - \beta)Y. \end{aligned}$$

Merk op dat deze formule voor Z_1 gelijk aan Z is, als in de eerste stap van ronde twee $t = 0$ gekozen is. Als $t = 1$ gekozen is, dan hebben we de formule voor Z_2 nodig:

$$\begin{aligned} Z_2 &= Z_1 - \omega G \\ &= k_1 G + (-\alpha k_1 - \beta)Y - \omega G \\ &= k_1 G - \omega G + (-\alpha k_1 - \beta)Y \\ &= (k_1 - \omega)G + (-\alpha k_1 - \beta)Y. \end{aligned}$$

We zien dat Z_2 inderdaad gelijk is aan Z voor $t = 1$.

5.2 ECDH SETUP

Voor een SETUP algoritme van ECDH kan in feite gewoon het SETUP algoritme voor het berekenen van de discrete logaritme gebruikt worden. Bij ECDH berekent Alice namelijk eerst haar publieke sleutel door haar geheime sleutel, een willekeurig geheel getal kleiner dan n te vermenigvuldigen met de generator G. Dit is precies wat het SETUP algoritme beschreven in paragraaf 5.1 doet. Vervolgens ontvangt Alice Bob's publieke sleutel en berekent de gedeelde geheime sleutel S door haar eigen geheime sleutel te vermenigvuldigen met Bob's publieke sleutel. Als we dus het SETUP algoritme voor de discrete logaritme samenvoegen met de berekening van S door Alice's geheime sleutel te vermenigvuldigen met Bob's publieke sleutel, hebben we een SETUP algoritme voor ECDH. De aanvaller kent Alice's en Bob's publieke sleutels en kan uit Alice's publieke sleutel haar geheime sleutel afleiden en deze vervolgens met Bob's publieke sleutel vermenigvuldigen om S te bepalen. Merk op dat deze SETUP pas werkt als Alice minstens 2 keer een nieuwe geheime en publieke sleutel opvraagt.

5.3 ECDSA SETUP

Het ECDSA algoritme met een SETUP is iets ingewikkelder dan het ECDH algoritme met een SETUP. Het handtekeningenschema ECDSA begint met het kiezen van een willekeurige k. Het doel van de SETUP is om deze k te achterhalen, met behulp van k kan namelijk de geheime sleutel d_A berekend worden. Ook nu kan weer de discrete log SETUP gebruikt worden [28, blz. 5]. We beschrijven het SETUP algoritme hieronder stap voor stap. ω, α en β zijn ook hier weer constante gehele getallen, met ω oneven. $H()$ is een hashfunctie met output kleiner dan n. x is de geheime sleutel van de aanvaller en $Y = xG$ is de publieke sleutel van de aanvaller. Ook nu gaat ronde 1, afgezien van het opslaan van k_1, hetzelfde als een ECDSA algoritme zonder SETUP.
Ronde 1:

1. Neem willekeurige $k_1 < n$.
2. Sla k_1 op in het permanente geheugen van de computer.
3. Bereken $R = (b_1, b_2) = k_1 G$.
4. Bereken $r_1 = b_1 \bmod n$.
5. Bereken $s_1 = k_1^{-1}(HASH(m) + d_A r_1) \bmod n$.
6. Output $\sigma_1 = (r_1, s_1)$.

Ronde 2:

1. (a) Neem $t \in \{0, 1\}$ willekeurig.
   (b) Bereken $Z = (k_1 - \omega t)G + (-\alpha k_1 - \beta)Y$.
   (c) Bereken $k_2 = H(Z)$.
2. Sla k_2 op in het permanente geheugen van de computer.
3. Bereken $R = (c_1, c_2) = k_2 G$.
4. Bereken $r_2 = c_1 \bmod n$.
5. Bereken $s_2 = k_2^{-1}(HASH(m) + d_A r_2) \bmod n$.
6. Output $\sigma_2 = (r_2, s_2)$.

De aanvaller heeft eigenlijk R uit ronde 1 nodig in plaats van r_1 om hetzelfde ontsleutelingsalgoritme te kunnen gebruiken als in de discrete log SETUP. We gebruiken daarom dat er op een elliptische kromme altijd precies twee punten zijn met dezelfde x-coördinaat. Allereerst vult de aanvaller dus r_1 in $E(\mathbb{F}_{2^m})$ in en vindt daarmee de twee punten R_1 en R_2 met x-coördinaat r_1. Vervolgens voert hij onderstaand algoritme uit met R_1 en als hij daarmee k_2 niet gevonden heeft, voert hij dit algoritme uit met R_2 [29].
Aanvaller:

1. Bereken $W = \alpha R_1 + \beta G$.
2. (a) Bereken $Z_1 = R_1 - xW$.
   (b) Als r_2 gelijk is aan de x-coördinaat van $H(Z_1)G \bmod n$, output $k_2 = H(Z_1)$.
3. (a) Bereken $Z_2 = Z_1 - \omega G$.
   (b) Als r_2 gelijk is aan de x-coördinaat van $H(Z_2)G \bmod n$, output $k_2 = H(Z_2)$.

Met k_2 kan de aanvaller nu de geheime sleutel d_A van Alice als volgt berekenen: $d_A = r_2^{-1}(k_2 s_2 - HASH(m))$ [28]. Bob kan nog gewoon op dezelfde manier als bij een handtekening met een ECDSA algoritme zonder SETUP controleren of het bericht van Alice afkomstig is.

5.4 De stappen in een SETUP algoritme en de keuze van de constanten

We zien dat in de beschreven algoritmes de SETUP in stap 1 a, b en c van ronde 2 zit. In deze paragraaf wil ik het idee achter deze stappen uitleggen en beschrijven waar rekening mee moet worden gehouden in het kiezen van de constanten. Een goede SETUP voldoet aan de 6 punten uit definitie 5.1. Voor de beschreven algoritmes met SETUP is het vooral belangrijk om ervoor te zorgen dat deze voldoen aan punt 5 en 6. Dat de algoritmes voldoen aan de andere 4 punten is niet zo moeilijk om in te zien. Voor punt 5 is het van belang dat k_2 even willekeurig lijkt als k_1, we willen dus dat k_2 zo veel mogelijk waarden tussen 0 en n aan kan nemen. Als $H()$ een hashfunctie is die alle waarden tussen 0 en n aan kan nemen, hangt het dus van Z af hoeveel van deze waarden de hashfunctie ook daadwerkelijk bereikt. Allereerst schrijven we Z wat anders:

$$\begin{aligned} Z &= (k_1 - \omega t)G + (-\alpha k_1 - \beta)Y \\ &= (k_1 - \omega t)G + (-\alpha k_1 - \beta)xG \\ &= k_1(1 - \alpha x)G + (-\omega t - \beta x)G. \end{aligned}$$

We definiëren $G_1 := (-\omega - \beta x)G$, $G_2 := (-\beta x)G$ en $G_3 := (1 - \alpha x)G$. We kunnen Z nu schrijven als $Z = k_1 G_3 + G_i$ met $i \in \{1, 2\}$ afhankelijk van de waarde van t. Om Z zo veel mogelijk verschillende waarden te laten bereiken, is het belangrijk om de constanten α, β en ω zo te kiezen dat G_1, G_2 en G_3 een zo hoog mogelijke orde hebben, het liefst orde n. We weten dat k_1 geheel willekeurig gekozen is en alle waarden tussen 0 en n aan kan nemen, dus als G_3 orde n zou hebben, bereikt Z alle mogelijke punten op de elliptische kromme en kan k_2 alle mogelijke waarden tussen 0 en n bereiken.

Dan rest nu nog de vraag waar t en ω voor nodig zijn. We weten dat t alleen maar de waarden 0 en 1 aan kan nemen, t is dan ook enkel bedoeld om ω 'aan' of 'uit' te zetten. Het doel van ω is het voorkomen dat de SETUP wordt ontdekt in het geval dat $H()$ per ongeluk inverteerbaar blijkt te zijn. In het artikel *Using Cryptography Against Cryptography* [30, blz. 68-69] van Yung en Young staat dit uitgebreider beschreven.

6 Backdoors in de praktijk

In dit hoofdstuk wil ik enkele voorbeelden geven van situaties waarin een backdoor voor problemen zou kunnen zorgen of waarin dit juist handig zou kunnen zijn. WhatsApp, Skype, Facebook en vele andere berichtendiensten maken gebruik van end-to-end encryptie, dit betekent dat de berichten die verstuurd worden, versleuteld worden op het apparaat van de verzender en pas weer ontsleuteld worden op het apparaat van de ontvanger, dit in tegenstelling tot 'normale' encryptie waar de berichten na verzenden door de berichtenservice worden ontsleuteld en vervolgens weer versleuteld naar de ontvanger worden doorgestuurd. Bij end-to-end encryptie kan de berichtenservice de berichten van de gebruikers dus niet lezen. Voor het genereren van een sleutel waarmee de berichten versleuteld worden, wordt een sleuteluitwisselingsprotocol gebruikt zoals bijvoorbeeld ECDH [31]. Elke gebruiker heeft een geheime en een publieke sleutel, die gemaakt zijn door zijn eigen apparaat. De geheime sleutel wordt opgeslagen op zijn eigen telefoon of computer en de publieke sleutel wordt verstuurd naar de berichtendienst. Ik neem nu als voorbeeld de berichtendienst WhatsApp. Merk op dat dit slechts een voorbeeld is, er is geen enkele aanleiding om te twijfelen aan de betrouwbaarheid van WhatsApp. Stel dat Alice nu een bericht naar Bob wil sturen via WhatsApp, dan stuurt de WhatsApp-server de publieke sleutel van Alice naar Bob en de publieke sleutel van Bob naar Alice op. Vervolgens berekenen de telefoons van Alice en Bob de gedeelde geheime sleutel door de geheime sleutel die opgeslagen is op de telefoon te vermenigvuldigen met de ontvangen publieke sleutel. Met de gedeelde geheime sleutel kunnen vervolgens de berichten versleuteld en weer ontsleuteld worden. WhatsApp maakt gebruik van het Signal-protocol, wat betekent dat er telkens een tijdelijke sleutel wordt gebruikt voor de encryptie van de berichten [32]. Dit is in principe heel veilig, want stel dat een aanvaller achter de geheime sleutel komt, dan kan deze geen oude berichten lezen, omdat die met een andere sleutel versleuteld zijn. Maar stel dat WhatsApp in de versie van ECDH die gebruikt wordt een backdoor heeft ingebouwd, dan kan WhatsApp al bij de tweede keer dat er een sleutel wordt aangemaakt, de geheime sleutels van de gebruikers berekenen. Met behulp van deze sleutels kan WhatsApp ook alle komende sleutels berekenen en vervolgens dus alle volgende berichten van deze gebruikers ontsleutelen en meelezen. Ik wil hier nogmaals opmerken dat ik WhatsApp slechts gebruik als voorbeeld, er is geen enkele aanwijzing dat WhatsApp ook daadwerkelijk gebruik maakt van backdoors.

Ook als het gaat om officiële documenten kan een backdoor voor problemen zorgen. Als bijvoorbeeld de overheid een officieel document verstuurt, dan wordt dat document ondertekend met een digitale handtekening, bijvoorbeeld gemaakt met ECDSA, zodat degene die het ontvangt zeker weet dat het bericht inderdaad van de overheid afkomstig is. Als iemand dat document wil vervalsen, kan diegene het document niet ondertekenen omdat hij de geheime sleutel van de overheid niet

weet. Zou deze persoon echter een deal sluiten met de softwarefabrikant van het programma waar de overheid de digitale handtekeningen mee zet en de softwarefabrikant bouwt een backdoor in in de software, dan kan de geheime sleutel berekend worden en kan met deze sleutel het vervalste document alsnog ondertekend worden.

Een backdoor kan ook binnen de financiële wereld voor problemen zorgen. Er zijn banken die bij internetbankieren de transacties versleutelen en ondertekenen. Als men in dit geval de geheime sleutel weet te achterhalen, kunnen er, terwijl de juiste transactie op het scherm getoond wordt, vervalste transacties op de achtergrond ondertekend en verstuurd worden [33].

Backdoors hoeven niet altijd het werk van criminelen te zijn. Rond dezelfde tijd dat WhatsApp met de end-to-end encryptie kwam, was er een discussie tussen het bedrijf Apple en de FBI gaande. De FBI wilde toegang tot de iPhone van een terrorist, maar de encryptie van Apple was te sterk. De FBI wilde dat dit niet nog eens voor zou komen en vroeg Apple om een oplossing. De oplossing zou er op neer komen dat Apple een backdoor zou inbouwen in de iPhone. Apple weigerde dit echter. Apple was bang dat het niet alleen bij deze zaak zou blijven en dat de backdoor de privacy van de gebruikers in gevaar zou brengen [34].

Een ander voorbeeld van een situatie waarin er een backdoor gebruikt is, is de DUAL_EC_DRBG, gepubliceerd door de NSA. DUAL_EC_DRBG is een algoritme dat willekeurige getallen genereert. De gegenereerde getallen worden vervolgens gebruikt als input voor cryptografische algoritmes. Deze getalgenerator werd door veel bekende organisaties als standaard aangeraden. Als DUAL_EC_DRBG dus inderdaad een backdoor zou bevatten, zou de NSA heel veel cryptografische algoritmes kunnen omzeilen. Vlak voor de eerste publicatie was al ontdekt dat de DUAL_EC_DRBG mogelijk een backdoor zou bevatten, maar deze backdoor had de eigenschap dat het onmogelijk was voor iemand anders dan de NSA zelf om het daadwerkelijke bestaan van deze backdoor te bevestigen. In 2013 werd, volgens The New York Times, door uitlekken van geheime informatie bekend dat de DUAL_EC_DRBG inderdaad een backdoor bevat en dat de NSA aan RSA Security, een Amerikaans security bedrijf, 10 miljoen dollar betaald had om de DUAL_EC_DRBG als standaard te gebruiken [35]. Het bleek dat de NSA een geheim project onder de naam Bullrun heeft. Het doel van dit project is het ontcijferen van versleutelde berichten. Ook de Britten hebben een soortgelijk project. Door onder andere deze projecten zijn er waarschijnlijk heel wat meer backdoors aanwezig in verschillende cryptografische algoritmes dan dat er tot nu toe bekend zijn [36]. Deze backdoors zullen ook niet zo snel bekend worden, want een backdoor die aan de 6 eisen uit definitie 5.1 voldoet, is heel moeilijk op te sporen. Voor zover ik weet, is naar het opsporen van backdoors ook nog maar weinig onderzoek gedaan.

7 Conclusie

In de software die we elke dag gebruiken, zijn naar alle waarschijnlijkheid vaker backdoors aanwezig dan dat we van te voren zouden denken. Dit vooral omdat bepaalde veiligheidsorganisaties zoals de NSA en de FBI graag inzicht willen hebben in de dingen die mensen online doen, zoals bijvoorbeeld de berichten die ze versturen. Die backdoors kunnen ten koste gaan van de privacy van mensen, maar soms ook juist weer handig zijn in bijvoorbeeld onderzoek naar criminelen. Helaas maken niet alleen veiligheidsorganisaties gebruik van backdoors, ook criminelen zelf kunnen software met een backdoor op de markt brengen. Ik heb onderzocht hoe backdoors in algoritmes die gebaseerd zijn op het discrete logaritme probleem, precies in elkaar zitten. Hierbij heb ik ervoor gekozen om me te richten op die algoritmes die rekenen aan de hand van elliptische krommen. Ik heb laten zien wat elliptische krommen precies zijn en hoe cryptografie met elliptische krommen werkt. Uiteindelijk heb ik specifieke voorbeelden van een implementatie van een backdoor in het sleuteluitwisselingsprotocol ECDH en in het handtekeningenschema ECDSA gegeven. Deze implementaties zijn gemakkelijk door te trekken naar andere cryptografische algoritmes die gebaseerd zijn op het discrete logaritme probleem.

Referenties

[1] NU.nl, *Nederlandse overheid stopt met russische antivirus-software kaspersky*, https://www.nu.nl/internet/5265584/nederlandse-overheid-stopt-met-russische-antivirussoftware-kaspersky.html (2018), geraadpleegd op 29-5-2018.

[2] N. Perlroth, *How antivirus software can be turned into a tool for spying*, https://www.nytimes.com/2018/01/01/technology/kaspersky-lab-antivirus.html (2018), geraadpleegd op 29-5-2018.

[3] *Wikipedia, field (mathematics)*, https://en.wikipedia.org/wiki/Field_(mathematics) (z.d.), geraadpleegd op 30-4-2018.

[4] *Wikipedia, finite field arithmetic*, https://en.wikipedia.org/wiki/Finite_field_arithmetic (z.d.), geraadpleegd op 30-4-2018.

[5] E. Wenger and M. Hutter, in *Information Security Technology for Applications*, edited by P. Laud, NordSec 2011 (Springer, Berlin, Heidelberg, 2012), vol. 7161 of *Lecture Notes in Computer Science*, pp. 256–257.

[6] *Wikipedia, irreducible polynomial*, https://en.wikipedia.org/wiki/Irreducible_polynomial (z.d.), geraadpleegd op 1-5-2018.

[7] D. Hankerson, J. López-Hernandez, and A. Menezes, in *Cryptographic Hardware and Embedded Systems*, edited by C. K. Koç and C. Paar, CHES 2000 (Springer, Berlin, Heidelberg, 2000), vol. 1965 of *Lecture Notes in Computer Science*, pp. 1–24.

[8] P. Joshi, *Why are they called elliptic curves?*, Blog op Perpetual Enigma, https://prateekvjoshi.com/2015/02/07/why-are-they-called-elliptic-curves/ (2015), geraadpleegd op 1-5-2018.

[9] D. Ravenel, *Elliptic curves: what they are, why they are called elliptic, and why topologists like them, i*, Wayne State University Mathematics Colloquium, https://web.math.rochester.edu/people/faculty/doug/mypapers/wayne1.pdf (2007), geraadpleegd op 1-5-2018.

[10] K. Rabah, Information Technology Journal **5**, 204 (2006), URL https://scialert.net/fulltext/?doi=itj.2006.204.229#e10.

[11] Certicom, *Ecc tutorial*, https://www.certicom.com/content/certicom/en/ecc-tutorial.html (2016), geraadpleegd op 1-5-2018.

[12] S. I. Serengil, *The math behind elliptic curves over binary field*, https://sefiks.com/2016/03/13/the-math-behind-elliptic-curves-over-binary-field/ (2016), geraadpleegd op 1-5-2018.

[13] D. R. Susantio and I. Muchtadi-Alamsyah, Journal of Physics: Conference Series **710**, 012022 (2016), URL http://stacks.iop.org/1742-6596/710/i=1/a=012022.

[14] J. H. Silverman, *The arithmetic of elliptic curves* (Springer, Dordrecht, Heidelberg, 2009), par. 3.2: The group law.

[15] M. S. Anoop, *Elliptic curve cryptography - an implementation guide*, http://www.infosecwriters.com/Papers/Anoopms_ECC.pdf (2015).

[16] J. Bauer, *Ecc tutorial*, https://www.johannes-bauer.com/compsci/ecc/?menuid=4 (z.d.), geraadpleegd op 1-5-2018.

[17] *Wikipedia, elliptic-curve cryptography*, https://en.wikipedia.org/wiki/Elliptic-curve_cryptography (z.d.), geraadpleegd op 1-5-2018.

[18] *Wikipedia, elliptic-curve diffie-hellman*, https://en.wikipedia.org/wiki/Elliptic-curve_Diffie-Hellman (z.d.), geraadpleegd op 1-5-2018.

[19] *Wikipedia, integrated encryption scheme*, https://en.wikipedia.org/wiki/Integrated_Encryption_Scheme (z.d.), geraadpleegd op 1-5-2018.

[20] A. Corbellini, *Elliptic curve cryptography: Ecdh and ecdsa*, http://andrea.corbellini.name/2015/05/30/elliptic-curve-cryptography-ecdh-and-ecdsa/ (2015).

[21] *Wikipedia, elliptic curve digital signature algorithm*, https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm (z.d.), geraadpleegd op 1-5-2018.

[22] D. R. L. Brown, *SEC1: Elliptic Curve Cryptography*, Standards for Efficient Cryptography Group, 2nd ed. (2009), certicom Research.

[23] D. R. L. Brown, *SEC2: Recommended Elliptic Curve Domain Parameters*, Standards for Efficient Cryptography Group, 2nd ed. (2010), certicom Research.

[24] C. Easttom, Journal of Information System Security **3**, 177 (2018), URL https://www.academia.edu/14152138/Cryptographic_Backdoors.

[25] *Wikipedia, kleptography*, https://en.wikipedia.org/wiki/Kleptography (z.d.), geraadpleegd op 3-5-2018.

[26] *Wikipedia, moti yung*, https://en.wikipedia.org/wiki/Moti_Yung (z.d.), geraadpleegd op 3-5-2018.

[27] A. Young and M. Yung, Advances in Cryptology Crypto '97 pp. 264–276 (1997), URL https://link-springer-com.proxy.library.uu.nl/content/pdf/10.1007%2FBFb0052241.pdf.

[28] S. Verbücheln, CoRR (2015), 1501.00447, URL http://arxiv.org/abs/1501.00447.

[29] E. Mohamed and H. Elkamchouchi, IJCSNS **10**, 264 (2010).

[30] A. Young and M. Yung, Advances in Cryptology Crypto '97 pp. 62–74 (1997), URL http://cryptome.org/2013/09/klepto-crypto.pdf.

[31] WhatsApp, *Whatsapp encryption overview, technical whitepaper*, https://www.bof.nl/wp-content/uploads/WhatsApp-Security-Whitepaper.pdf (2016).

[32] S. van Voorst, *De nieuwe beveiliging van whatsapp: De stap naar end-to-end encryptie*, https://tweakers.net/reviews/4515/de-nieuwe-beveiliging-van-whatsapp.html (2016), geraadpleegd op 28-5-2018.

[33] I. Koskosas, Business Excellence and Management **1**, 49 (2011).

[34] D. Parrack, *Apple refuses to help the fbi*, https://www.makeuseof.com/tag/apple-refuses-to-help-the-fbi-popcorn-time-returns-tech-news-digest/ (2016), geraadpleegd op 28-5-2018.

[35] *Wikipedia, dual_ec_drbg*, https://en.wikipedia.org/wiki/Dual_EC_DRBG (z.d.), geraadpleegd op 28-5-2018.

[36] *Wikipedia, bullrun (programma)*, https://nl.wikipedia.org/wiki/Bullrun_(programma) (z.d.), geraadpleegd op 28-5-2018.