Vermogen- en tijdsanalyse van cryptosystemen gebaseerd op elliptische krommen

Nele Mentens, Pieter Rommens en Marian Verhelst
3^{de} Ir. micro-elektronica

16 mei 2003

Dankwoord

Een eindwerk maak je niet alleen. Maar we waren dan ook met drie. Toch stelden we vast dat dit soms nog te weinig was. Berna wist ons met haar dagelijks goed humeur ook op moeilijke momenten weer te motiveren. Haar praktische kennis en grenzeloze inzet, zelfs lang na zonsondergang, maakten van haar de gedroomde begeleidster. De wekenlange minutieuze verbeteringen van Karel gaven deze tekst vorm en inhoud. Professor Preneel bedanken we voor het interessante onderwerp en de hartelijke sfeer in zijn onderzoeksgroep.

Het afsluiten van onze studies in Leuven doet ons terugblikken op de tijd die aan dit eindwerk vooraf ging. Reeds bijna een kwart eeuw lang geven onze ouders ons een thuis, waar we steeds tot rust kunnen komen. Hun steun en liefdevolle raad maakten ons tot de personen die we nu zijn. En wat zouden we doen zonder onze vrienden, bij wie we telkens onze batterijen weer konden opladen? Jullie zijn niet weg te denken uit ons leven. Bijzonder blijven Mimi, Bart en Thomas, in wiens armen we de nodige afleiding vinden. Samen gaan we de toekomst in.

Pieter, Nele en Marian
mei 2003

Samenvatting

Door middel van vermogen- en tijdsmetingen op een cryptosysteem is het mogelijk te achterhalen wat er zich in de chip afspeelt. De juiste interpretatie van de metingen maakt de weg vrij om de geheime cryptografische sleutel op te sporen en zo verstuurde boodschappen af te luisteren of te manipuleren.

De vakliteratuur gaat uitgebreid in op de resultaten hiervan voor ASIC-implementaties, maar van onderzoek op FPGA-implementaties van cryptosystemen was nog nooit sprake. Dit eindwerk onderzoekt dan ook voor het eerst de veiligheid van een cryptosysteem geïmplementeerd op FPGA. Uit dit werk blijkt bovendien dat de resultaten van metingen op een FPGA overeenkomen met die van een ASIC. Dit maakt van deze herprogrammeerbare hardware het ideale middel voor het maken van prototypes van cryptochips en voor het testen van de veiligheid ervan.
Het aan te vallen systeem is in dit geval gebaseerd op elliptische kromme cryptografie. Deze vorm van publieke sleutel cryptografie steunt volledig op de vermenigvuldiging van een gekend punt op een zogenaamde elliptische kromme met een geheime scalar, de te achterhalen sleutel.
In het kader van dit eindwerk werd een VHDL-implementatie ontwikkeld van een elliptische kromme puntvermenigvuldiging. Dit zowel voor elliptische krommen gebaseerd op velden met even als met oneven karakteristiek. Met behulp van tijdsanalyse blijkt dat in deze eerste naïeve implementatie van het systeem, het Hamming-gewicht van de sleutel zomaar voor het oprapen ligt. Gebruik makend van eenvoudige vermogenanalyse kan zelfs de gehele sleutel gewoon afgelezen worden. Het is echter niet moeilijk alternatieve implementaties te vinden die wel resistent zijn tegen deze vorm van aanvallen.
Deze verbeterde implementaties zijn daarmee nog steeds niet veilig tegen aanvallers. Dit eindwerk toont aan dat met behulp van de veel complexere differentiële vermogenanalyse het nog steeds mogelijk is de sleutel te achterhalen. Dit is meteen een primeur, vermits in de literatuur nog nooit melding is gemaakt van een uitvoering van differentiële vermogenanalyse op een elliptische kromme cryptosysteem. Dit eindwerk rondt af met verschillende technieken om de chip ook beter resistent te maken tegen deze vermogenaanval.

Inhoudsopgave

Lijst van symbolen

$GF(q)$	Galois veld met q elementen.
$\text{char}(GF(q))$	Karakteristiek van het eindig veld $GF(q)$.
$< G, + >$	Groep voor de bewerking + over de verzameling G.
$GF(p)$	Eindig veld met priemgetal p als karakteristiek.
$GF(2^n)$	Eindig veld met karakteristiek 2.
$GF(2)[x]$	Verzameling van alle veeltermen $F(x) = \Sigma_{i=0}^{k} a_i x^i$ met $a_i \in GF(2)$ en $k \in \mathbb{N}$.
$GF(2)[x]/(p(x))$	Verzameling van elementen uit $GF(2)[x]$ modulo de veelterm $p(x)$.
$\overline{\mathbb{F}}$	Algebraïsche sluiting van het veld $\mathbb{F}$.
$\mathcal{O}$	Punt op oneindig.
$E/\mathbb{F}$	Elliptische kromme E over het veld $\mathbb{F}$.
$E(\mathbb{F})$	Verzameling van alle punten op de kromme $E/\mathbb{F}$.
$(a_{n-1}, \ldots, a_1, a_0)_b$	Vectornotatie voor $a_{n-1} * b^{n-1} + \ldots + a_1 * b + a_0 * 1$ met basis b (ontbreken van b betekent basis gelijk aan het nulpunt van de irreduceerbare veelterm).
$ggd(a, b)$	Grootste gemene deler van a en b.
r^{-1}	Inverse van r voor de vermenigvuldiging.
$x \bmod p$	x modulo p of de rest van x na deling door p.
$Mont(x, y)$	Montgomery vermenigvuldiging van x en y.
$(x, y) \rightarrow (a, b)$	Coördinatentransformatie van (x, y) naar (a, b).
$[k]P$	Puntvermenigvuldiging van het punt P met de scalar k.
$\otimes$	Logische exclusieve OR bewerking (XOR).
$*$	Logische AND bewerking of aritmetische vermenigvuldiging.
$+$	Logische OR bewerking of aritmetische optelling.
$\overline{p}$	Logische inverse van bit p (NOT).
$\bullet$	Brent Kung operator.
$x << i$	i plaatsen naar links schuiven van de bitstring x.

Hoofdstuk 1

Inleiding

1.1 Wat is cryptografie?

Geheimen zijn zo oud als de mens zelf. Gaande van hoogst persoonlijke brieven tot strategieën van nationaal belang, de mens was steeds zeer vindingrijk als het er op aan kwam enkel de juiste persoon zijn boodschap te laten lezen. Julius Caesar liet zijn geheime documenten versleutelen door iedere letter te vervangen door de letter drie plaatsen verder in het alfabet. Een A werd zo een D, een B een E enz. Op het eerste zicht een eenvoudige en doeltreffende methode. Maar wat als een derde persoon de versleutelde tekst wat langer onderzoekt en snugger opmerkt dat de H (die staat voor een E) en de D (die staat voor een A) opvallend vaker voorkomen? Dan is de stap naar het zoeken naar de statistieken van de letters van het alfabet snel gezet. Een klinker zal in een Latijnse tekst, zoals ook in een Nederlandse tekst, statistisch vaker voorkomen dan de afzonderlijke medeklinkers. Eens de sleutel (schuifmechanisme over drie letters) gevonden is, wordt de encryptie waardeloos. In de loop der eeuwen is de mens dan ook verder op zoek gegaan naar steeds vernuftigere systemen om zijn informatie te beschermen.

De ontwikkeling en massaverspreiding van computers en andere elektronische communicatiesystemen vanaf de jaren '60, creëerde een plotse behoefte aan veiligheidsmaatregelen voor het beschermen van digitale informatie. De IC-revolutie verlegde de grenzen van het kopiëren en aanpassen van informatie drastisch. Men kan moeiteloos enkele duizenden kopieën van elektronisch opgeslagen data maken, waarbij het onmogelijk wordt het origineel te onderscheiden van een kopie. De bescherming van data en geheimhouding ervan tijdens de overdracht over onveilige kanalen is het doel van de *cryptografie*.

Hoewel het kennisgebied van de cryptografie een enorme evolutie heeft doorgemaakt door de exponentiële groei van de rekenkracht van computers, wordt de cryptograaf toch voortdurend opnieuw uitgedaagd door de even snelle ontwikkeling van aanvalsmethoden. De *cryptanalyse* zoekt de zwakheden van een gegeven versleutel-algoritme of de realisatie ervan in hardware. De crytograaf moet dus moeiteloos als verdediger in de huid van de aanvaller kunnen kruipen en omgekeerd. Van zodra een systeem 'gebroken' is, geeft dit hem juist een extra stimulans om een verbeterd algoritme te ontwikkelen dat dan in de volgende versie van het systeem wordt ingebouwd.

1.2 Basisbegrippen

De voorbije decennia is er een brede waaier van cryptografische algoritmes ontstaan. De cryptografie maakt voor de beschrijving van de eigenschappen van deze algoritmes gebruik van de volgende terminologie:

Klaartekst De eigenlijke informatie die men veilig wil verzenden, de boodschap voordat zij versleuteld is. Zij kan bestaan uit een binaire code, Nederlandse tekst, computer code, ...

Versleutelde tekst De uitgang na encryptie ofwel de versleutelde klaartekst. De versleutelde tekst wordt via het kanaal fysisch verzonden tussen twee entiteiten.

Zender De entiteit of persoon in een communicatie tussen twee partijen die de legitieme bron van informatie vormt. In Figuur 1.1 is Alice de zender.

Ontvanger De entiteit of persoon in een communicatie tussen twee partijen die het legitieme doel is van de informatiestroom. In Figuur 1.1 is Bob de ontvanger.

Aanvaller De entiteit of persoon in een communicatie tussen twee partijen die noch de zender noch de ontvanger is en die tracht de klaartekst te achterhalen of te veranderen. In Figuur 1.1 is de aanvaller Eve.

Enkele van de belangrijkste doelstellingen van de cryptografie zijn:

Data Integriteit De garantie dat de informatie niet ongeoorloofd veranderd of vernietigd werd. Onder data manipulatie verstaan we het invoegen, verwijderen of vervangen van data.

Vertrouwelijkheid De zekerheid dat de inhoud van de boodschap enkel en alleen in geoorloofde handen komt.

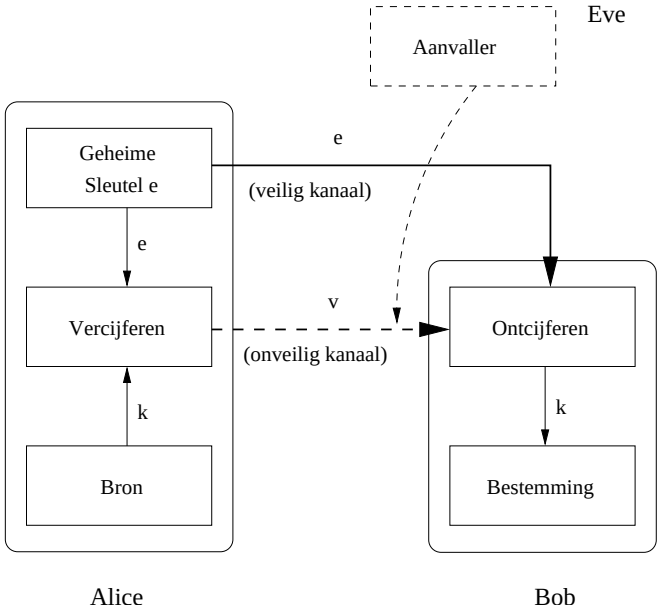
Authentisering Men maakt een onderscheid tussen authentisering van entiteiten en van de oorsprong van data. Men spreekt van identificatie of authentisering van entiteiten wanneer men wordt overtuigd van de identiteit van een andere partij **en** van het feit dat deze partij actief aanwezig is tijdens de communicatie. Authentisering van oorsprong van data geeft zekerheid over de bron. Dit impliceert de integriteit, daar iedere tussenkomst van derden de boodschap zou manipuleren.

Deze lijst met doelstellingen, die men met behulp van de cryptografie wil verwezenlijken, is zeker niet volledig. In het kader van dit eindwerk lichten we slechts een tip van de sluier op van de mogelijkheden die cryptografen in de tweede helft van de voorbije eeuw bedachten. De auteurs van [21] geven een zo volledig mogelijk overzicht van de bestaande toepassingen, eigenschappen en doelstellingen van de cryptografie.

1.3 Sleutelbeheer

Ieder algoritme kan ondergebracht worden in een bepaalde klasse, elk met zijn specifieke eigenschappen en functionaliteit. Afhankelijk van de strategie voor het sleutelbeheer, kiest men voor één van de twee hoofdklassen waarin de cryptografie is onderverdeeld: de publieke of de geheime sleutel cryptografie. Het cryptosysteem gebaseerd op elliptische krommen dat in dit eindwerk voorgesteld wordt, hoort in de klasse van de publieke sleutelalgoritmes thuis.

In de *geheime* of *symmetrische* sleutel cryptografie gaat men er van uit dat beide partijen (en niemand anders) toegang hebben tot dezelfde geheime sleutel. Figuur 1.1 illustreert schematisch de sleuteloverdracht via een veilig kanaal. Een van de grote uitdagingen in geheime sleutel cryptografie is het vinden van een efficiënte methode voor deze sleuteldistributie. Een mogelijke aanvaller wordt verondersteld enkel toegang te hebben tot het onveilig kanaal. Een nadeel is de nood aan veel verschillende sleutels voor de onderlinge communicatie van niet twee maar veel partijen. Voor ieder paar dat wil communiceren, is er één sleutel nodig.

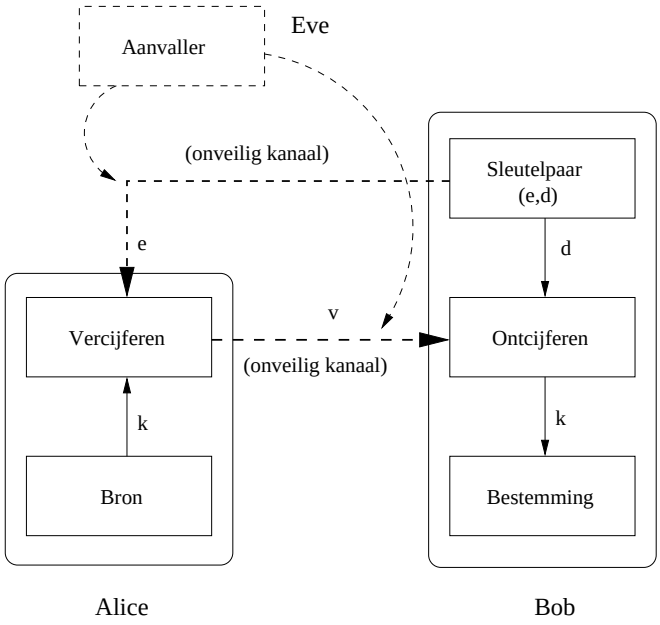


Figuur 1.1: Schematische voorstelling van de geheime sleutel encryptie techniek. De klaartekst k wordt met de geheime sleutel e versleuteld en als versleutelde tekst v verstuurd.

In *publieke* sleutel cryptografie genereert de toekomstige ontvanger vooraf twee verschillende sleutels voor encryptie en decryptie, de publieke en private sleutel. Men noemt dit daarom ook *asymmetrisch* sleutelbeheer. De ontvanger stuurt dan enkel de encryptie-sleutel naar de zender, mogelijk over een onveilig kanaal. Aangezien enkel de ontvanger Bob de passende decryptie-sleutel bezit, is hij de enige die in staat is de inkomende versleutelde tekst te ontcijferen. In Figuur 1.2 zendt Bob de encryptie-sleutel e naar Alice over een willekeurig kanaal, maar houdt de decryptie-sleutel d geheim. Alice kan vervolgens een boodschap naar Bob zenden, die ze met de publieke sleutel e van Bob versleutelt. Bob ontcijfert de versleutelde tekst met zijn unieke geheime decryptie-sleutel d. Enkel Bob kan de klaartekst ontcijferen, maar hij weet niet met zekerheid dat het wel Alice was die zijn publieke sleutel heeft gebruikt om hem een boodschap te sturen. Op deze manier is de vertrouwelijkheid gegarandeerd, de authentisering van entiteiten echter niet.

Anderzijds kan Alice met haar private sleutel de informatie handtekenen, zodat wanneer Bob haar publieke sleutel gebruikt om de klaartekst te lezen, hij met zekerheid weet dat enkel zij de tekst kan gehandtekend hebben. Naast Bob kan ook iedereen die toegang heeft tot de publieke sleutel van Alice de klaartekst achterhalen. Deze duale toepassing van het sleutelpaar, digitale handtekening genoemd, geeft dus wel

authentisering van entiteiten maar dan weer geen vertrouwelijkheid meer.



Figuur 1.2: Schematische voorstelling van de publieke sleutel encryptie techniek. De klaartekst k wordt met de geheime sleutel e versleuteld en als versleutelde tekst v verstuurd. De decryptie-sleutel d, enkel in bezit van Bob, vormt samen met e het sleutelpaar (e, d)

Een groot voordeel van asymmetrisch (of publiek) sleutelbeheer is de gemakkelijke sleuteldistributie van de publieke sleutel. Dit is in het algemeen eenvoudiger dan een veilige manier ontwikkelen om de geheime sleutel bij symmetrische cryptografie te verdelen. Bovendien valt het nadeel van de vele sleutels, die nodig waren voor symmetrische encryptie bij meer dan twee partijen, weg. Iedere partij heeft enkel zijn sleutelpaar van private en publieke sleutel nodig, onafhankelijk van het aantal communicerende partijen.

Publieke sleutelalgoritmes zijn in de regel beduidend trager dan symmetrische sleutel algoritmes. Daarom vindt de publieke sleutel encryptie zijn meeste toepassingen in het afspreken van geheime sleutels over een publiek kanaal. Op die manier komen beide partijen in het bezit van een gemeenschappelijke identieke geheime sleutel, die ze vervolgens kunnen inzetten voor een sneller symmetrische sleutelalgoritme. Andere praktische toepassingen van publieke sleutel algoritmes zijn het versturen van korte data pakketjes, zoals nummers van kredietkaarten en PIN's, en de hoger vernoemde digitale handtekening.

1.4 Publieke sleutel protocols

De vermogen- en tijdsaanvallen op een elliptische krommen gebaseerd cryptosysteem (EKC) vormen het onderwerp van dit eindwerk. Daar het EKC-algoritme zelf een asymmetrisch sleutel algoritme is, gaat deze paragraaf dieper in op enkele van de meest gekende publieke sleutel algoritmes. De werking en eigenschappen van het Diffie-Hellman protocol, het later ontwikkelde RSA protocol en het elliptische kromme

Diffie-Hellman protocol zijn een inleiding op Hoofdstuk 2. Daarin komt de wiskundig complexe elliptische kromme encryptie uitgebreid aan bod.

1.4.1 Diffie-Hellman protocol

In [13] publiceerden W. Diffie en M.E. Hellman voor het eerst een methode voor een veilige sleutel overeenkomst tussen twee partijen over een onveilig kanaal. Hun publieke sleutel encryptie leidde een nieuw hoofdstuk in in de cryptografie. Het Diffie-Hellman protocol maakt gebruik van een

- commutatieve
- één-wegs-functie.

Een één-wegs-functie is een functie die gemakkelijk te berekenen is, maar die rekenkundig zeer moeilijk inverteerbaar is. Ofschoon er nog steeds geen bewijs is geleverd voor het bestaan van één-wegs-functies, wordt algemeen aangenomen dat ze mogelijk zijn. De één-wegs-functie die het Diffie-Hellman protocol gebruikt is dus veilig totdat het tegendeel bewezen is. Dit concept van één-wegs-functies laat toe de boodschap over een onveilig kanaal te sturen. Het Diffie-Hellman protocol is hier een voorbeeld van.

Het protocol bestaat uit drie opeenvolgende stappen waarin beide partijen Alice en Bob informatie uitwisselen over een mogelijk onveilig kanaal. Na de derde stap delen Alice en Bob dezelfde geheime sleutel. Een derde partij Eve, die alle informatie verstuurd over het kanaal heeft afgeluisterd, kan hiermee niet deze geheime sleutel achterhalen.

In een **eerste stap** kiest Alice 3 waarden p, b en x_A:

$$\begin{cases} p: & \text{Een groot priemgetal} \\ b: & \text{De basis, een willekeurig getal} \\ x_A: & \text{Alice haar geheime exponent, een willekeurig getal} \end{cases}$$

Met deze drie waarden berekent Alice dan de publieke waarde y_A:

$$y_A = b^{x_A} \bmod p \tag{1.1}$$

Alice zendt vervolgens b, p en y_A naar Bob. Ze houdt x_A geheim.

In de **tweede stap** herhaalt Bob dezelfde berekening. Hij neemt het priemgetal p en de basis b van Alice, kiest een eigen geheime waarde x_B en berekent zijn y_B:

$$y_B = b^{x_B} \bmod p \tag{1.2}$$

Bob zendt op zijn beurt y_B naar Alice. Ook hij houdt zijn eigen x_B geheim.

In de **derde en laatste stap** berekenen zowel Alice als Bob elk afzonderlijk dezelfde geheime sleutel s. Zonder s ooit over het onveilige kanaal gestuurd te hebben, kennen enkel Alice en Bob na de drie stappen van het Diffie-Hellman protocol de geheime sleutel. Enkel zij kunnen namelijk met hun eigen geheime x_A en x_B en de publieke y_B en y_A de sleutel s berekenen. Tabel 1.1 toont de berekeningen van Alice en Bob in hun laatste stap.

Tabel 1.1: Beide partijen berekenen dezelfde geheime sleutel s.

	Alice		Bob
s	$= y_B^{x_A} \bmod p$	s	$= y_A^{x_B} \bmod p$
	$= (b^{x_B})^{x_A} \bmod p$		$= (b^{x_A})^{x_B} \bmod p$
	$= b^{x_B * x_A} \bmod p$		$= b^{x_A * x_B} \bmod p$

Voorbeeld 1.4.1.1 *Een rekenvoorbeeld verduidelijkt hoe beide partijen met de verschillende ingangswaarden toch hetzelfde resultaat bekomen. Veronderstel dat Alice kiest voor*

$$\begin{cases} p: & = 53 \\ b: & = 18 \\ x_A: & = 8 \end{cases}$$

zodat $y_A = 18^8 \bmod 53 = 24.$

Bob ontvangt de driedelige publieke sleutel van Alice (53,18,24) en start zijn eigen berekeningen. Hij kiest $x_B = 11$ *en vindt* $y_B = 18^{11} \bmod 53 = 48.$

Alice ontvangt op haar beurt de $y_B = 48$ *van Bob. Zowel Alice als Bob beschikken nu over alle gegevens om de dezelfde geheime sleutel* s *te genereren:*

$$\begin{cases} s_A & = 48^8 \bmod 53 = 15 = s \\ s_B & = 24^{11} \bmod 53 = 15 = s \end{cases}$$

De geheime sleutel kunnen ze gebruiken voor de encryptie van hun klaartekst met behulp van een symmetrisch algoritme.

Veronderstel nu dat Eve de communicatie tussen Alice en Bob van Voorbeeld 1.4.1.1 afluistert. Ze bezit dus de publieke sleutel van Alice (p,b,y_A)=(53,18,24) en Bobs $y_B = 48$. Om met deze waarden de geheime sleutel s te achterhalen, moet Eve één van volgende vergelijkingen oplossen:

$$\begin{cases} 24 & = 18^{x_A} \bmod 53 \\ 48 & = 18^{x_B} \bmod 53 \end{cases}$$

Dit probleem is bekend als het *discrete logaritme* probleem. Er zijn wel degelijk wiskundige methoden om x_A en x_B uit de bovenstaande vergelijkingen af te leiden. Maar voor voldoende grote waarden van p, b, x_A en x_B zou dit zelfs voor een zeer krachtig computersysteem veel te lang duren. De rekentijd van de beste, gekende algoritmes voor het oplossen van het discrete logaritme probleem groeit nog steeds subexponentieel met de lengte van de sleutel. Momenteel ligt de veilige grens voor de lengte van de sleutelwaarden bij 1024 bits.

1.4.2 RSA protocol

Het meest populaire publieke sleutel algoritme is RSA. Het kreeg zijn naam van zijn uitvinders Rivest, Shamir en Adleman [28]. Hun protocol baseert zich op het eeuwenoude theorema van Fermat:

$$m^{p-1} \bmod p = 1 \qquad \text{met } p \text{ priem} \tag{1.3}$$

Met p een priemgetal, geeft een willekeurig getal m verheven tot de macht $(p-1)$ en modulo p het resultaat 1.

Euler bouwde hierop verder en ontdekte de gelijkheid:

$$m^{(p-1)(q-1)} \bmod n = 1 \qquad \text{met } p, q \text{ priem en } p * q = n \tag{1.4}$$

De modulus n is het product van twee priemgetallen p en q. Ieder getal m verheven tot de macht $(p-1)(q-1)$ geeft als resultaat 1 mod n. Hier geldt wel de bijkomende voorwaarde dat m en n relatief priem zijn. Met andere woorden, m en n hebben geen gemeenschappelijke delers. Door in Vergelijking (1.4) linker- en rechterlid met m te vermenigvuldigen krijgen we:

$$m^{(p-1)(q-1)+1} \bmod n = m \tag{1.5}$$

Een gekozen ingangswaarde m verschijnt na het uitvoeren van de berekeningen opnieuw als resultaat m aan de uitgang. Het concept van het RSA protocol bestaat uit het opsplitsen in twee delen (één voor Bob en één voor Alice) van deze berekening.

Een wezenlijk verschil met het Diffie-Hellman protocol ligt in manier waarop de communicatie tussen zender en ontvanger start. Paragraaf 1.4.1 beschrijft hoe het initiatief voor een communicatie in het Diffie-Hellman protocol bij de zender Alice ligt. Zij stuurt in de eerste stap haar publieke sleutel naar de ontvanger Bob. In het RSA protocol daarentegen zal de ontvanger Bob in een eerste stap zijn publieke sleutel naar de zender Alice sturen. Stap voor stap verloopt het protocol als volgt:

In een **eerste stap** kiest de ontvanger Bob twee priemgetallen p en q. Hij berekent hiermee

$$\begin{cases} n &= p * q \\ \phi &= (p-1) * (q-1) \end{cases}$$

Bob kiest vervolgens een e (met $1 < e < \phi$) die relatief priem is met zowel $(p-1)$ als $(q-1)$. Tenslotte zoekt hij met behulp van het uitgebreide algoritme van Euclides (zie [21]) een d zodat $e * d = 1 \bmod (p-1)(q-1)$

Bob zijn publieke sleutel bestaat uit:

$$\begin{cases} \mathbf{e}: & \text{de publieke exponent} \\ \mathbf{n}: & \text{het modulogetal} \end{cases}$$

Hij houdt voor zich geheim:

$$\begin{cases} \mathbf{d}: & \text{de tweede, geheime exponent} \\ \mathbf{p, q}: & \text{de priemgetallen die n genereren} \end{cases}$$

In de **tweede stap** gebruikt de zender Alice de publieke sleutel (e, n) van Bob om haar klaartekst m te versleutelen. Alice zendt Bob de versleutelde tekst c:

$$c = m^e \bmod n \tag{1.6}$$

In de **derde en laatste stap** ontcijfert Bob de ontvangen versleutelde tekst c met behulp van zijn geheime tweede exponent d:

$$c^d \bmod n = m^{e*d} \bmod n = m \tag{1.7}$$

De laatste gelijkheid volgt uit Vergelijking (1.5). Het bewijs staat in [21].

Voorbeeld 1.4.2.1 *Een rekenvoorbeeld verduidelijkt het RSA protocol. Stel Bob kiest en berekent*

$p = 5$
$q = 11$
$n = p * q = 55$
$\phi = (p-1) * (q-1) = 40$
$e = 3$
$d = 27$ *zodat* $e * d = 1 \bmod \phi$

Bob maakt zijn publieke sleutel $(e, n) = (3, 55)$ *bekend. Alice kan nu haar klaartekst die ze naar Bob wil zenden met Bob zijn publieke sleutel vercijferen. Stel Alice wil de waarde* $m = 19$ *naar Bob verzenden (staat bijvoorbeeld voor de letter s, de 19de letter van het alfabet). Ze versleutelt haar tekst met de publieke exponent van Bob. De versleutelde tekst c wordt zo:*

$$c = 19^3 \bmod 55 = 39$$

Enkel Bob kent de tweede exponent $d = 27$ *om de klaartekst te berekenen:*

$$39^{27} \bmod 55 = 19 = m$$

Opnieuw rijst de vraag of een derde persoon Eve, die het gesprek tussen Alice en Bob afluistert, in staat is de klaartekst te achterhalen. Eve onderschept de versleutelde tekst c en heeft toegang tot de publieke sleutel van Bob (e, n). Hoe kan ze hieruit m halen? Daarvoor moet ze de tweede exponent d vinden. Eve kan enkel d achterhalen indien ze p en/of q kent. Om de priemgetallen p en q te vinden moet ze het modulogetal n factoriseren. Net zoals het discrete logaritme probleem bij het Diffie-Hellman protocol garant moest staan voor de vertrouwelijkheid, zal voor het RSA protocol het factoriseren van grote getallen in priemfactoren de vertrouwelijkheid bepalen. Algemeen wordt aangenomen dat het discrete logaritme probleem en het factoriseren rekenkundig equivalent zijn. Net zoals bij het Diffie-Hellman protocol geldt ook voor RSA dat de veilige grens voor de lengte van de sleutels op dit moment bij 1024 bits ligt.

1.4.3 Elliptische kromme Diffie-Hellman protocol

Hoofdstuk 2 belicht uitvoerig de wiskundige achtergrond, de verschillende onderdelen en eigenschappen van het elliptische krommen cryptosysteem EKC. Het is eveneens een publieke sleutel algoritme. De één-wegs-functie binnen het elliptische krommen algoritme is de zogenaamde puntvermenigvuldiging. Deze functie inverteren neemt een exponentiële rekentijd in beslag en krijgt de naam *elliptische krommen discrete logaritme probleem* (EKDLP), naar analogie met het Diffie-Hellman DLP. Zowel het Diffie-Hellman discrete logaritme als het RSA factorisatie probleem (FP) vereisen slechts een subexponentiële rekentijd. Het is dus mogelijk met EKC reeds bij kleinere sleutels dezelfde veiligheidsgraad te behalen als met RSA en Diffie-Hellman.

Het elliptische krommen algoritme vindt een mogelijke toepassing in het elliptische kromme Diffie-Hellman protocol (EKDH). Net zoals bij het gebruikelijke Diffie-Hellman protocol zullen na afloop van het EKDH protocol Bob en Alice afzonderlijk dezelfde geheime sleutel gegenereerd hebben. Zonder nu reeds op de details in te gaan, verlopen de drie stappen als volgt:

In een **eerste stap** kiest Alice haar elliptische kromme. Drie parameters a, b en p bepalen haar kromme. Ze kiest vervolgens een punt P op haar kromme. Dit punt

P vermenigvuldigt ze met behulp van het puntvermenigvuldigingsalgoritme met haar geheime sleutel d_A. Ze maakt het resultaat $Q_A = d_A * P$ samen met a, b, p en P publiek.

Publieke sleutel Alice:	a, b, p, P en Q_A
Geheime sleutel Alice:	d_A

In de **tweede stap** neemt Bob de kromme en het punt P van Alice over (a, b, p en P) en genereert met zijn eigen geheime sleutel d_B zijn $Q_B = d_B * P$. Ook hij gebruikt hiervoor het puntvermenigvuldiging-algoritme.

Publieke sleutel Bob:	a, b, p, P en Q_B
Geheime sleutel Bob:	d_B

In de **derde en laatste stap** berekenen zowel Alice als Bob dezelfde geheime sleutel S met hun eigen geheime sleutel d en elkaars publieke punt Q:

Bob:	$d_B * Q_A = d_B * d_A * P = S$
Alice:	$d_A * Q_B = d_A * d_B * P = S$

Een andere bekende toepassing van het EK-algoritme is het digitale handtekening algoritme (Eng.: Digital Signature Algorithm, DSA). EK DSA garandeert de authentisering van entiteiten en laat tegelijkertijd de ontvanger toe de data integriteit van de informatie te verifiëren. Meer hierover in [15].

1.5 Aanvallen op cryptosystemen

Een cryptosysteem is 'gebroken' van zodra de cryptanalyst erin slaagt de geheime of private sleutel k te achterhalen. Deze paragraaf bespreekt enkele mogelijke aanvalsmethoden om dit doel te bereiken.

De *wiskundige* aanval gaat op zoek naar de fundamentele zwakheden van het cryptografisch algoritme. Zo zal bijvoorbeeld de "*brute kracht*" aanval op een symmetrisch algoritme simpelweg iedere mogelijke sleutel uitproberen totdat de juiste sleutel is ontdekt. De sterk toenemende rekenkracht van computers maakt algoritmes die eens veilig geacht werden voor dergelijke brute kracht aanvallen, langzaamaan toch kwetsbaar. Meer geavanceerde wiskundige aanvallen op publieke sleutel algoritmes zoeken naar methoden voor het oplossen van de bovengenoemde factorisatie en discrete logaritme problemen. Hierbij gaat men er van uit dat de aanvaller mogelijk het algoritme kent.

Sinds enkele jaren ontstaat naast de traditioneel wiskundige aanval een nieuwe tak: de *implementatie*-aanval. Deze aanval zoekt de zwakke plekken van een specifieke implementatie van het cryptografisch algoritme. De aanvaller vangt informatie die uitlekt via een neven-kanaal op. Als de verkregen informatie gecorreleerd is met de geheime sleutel brengt dit de veiligheid van het cryptosysteem in het gedrang. Het bekomen van toegang tot de neven-kanalen kan op een destructieve en niet-destructieve manier. Bij destructieve aanvallen wordt de verpakking van de cryptochip verwijderd om een directe toegang tot het chipoppervlak te verkrijgen. Dit laat niet alleen toe het geïntegreerde circuit te observeren, maar ook te manipuleren en bij te sturen. Niet-destructieve aanvallen daarentegen beperken zich tot het observeren van de cryptochip in zijn normale werking. Een smartcard, waar de aanvaller volledige controle heeft over vermogen- en kloktoevoer, is bijzonder kwetsbaar voor deze vorm van aanvallen. Paul Kocher toont in [16] hoe enkel met behulp van de informatie over de rekentijd van

verschillende stappen in het algoritme, een aanval mogelijk is. Niet lang na de publicatie van deze *tijdsaanval* kwam dezelfde Kocher met het idee van de *vermogenaanval*. Hierin tracht hij het vermogenverbruik op een welbepaald tijdstip in verband te brengen met de afzonderlijke instructies van het algoritme. Opnieuw hoopt hij op die manier bijkomende informatie te verzamelen over geheime data.

1.6 Overzicht van dit eindwerk

Dit eindwerk heeft tot doel zwakheden te ontdekken van een elliptische kromme cryptosysteem met behulp van vermogen- en tijdsgebaseerde aanvallen. In een tweede stap geeft deze tekst gerichte tegenmaatregelen om de blootgelegde zwakheden weg te werken. Vermogen- en tijdsaanvallen zijn tot op heden enkel uitgevoerd op ASIC (o.a. smartcard) en dit zowel voor DES [17], AES [4, 11, 8] als EKC [10]. In het kader van dit eindwerk werden de aanvallen uitgevoerd op een implementatie van het systeem op FPGA (Field Programmable Gate Array, een reconfigureerbare hardware component). Een FPGA-implementatie heeft het belangrijke voordeel ten opzichte van een ASIC (Application Specific Integrated Circuit, een hardgebakken chip) dat ze snel aangepast kan worden. Werken met een ASIC zou impliceren dat voor elke wijziging van de algoritmes een volledig nieuwe chip gebakken moet worden. Uit kost- en tijdsoverwegingen is dit onmogelijk. Bovendien kan men aannemen (zie Paragraaf 4.2) dat het vermogenverbruik van deze FPGA-implementatie sterk gelijkt op dat van de overeenkomstige ASIC. Implementaties op FPGA zijn dus zeer geschikt voor chiponderzoek in een vroege fase van ontwerp. De resultaten van dat onderzoek kunnen daarna in de commerciële ASIC's ingepast worden.

Elliptische kromme algoritmen steunen op een relatief moeilijke wiskundige basis. Hoofdstuk 2 gaat daarom wat dieper in op de wiskundige achtergrond. De eigenlijke FPGA-implementatie, het schrijven van de VHDL-code, komt aan bod in Hoofdstuk 3. Hoofdstuk 4 bespreekt de gebruikte meetopstelling voor zowel de vermogenals de tijdsgebaseerde metingen. De metingen zelf komen vervolgens aan bod in Hoofdstuk 5 (tijdsanalyse), Hoofdstuk 6 (eenvoudige vermogenanalyse) en Hoofdstuk 7 (differentiële vermogenanalyse). Deze drie hoofdstukken beschrijven naast de aanvallen zelf ook enkele alternatieven om het systeem resistent te maken tegen de aanvallen. Hoofdstuk 8 vat onze resultaten samen en geeft kort aan op welke vlakken het onderzoek nog kan verdergezet worden.

Deze bundel omvat eigenlijk twee eindwerken:

- Tijdsanalyse van cryptosystemen gebaseerd op elliptische krommen
- Vermogenanalyse van cryptosystemen gebaseerd op elliptische krommen

Het eerste eindwerk voert de tijdsanalyse uit op elliptische kromme cryptosystemen in $\mathrm{GF}(2^n)$ terwijl het tweede de vermogenanalyse uitvoert op elliptische kromme cryptosystemen in $\mathrm{GF}(p)$. Gezien de gemeenschappelijke basis waarop beide onderwerpen steunen, leek een samenvoegen van de resultaten in één eindwerktekst opportuun. De drie auteurs werkten de afgelopen negen maanden nauw samen en twee afzonderlijke verslagen van deze samenwerking zou nodeloze herhalingen met zich meebrengen. Om de twee titels toch van elkaar te kunnen onderscheiden, geeft Tabel 1.2 aan welke delen van dit werk bij welke titel horen.

Tabel 1.2: Verdeling van de hoofdstukken over de twee eindwerktitels

Onderwerp	Tijdsanalyse	Vermogenanalyse
Inleiding	Hoofdstuk 1	
Wiskundige achtergrond	Hoofdstuk 2	
Meetopstelling	Hoofdstuk 4	
Algemeen besluit	Hoofdstuk 8	
Implementatie	Paragraaf 3.3	Paragraaf 3.2
Aanvallen	Hoofdstuk 5	Hoofdstuk 6 en 7

Hoofdstuk 2

Wiskundige achtergrond

2.1 Inleiding

Dit hoofdstuk geeft een overzicht van de mathematische principes waarop het vervolg van deze tekst gebaseerd is. Een meer diepgaande studie van de verschillende onderwerpen bevindt zich in de boeken waarnaar verwezen wordt doorheen het hoofdstuk.

De belangrijkste bewerking in elliptische kromme cryptografie is de puntvermenigvuldiging, waarbij een punt op de curve vermenigvuldigd wordt met een natuurlijk getal. Deze puntvermenigvuldiging is veel complexer dan een traditionele vermenigvuldiging. Ze bestaat uit verschillende kleinere bewerkingen op punten van de kromme: puntverdubbeling, puntoptelling en puntaftrekking. Deze primitieve puntbewerkingen kunnen op hun beurt opgedeeld worden in bewerkingen op elementen van het onderliggende veld: modulaire optelling, modulaire aftrekking, modulaire vermenigvuldiging en modulaire kwadratering. Figuur 2.1 toont deze hiërarchie.

Figuur 2.1: De puntvermenigvuldiging is opgebouwd uit puntoptelling en -verdubbeling, die op hun beurt weer bestaan uit enkele basisbewerkingen

De basisconcepten van groepen- en veldentheorie worden opgefrist in de Bijlagen A en B. Paragraaf 2.2 besteedt extra aandacht aan bewerkingen in velden met even en oneven karakteristiek.

Paragraaf 2.3 introduceert elliptische krommen en de Weierstrass vergelijking. Vervolgens wordt de groepswet, die bewerkingen op punten van elliptische krommen mogelijk maakt, gedefinieerd. Puntverdubbeling en puntoptelling worden besproken. Voor beide wordt zowel een basisalgoritme als een efficiënter algoritme gegeven. Het laatste deel van deze paragraaf, 2.3.4, behandelt de belangrijkste bewerking bij elliptische krommen: puntvermenigvuldiging. Een eenvoudig algoritme wordt gegeven.

2.2 Groepen en velden

Groepen- en veldentheorie in Bijlagen A en B.

2.2.1 Rekenen in $GF(2^n)$

$GF(2^n)$ kan men beschouwen als een vectorruimte van dimensie n over $GF(2)$. Als $\{\beta_{n-1}, \ldots, \beta_1, \beta_0\}$ een basis is voor $GF(2^n)$ over $GF(2)$, dan kan elk element $\alpha \in GF(2^n)$ eenduidig voorgesteld worden door

$$\alpha = \Sigma_{i=0}^{n-1} a_i \beta_i$$

of

$$\alpha = \Pi_{i=0}^{n-1} (\beta_i)^{a_i}$$

met $a_i \in GF(2)$.
Er bestaan verschillende basissen voor $GF(2^n)$ over $GF(2)$ waarvan er drie interessant zijn voor cryptografische toepassingen: de veeltermbasis, de normale basis en de duale basis.
Optelling en aftrekking zijn in elke basis bitgewijze XOR operaties. De vermenigvuldiging wordt enkel uitgelegd in een veeltermbasis, omdat dit de enige basis is die we in deze thesis gaan gebruiken.

De veeltermbasis

Beschouw de verzameling $GF(2)[x]$ van alle veeltermen $f(x) = \Sigma_{i=0}^{k} a_i x^i$ met $a_i \in GF(2)$ en $k \in \mathbb{N}$.
Stel $p(x) \in GF(2)[x]$. Beschouw de deling van elk element van $GF(2)[x]$ door $p(x)$. De mogelijke resten die telkens ontstaan, vormen de verzameling $GF(2)[x]/(p(x))$. Met andere woorden: $GF(2)[x]/(p(x))$ is de verzameling van elementen uit $GF(2)[x]$ modulo $p(x)$.

Definitie 2.2.1.1 *Als $p(x)$ niet kan worden opgesplitst in factoren, dan noemt men $p(x)$ irreduceerbaar.*

Eigenschap 2.2.1.1 *$< GF(2)[x]/(p(x)), +, * >$ is een veld a.s.a. $p(x)$ een irreduceerbare veelterm is.*

Voorbeeld 2.2.1.1 *Om de elementen van het veld $GF(2)[x]/(x^2 + x + 1)$ te bepalen, delen we elke veelterm in $GF(2)[x]$ door $x^2 + x + 1$ en bepalen we de restveelterm. Tabel 2.1 geeft een overzicht van deze berekeningen.*

Het is duidelijk dat er slechts vier restveeltermen mogelijk zijn bij de deling van een element in $GF(2)[x]$ door $x^2 + x + 1$, dus $GF(2)[x]/(x^2 + x + 1) = \{0, 1, x, x + 1\}$.

Eigenschap 2.2.1.2 *Stel $p(x) \in GF(2)[x]$ is een irreduceerbare veelterm van graad n. Dan is $GF(2)[x]/(p(x))$ een veld van kardinaliteit 2^n.*

De voorgaande eigenschap geeft aan dat de elementen van $GF(2^n)$ kunnen voorgesteld worden door veeltermen van graad $n - 1$:

$$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$$

Tabel 2.1: Berekening van de elementen van $GF(2)[x]/(x^2+x+1)$

deeltal	rest
0	0
1	1
x	x
$x+1$	$x+1$
x^2	$x+1$
x^2+1	x
x^2+x	1
x^2+x+1	0
x^3	1
x^3+1	0
x^3+x	$x+1$
x^3+x^2	x
...	...

Deze voorstelling komt overeen met

$$a_{n-1}\alpha^{n-1} + a_{n-2}\alpha^{n-2} + \ldots + a_1\alpha + a_0$$

Hierin is α een nulpunt van de irreduceerbare veelterm, zodat

$$p_n\alpha^n = p_{n-1}\alpha^{n-1} + p_{n-2}\alpha^{n-2} + \ldots + p_1\alpha + p_0$$

met

$$p(x) = p_n x^n + p_{n-1}x^{n-1} + p_{n-2}x^{n-2} + \ldots + p_1 x + p_0$$

$\{\alpha^{n-1}, \ldots, \alpha^2, \alpha, 1\}$ vormt een *veeltermbasis* voor $GF(2^n)$ over $GF(2)$.
De overeenkomstige vectornotatie van een element in $GF(2^n)$ is $(a_{n-1}, \ldots, a_1, a_0)_\alpha$.

Voorbeeld 2.2.1.2 *$GF(2^3)$ is isomorf met $GF(2)[x]/(x^3+x+1)$ want x^3+x+1 is irreduceerbaar. Het nulelement in dit veld is x^3+x+1, dit komt overeen met (0,0,0) in vectornotatie. Tabel 2.2 geeft de overige elementen van $GF(2^3)$ zowel in vector- als in veeltermnotatie.*

Stel dat

$$\begin{aligned} a(x) &= a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 \\ en\ b(x) &= b_{n-1}x^{n-1} + \ldots + b_1 x + b_0 \end{aligned}$$

twee veldelementen zijn en

$$c(x) = c_{n-1}x^{n-1} + \ldots + c_1 x + c_0$$

hun product modulo

$$p(x) = p_n x^n + \ldots + p_1 x + p_0.$$

Tabel 2.2: Vector- en veeltermnotatie van de elementen van $GF(2)[x]/(x^3+x+1)$ met uitzondering van het nulelement.

α^0	$(0,0,1)$	1
α^1	$(0,1,0)$	x
α^2	$(1,0,0)$	x^2
$\alpha^3 = \alpha + 1$	$(0,1,1)$	$x+1$
$\alpha^4 = \alpha^3\alpha = \alpha^2 + \alpha$	$(1,1,0)$	x^2+x
$\alpha^5 = \alpha^3\alpha^2 = \alpha^3 + \alpha^2 = \alpha^2 + \alpha + 1$	$(1,1,1)$	x^2+x+1
$\alpha^6 = \alpha^3\alpha^3 = \alpha^2 + 1$	$(1,0,1)$	x^2+1
$\alpha^7 = \alpha^6\alpha = \alpha^3 + \alpha = 1 = \alpha^0$	$(0,0,1)$	1

$P(x)$ is de irreduceerbare veelterm die gekozen werd om het veld $GF(2^n)$ te genereren uit $GF(2)$. Dus:

$$c(x) = a(x)b(x) \bmod p(x)$$

$$= [b_{n-1}x^{n-1}a(x) + \ldots + b_1 x a(x) + b_0 a(x)] \bmod p(x)$$

Elke term $b_i x^i a(x)$ in deze vergelijking kan recursief berekend worden:

$$b_i x^i a(x) \bmod p(x) = (\ldots((b_i x a(x) \bmod p(x))x \bmod p(x))\ldots)x \bmod p(x)$$

Voorbeeld 2.2.1.3 $a(x) = x+1$, $b(x) = x^2+x$, $p(x) = x^3+x+1$

$$\begin{aligned} c(x) &= x^2(x+1) + x(x+1) + 0(x+1) \pmod{x^3+x+1} \\ &= x^3 + x^2 + x^2 + x \pmod{x^3+x+1} = x^3 + x \pmod{x^3+x+1} = 1 \end{aligned}$$

De normale basis

Een *normale basis* voor $GF(2^n)$ is van de vorm

$$\{\alpha^{2^{n-1}}, \ldots, \alpha^2, \alpha\}$$

waarbij α een wortel is van de irreduceerbare veelterm. Elk element kan voorgesteld worden als $\sum_{i=0}^{n-1} a_i \alpha^{2^i}$ met $a_i \in GF(2)$.

Voorbeeld 2.2.1.4 *Beschouw $GF(2^4)$ met x^4+x^3+1 als irreduceerbare veelterm. Een normale basis voor $GF(2^4)$ is $\{\alpha^8, \alpha^4, \alpha^2, \alpha\}$. Tabel 2.3 geeft de vectornotatie voor de elementen van $GF(2^4)$.*

De duale basis

De beschrijving van een *duale basis* maakt gebruik van het begrip *spoor*.

Definitie 2.2.1.2 *Stel $\alpha \in F = GF(q^n)$ en $K = GF(q)$. Het* spoor $Tr_{F/K}(\alpha)$ *van α over K is gedefinieerd als*

$$Tr_{F/K}(\alpha) = \alpha^{q^{n-1}} + \ldots + \alpha^q + \alpha$$

Tabel 2.3: Vectornotatie van de elementen van $GF(2)[x]/(x^4+x^3+1)$ voor een normale basis.

α^0	$(1,1,1,1)$
α^1	$(0,0,0,1)$
α^2	$(0,0,1,0)$
α^3	$(1,0,1,1)$
α^4	$(0,1,0,0)$
α^5	$(0,1,0,1)$
α^6	$(0,1,1,1)$
α^7	$(1,1,0,0)$

α^8	$(1,0,0,0)$
α^9	$(1,1,0,1)$
α^{10}	$(1,0,1,0)$
α^{11}	$(0,1,1,0)$
α^{12}	$(1,1,1,0)$
α^{13}	$(0,0,1,1)$
α^{14}	$(1,0,0,1)$

Twee basissen $\{\beta_{n-1},\ldots,\beta_1,\beta_0\}$ en $\{\gamma_{n-1},\ldots,\gamma_1,\gamma_0\}$ voor $GF(2^n)$ over $GF(2)$ zijn duaal als, voor $0 \leq i,\, j \leq n-1$, het volgende geldt:

$$Tr(\beta_i\gamma_j) = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$$

Eigenschap 2.2.1.3 *Voor elke basis bestaat er een duale basis.*

Sommige rekenregels maken gebruik van twee basissen die duaal zijn. We bespreken deze regels niet, omdat ze nergens in deze thesis gebruikt worden.

2.2.2 Rekenen in $GF(p)$

Deze paragraaf behandelt kort het rekenen in $GF(p)$, de verzameling gehele getallen modulo p. De Montgomery vermenigvuldiging krijgt hier, wegens haar groot belang voor onze implementatie, extra aandacht.

Definitie 2.2.2.1 *De* modulaire reductie *van het geheel getal z voor modulus p is de gehele rest na de deling van z door p, die steeds in het interval [0, p-1] ligt.*

Voorbeeld 2.2.2.1 *In* $\mathbb{Z}_{25} = \{1,2,\ldots,24\}$ *wordt* $13+16 = 4$, *aangezien* $13+16 = 29 \equiv 4(\mathrm{mod}25)$. *Op dezelfde manier geldt:* $13*16 = 8(\mathrm{mod}25)$

Voor de modulaire optelling en aftrekking volstaat steeds één bijkomende stap om het resultaat tot een element van $GF(p)$ te herleiden.

Immers, neem x en y zodat $0 \leq x, y < p$. Voor elke mogelijke optelling of aftrekking van x en y vinden we :

$$x + y < 2p \tag{2.1}$$

$$x \geq y \Rightarrow 0 \leq x - y < p \tag{2.2}$$

$$x < y \Rightarrow 0 < x + p - y < p \tag{2.3}$$

Voor de modulaire optelling is volgens vergelijking (2.1) enkel in het geval $x + y \geq p$ één bijkomende aftrekking van p nodig. De modulaire aftrekking verlangt alleen voor de waarden van vergelijking (2.3) één bijkomende optelling met p.

De modulaire vermenigvuldiging kan niet op een gelijkaardig eenvoudige manier worden gerealiseerd. Het klassieke algoritme berekent de rest van de deling van $x*y$ door p. De klassieke reductie (zie [21]) is echter een erg dure operatie in hardware. De *Montgomery reductie* biedt een oplossing.

Definitie 2.2.2.2 *Neem $p, R, T \in GF(p)$ zodat $R > p$, $ggd(p, R) = 1$ en $0 \leq T < pR$ Het product TR^{-1} mod p is de* Montgomery reductie van T modulo p voor R.

Het standaard-algoritme voor de Montgomery reductie kan eveneens in [21] gevonden worden. Deze Montgomery reductie ligt aan de basis voor de *Montgomery vermenigvuldiging.*

Definitie 2.2.2.3 *Neem $p, x, y, R \in GF(p)$ met $0 \leq x, y < p$ en welbepaalde voorwaarden voor R (zie Algoritme 2.2.2.1). Het product xyR^{-1} mod p is de* Montgomery vermenigvuldiging van x en y modulo p voor R *(Mont(x,y)).*

Voor deze vermenigvuldiging bestaan een hele reeks algoritmes. Ze zijn echter allemaal gebaseerd op het basis-algoritme 2.2.2.1.

Opmerking: *Voortaan noteren we $a_{n-1} * b^{n-1} + \ldots + a_1 * b^1 + a_0 * b^0$ verkort als $(a_{n-1}, \ldots, a_1, a_0)_b$. b is hier de radix of basis voor de voorstelling van a. Zo stelt men getallen meestal voor met basis 10. Machine berekeningen gebeuren dan weer vaak met basis 2 (binaire representatie).*

Algoritme 2.2.2.1 : Montgomery vermenigvuldiging

INPUT: Gehele getallen $p = (p_{n-1}, \ldots, p_1, p_0)_b$, $x = (x_{n-1}, \ldots, x_1, x_0)_b$,
$y = (y_{n-1}, \ldots, y_1, y_0)_b$ met $0 \leq x, y < p$, $R = b^n$ en $ggd(p, b) = 1$,
en $p' = -p^{-1} \bmod b$.
OUTPUT: $xyR^{-1} \bmod p$.

1. $T \leftarrow 0$. (met $T = (t_n, t_{n-1}, \ldots t_1, t_0)_b$)
2. **For** i from 0 to $(n-1)$ **do**:
   2.1 $u_i \leftarrow (t_0 + x_i y_0)p' \bmod b$.
   2.2 $T \leftarrow (T + x_i y + u_i p)/b$.
3. **If** $T \geq p$ **then** $T \leftarrow T - p$.
4. Return T

Om het product modulo p van $x, y \in GF(p)$ te berekenen, gaat men als volgt te werk:

$$Mont(x, y) = xyR^{-1} \bmod p$$

$$Mont(xyR^{-1}, R^2) = xy \bmod p$$

Eén Montgomery vermenigvuldiging levert echter geen winst op ten opzichte van de gewone vermenigvuldiging. Deze winst ontstaat pas wanneer een hele reeks van vermenigvuldigingen na elkaar uitgevoerd moeten worden. Enerzijds doordat de berekeningen parallel kunnen uitgevoerd worden, anderzijds doordat de bijkomende correctiestappen om R^{-1} weg te werken kunnen herleid worden tot enkele stappen aan het begin en einde van de reeks vermenigvuldigingen. Het volstaat op de ingangen en uitgangen een voor- respectievelijk naberekening uit te voeren. Op die manier werkt het Algoritme (2.2.2.1) steeds met de Montgomery voorstelling van de variabelen.

Voorberekening: Alle ingangen omzetten in hun Montgomery voorstelling:

$$x \rightarrow Mont(x, R^2) = xR^2R^{-1} \bmod p = xR \bmod p$$

Hoofdalgoritme: Alle tussenuitkomsten blijven in Montgomery voorstelling:

$$T \rightarrow Mont(xR, yR) = xR * yR * R^{-1} \mod p = xyR \mod p$$

Naberekening: De uitgang omzetten vanuit de Montgomery voorstelling:

$$resultaat \rightarrow Mont(T, 1) = xyR * 1 * R^{-1} \mod p = xy \mod p$$

Samen met de voor- en naberekeningen zal het Montgomery vermenigvuldigingsalgoritme (2.2.2.1) ingevoerd worden bij de effectieve implementatie van de cryptochip.(zie Paragraaf 3.2.3)

2.3 Berekeningen op elliptische krommen

Deze paragraaf start met de definitie van een elliptische kromme en het invoeren van de affiene Weierstrass vergelijking. Vervolgens definiëren we de groepswet, die berekeningen op elliptische krommen mogelijk maakt. Enkele eenvoudige algoritmes voor de basisbewerkingen op punten van elliptische krommen, verdubbeling van een punt en optelling van twee punten, zullen aan bod komen. Deze bewerkingen bestaan op hun beurt elk uit verschillende modulaire optellingen, vermenigvuldigingen en inversies. Modulaire inversies zijn echter de meest rekenintensieve veldoperaties. Daarom zal het projectieve vlak worden ingevoerd, samen met enkele nieuwe algoritmes voor puntverdubbeling en -optelling in dit vlak, die geen gebruik maken van modulaire inversie. In [22] gaat men dieper in op begrippen behandeld in deze paragraaf.

2.3.1 Elliptische krommen

Deze paragraaf introduceert de definitie van een elliptische kromme in het affiene vlak. De definitie die gebruik maakt van projectieve in plaats van affiene coödinaten bevindt zich in Paragraaf 2.3.3.

Definitie 2.3.1.1 *Beschouw een veld* $\mathbb{F}$*. De verzameling* $\mathbb{F}^2$ *(ook voorgesteld door* $\mathbb{A}^2(\mathbb{F})$ *of eenvoudiger* $\mathbb{A}^2$ *als het veld voor de hand ligt) wordt het* affiene vlak *van het veld* $\mathbb{F}$ *genoemd.*

Definitie 2.3.1.2 *Een* affiene Weierstrass vergelijking *is een vergelijking in* x *en* y *van de vorm*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.4}$$

met $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$*.*

De Weierstrass vergelijking kan singulier of niet-singulier zijn. Voor meer details, zie [22]. Belangrijk voor deze tekst is dat een vergelijking van de eenvoudige vorm

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

singulier is indien

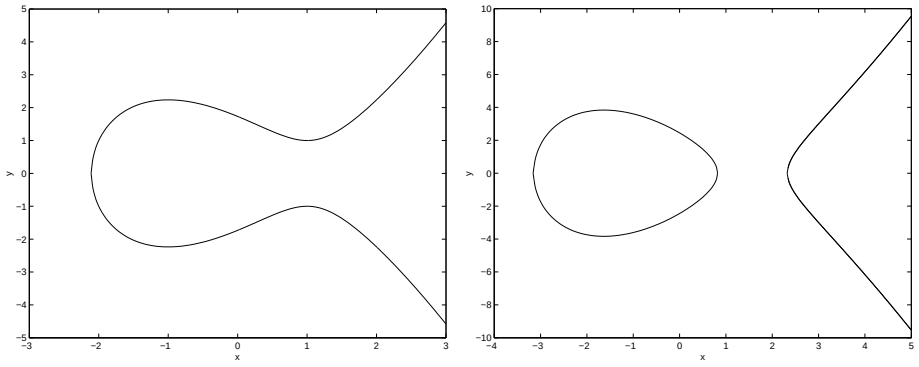
$$4a^3 + 27b^2 = 0.$$

In deze thesis maken we steeds gebruik van een niet-singuliere Weierstrass vergelijking.

Definitie 2.3.1.3 *Een* elliptische kromme E *is de verzameling van alle oplossingen in het affiene vlak* $\mathbb{A}^2(\mathbb{F})$ *van een niet-singuliere Weierstrass vergelijking, samen met het punt op oneindig, aangeduid door* $\mathcal{O}$*.*

Deze definities stellen duidelijk dat een elliptische kromme de set van punten (x, y) is, die in het affiene vlak $\mathbb{A}^2(\mathbb{F}) = (\mathbb{F}) \times (\mathbb{F})$ voldoen aan de Weierstrass vergelijking (2.4), samen met het extra punt $\mathcal{O}$. Als $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$, dan zegt men dat E gedefinieerd is over $\mathbb{F}$, en wordt dit aangeduid door $E/\mathbb{F}$. Als E gedefinieerd is over $\mathbb{F}$, dan is de set van $\mathbb{F}$-rationale punten van E, aangeduid door $E(\mathbb{F})$, de set van punten wiens beide coördinaten in $\mathbb{F}$ liggen, samen met het punt $\mathcal{O}$. Ofschoon verwarrend, wordt toch doorgaans zowel de vergelijking van de elliptische kromme als de set van oplossingen ervan voorgesteld door E.

Voorbeeld 2.3.1.1 *Figuur 2.2 toont de plot van twee elliptische krommen:* $y^2 = x^3 - 3x + 3$ *en* $y^2 = x^3 - 8x + 6$ *over het veld* $\mathbb{R}$*. Figuur 2.3 toont de plot van twee singuliere krommen:* $y^2 = x^3$ *en* $y^2 = x^2(x+1)$ *over het veld* $\mathbb{R}$*. Deze laatste zijn geen elliptische krommen, vermits ze beide een singulariteit in de oorsprong hebben.*



(a) $y^2 = x^3 - 3x + 3$ (b) $y^2 = x^3 - 8x + 6$

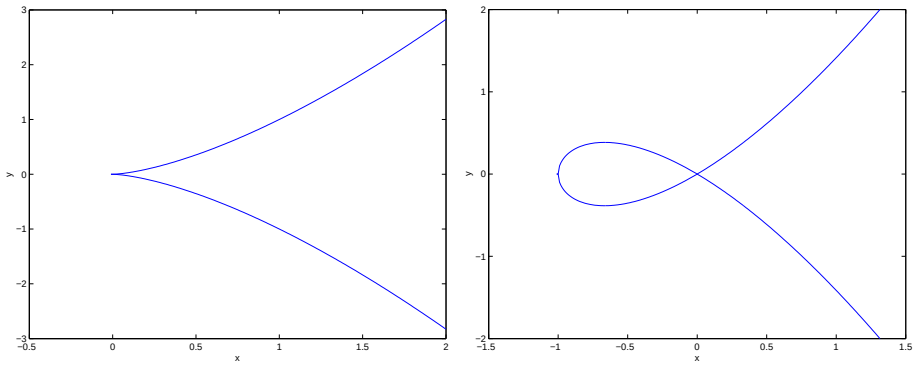
Figuur 2.2: Plot van de elliptische kromme $y^2 = x^3 - 3x + 3$ en $y^2 = x^3 - 8x + 6$ over het veld $\mathbb{R}$

De vergelijking (2.4) wordt vaak de *lange Weierstrass vergelijking* genoemd. Een eenvoudigere vorm van deze Weierstrass vergelijking kan, onder bepaalde condities, verkregen worden door isomorfismen te gebruiken.

In Definitie A.0.0.6 wordt het begrip groepsisomorfisme ingevoerd. Dit concept kan ook toegepast worden op elliptische krommen. Twee elliptische krommen $E1$, $E2$ gedefinieerd over een veld $\mathbb{F}$ zijn isomorf over $\mathbb{F}$ als er morfismen $\phi : E1 \to E2$, $\psi : E2 \to E1$ (ϕ, ψ gedefinieerd over $\mathbb{F}$) bestaan, zodanig dat $\psi \circ \phi$ en $\phi \circ \psi$ $E1$, respectievelijk $E2$, terug afbeelden op zichzelf. De volgende stellingen leggen het verband tussen de notie van isomorfisme van elliptische krommen en de coëfficiënten van de Weierstrass vergelijking die deze krommen definiëren.

Stelling 2.3.1.1 *Twee elliptische krommen E1/*$\mathbb{F}$ *en E2/*$\mathbb{F}$ *gegeven door de vergelijkingen*

$$E1 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$



(a) $y^2 = x^3$ (b) $y^2 = x^2(x+1)$

Figuur 2.3: Plot van de elliptische kromme $y^2 = x^3$ en $y^2 = x^2(x+1)$ over het veld $\mathbb{R}$

$$E2 : y^2 + b_1xy + b_3y = x^3 + b_2x^2 + b_4x + b_6$$

zijn isomorf over $\mathbb{F}$*, aangeduid door* $E1/\mathbb{F} \cong E2/\mathbb{F}$*, als en slechts als er* $u,r,s,t \in \mathbb{F}$*,* $u \neq 0$ *bestaan, zodanig dat de verandering van variabelen*

$$(x, y) \rightarrow (u^2x + r, u^3y + u^2sx + t) \tag{2.5}$$

E1 transformeert in E2. De relatie van isomorfisme is een equivalentierelatie.

De verandering van variabelen (2.5) wordt een *toelaatbare verandering van variabelen* genoemd.
Dus, als $E1 \cong E2$ over $\mathbb{F}$, dan transformeert de verandering van variabelen (2.5) kromme $E1$ in kromme $E2$. Dit levert de volgende set van vergelijkingen op:

$$\begin{aligned} ub_1 &= a_1 + 2s \\ u^2b_2 &= a_2 - sa_1 + 3r - s^2 \\ u^3b_3 &= a_3 + ra_1 + 2t \\ u^4b_4 &= a_4 - sa_3 + 2ra_2 - (t + rs)a_1 + 3r^2 - 2st \\ u^6b_6 &= a_6 + ra_4 + r^2a_2 + r^3 - ta_3 - t^2 - rta_1 \end{aligned} \tag{2.6}$$

Dit isomorfisme is erg handig bij de studie van elliptische krommen, vermits alle resultaten voor één type van kromme ook waar zijn voor alle krommen isomorf met deze eerste kromme. Dit zorgt ervoor dat de Weierstrass vergelijking (2.4) merkelijk vereenvoudigd kan worden voor elliptische krommen in bepaalde velden.

Krommen over $GF(q)$**,** $char(GF(q)) \neq 2, 3$

Neem $E/GF(q)$ een elliptische kromme gegeven door de Weierstrass Vergelijking (2.4). Als $char(GF(q)) \neq 2$, dan transformeert de toelaatbare verandering van variabelen

$$(x, y) \rightarrow (x, y - \frac{a_1}{2}x - \frac{a_3}{2})$$

$E/GF(q)$ in de kromme

$$E'/GF(q) : y^2 = x^3 + b_2x^2 + b_4x + b_6.$$

Merk op dat $E \cong E'$ over $GF(q)$.
Als $char(GF(q)) \neq 2, 3$, dan transformeert de toelaatbare verandering van variabelen

$$(x, y) \rightarrow (\frac{x - 3b_2}{36}, \frac{y}{216})$$

E' verder in de kromme

$$E''/GF(q) : y^2 = x^3 + ax + b.$$

Merk weer op dat $E' \cong E''$ over $GF(q)$, en dus $E \cong E''$ over $GF(q)$.
Bijgevolg kunnen we, als $char(GF(q)) \neq 2, 3$,veronderstellen dat $E/GF(q)$ de vorm

$$E : y^2 = x^3 + ax + b,\ a, b \in GF(q) \tag{2.7}$$

heeft. Dit betekent dat we altijd een Weierstrass vergelijking kunnen selecteren voor E waarvoor $a_1 = a_2 = a_3 = 0$. Deze vergelijking wordt de *korte Weierstrassvorm* genoemd. De sigulariteitseis impliceert nog steeds dat

$$4a^3 + 27b^2 = 0.$$

Bijgevolg kan Stelling 2.3.1.1 betreffende isomorfe krommen sterk vereenvoudigd worden voor elliptische krommen over een veld $GF(q)$ met karakteristiek 2 noch 3.

Stelling 2.3.1.2 *De elliptische krommen*

$$E1/GF(q) : y^2 = x^3 + ax + b$$
$$E2/GF(q) : y^2 = x^3 + \overline{a}x + \overline{b}$$

zijn isomorf over $GF(q)$ *als en slechts als er een* $u \in GF(q)^*$[1] *bestaat, waarvoor geldt:*

$$u^4\overline{a} = a$$
$$u^6\overline{b} = b.$$

Als $E1 \cong E2$ *over* $GF(q)$*, wordt het isomorfisme gegeven door*

$$\phi : E1 \rightarrow E2,\ \phi : (x, y) \rightarrow (u^{-2}x, u^{-3}y),$$

of equivalent

$$\psi : E2 \rightarrow E1,\ \psi : (x, y) \rightarrow (u^2x, u^3y).$$

Krommen over $GF(q)$**,** $char(GF(q)) = 2$

Ook voor elliptische krommen over een veld $GF(q)$ met even karakteristiek bestaat er een minder complexe vergelijking, equivalent met de Weierstrass vergelijking. De afleiding van deze vergelijking is echter veel gecompliceerder dan de afleiding in de vorige paragraaf. Ze kan gevonden worden in [22] en een meer diepgaande studie wordt aan de lezer overgelaten. Deze afleiding resulteert in twee vergelijkingen, equivalent aan de Weierstrass vergelijking:

$$E_1/GF(q) : y^2 + xy = x^3 + a_2x^2 + a_6 \tag{2.8}$$

$$E_2/GF(q) : y^2 + a_3y = x^3 + a_4x + a_6 \tag{2.9}$$

Elk van deze krommenvergelijkingen heeft haar specifieke omstandigheden waaronder ze toepasbaar is. We gaan hier niet dieper op in.

[1] $GF(q)^*$ is de multiplicatieve groep van het veld $GF(q)$

2.3.2 Groepswet

In deze paragraaf definiëren we de optelling op de punten van een elliptische kromme. Het resultaat is een verzameling van elementen, de punten van de elliptische kromme, waarop een operatie is gedefinieerd. Men kan aantonen dat deze verzameling van punten met de operatie een groep is. Dit is de elliptische kromme groep, waarop alle elliptische kromme cryptosystemen gebaseerd zijn. Voor een meer gedetailleerde behandeling van dit onderwerp, zie [6].

Beschouw een elliptische kromme E. Neem vervolgens P en Q, twee verschillende niet-singuliere punten op E. Noem de rechte lijn door P en Q l. Het theorema van Bézout (zie [3]) stelt dat elke rechte lijn l door twee niet-singuliere punten P, Q op E, de kromme E nog op exact één ander niet-singulier punt snijdt. Noem dit punt $\overline{R}$. Neem nu $\bar{l}$ de lijn door $\overline{R}$ en $\mathcal{O}$ en noem het derde snijpunt van $\bar{l}$ met de kromme R. (Dit punt R kan ook gevonden worden door $\overline{R}$ te spiegelen ten opzichte van de x-as.) De *groepswet* wordt nu gedefinieerd als volgt:

$$P + Q = R$$

Een punt P kan ook worden opgeteld bij zichzelf. Dit is de bewerking *puntverdubbeling*. Voor deze verdubbeling is er een soortgelijke procedure: Neem de raaklijn aan de kromme in het punt P. Deze raaklijn moet de kromme snijden in exact één ander punt $\overline{R}$ (Bézout). Spiegel ook hier $\overline{R}$ ten opzichte van de x-as om het punt R te bekomen. Dit punt R noemen we het dubbele van P, genoteerd als $[2]P = P + P$. Als de raaklijn aan de curve in het punt P vertikaal is, snijdt deze lijn de kromme in het punt op oneindig en is $P + P = \mathcal{O}$.

Deze samenstellingswet wordt vaak de *tangent-chord methode* genoemd. Ze wordt grafisch voorgesteld in Figuur 2.4. Figuur 2.4(a) toont de optelling van twee verschillende punten, Figuur 2.4(b) de verdubbeling van P gegeven.

Het kan bewezen worden dat de hierboven beschreven bewerking op punten van een elliptische kromme een additieve communicatieve groepswet definieert op $E(GF(q))$, met het punt op oneindig, $\mathcal{O}$, als neutraal element. Dit punt $\mathcal{O}$ situeert zich oneindig hoog op de y-as. De eigenschap dat drie punten van de kromme opgeteld nul geven als en slechts als ze op een rechte lijn liggen, vat deze wet samen.

Enkele expliciete algebraïsche formules kunnen afgeleid worden voor de hierboven gedefinieerde groepswet. Deze formules zijn geldig in velden van elke karakteristiek, maar in bepaalde gevallen zijn ze sterk vereenvoudigbaar.

Stelling 2.3.2.1 *Zij E een elliptische kromme, gegeven door*

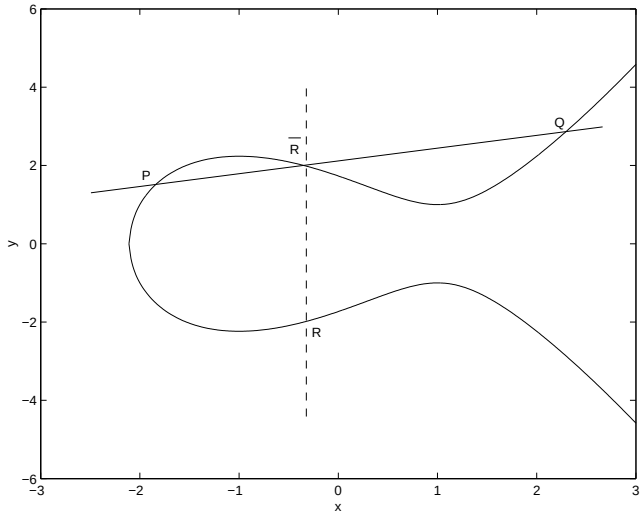
$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

en zij $P_1 = (x_1, y_1)$ en $P_2 = (x_2, y_2)$ punten op deze kromme. Dan is

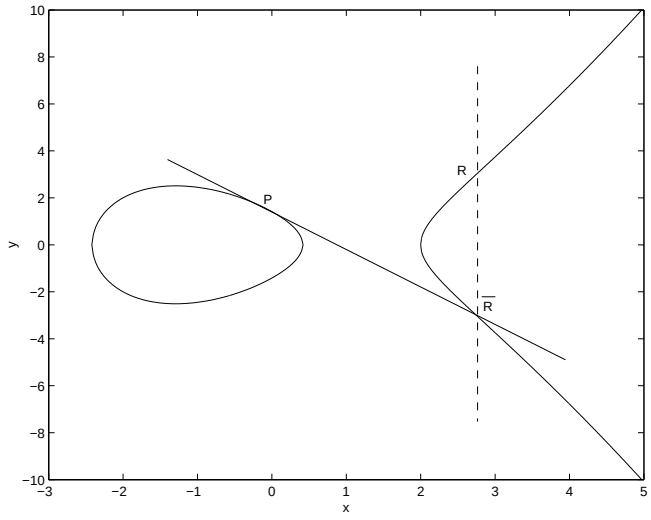
$$-P_1 = (x_1, -y_1 - a_1x_1 - a_3).$$

Definieer

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1},\ \mu = \frac{y_1x_2 - y_2x_1}{x_2 - x_1}$$



(a) Puntoptelling



(b) Puntverdubbeling

Figuur 2.4: Optelling van twee verschillende punten: $P + Q = R$ en optelling van een punt bij zichzelf: $P + P = [2]P = R$

wanneer $x_1 \neq x_2$, *en definieer*

$$\lambda = \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3},\ \mu = \frac{-x_1^3 + a_4x_1 + 2a_6 - a_3y_1}{2y_1 + a_1x_1 + a_3}$$

wanneer $x_1 = x_2$ *en* $P_2 \neq -P_1$.
Als

$$P_3 = (x_3, y_3) = P_1 + P_2 \neq \mathcal{O}$$

dan worden x_3 *en* y_3 *gegeven door de fomules*

$$\begin{aligned} x_3 &= \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \\ y_3 &= -(\lambda + a_1)x_3 - \mu - a_3. \end{aligned}$$

Afhankelijk van de karakteristiek van het onderliggende veld kunnen deze formules al dan niet vereenvoudigd worden.

Krommen over $GF(q)$**,** $char(GF(q)) \neq 2, 3$

Veronderstel het veld $GF(\mathrm{q})$, met $q = p^n$ voor p priem en $p > 3$, $n \geq 1$. Zoals eerder in dit hoofdstuk vermeld, herleidt de elliptische kromme vergelijking zich in dit geval tot de korte Weierstrass vorm (2.7)

$$E_{a,b} : y^2 = x^3 + ax + b$$

In dit geval vereenvoudigen de formules van Stelling 2.3.2.1 tot

$$-P_1 = (x_1, -y_1)$$

Wanneer $x_1 \neq x_2$ neem dan

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

en als $x_1 = x_2$, $y_1 \neq 0$ neem dan

$$\lambda = \frac{3x_1^2 + a}{2y_1}$$

Als

$$P_3 = (x_3, y_3) = P_1 + P_2 \neq \mathcal{O},$$

dan worden x_3 en y_3 gegeven door de formules

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= (x_1 - x_3)\lambda - y_1 \end{aligned}$$

Voorbeeld 2.3.2.1 *De vergelijking* $E : y^2 = x^3 + x + 4$ *over het eindige veld* $\mathbb{Z}_7$ *(de gehele getallen modulo 7) definieert een elliptische kromme. De* $\mathbb{Z}_7$*-rationale punten op* E *zijn*

$$E(\mathbb{Z}_7) = \{\mathcal{O}, (0,2), (0,5), (2,0), (4,3), (4,4), (5,1), (5,6), (6,3), (6,4)\}.$$

Een optelling van twee punten van deze kromme:

$$\begin{aligned} (x_3, y_3) &= (2,0) + (5,1) \\ \lambda &= \frac{1-0}{5-2} = \frac{1}{3}(\mathrm{mod}7) = 5 \\ x_3 &= 25 - 2 - 5 = 18(\mathrm{mod}7) = 4 \\ y_3 &= (2-4)5 - 0 = -10(\mathrm{mod}7) = 4 \end{aligned}$$

Het resultaat is (4,4). Omdat de punten van de elliptische kromme een groep vormen, gesloten onder de optelling, moet het resulterend punt altijd een punt van de elliptische kromme zijn. Dit is ook hier het geval.

Krommen over $GF(q)$**,** $char(GF(q)) = 2$

Opnieuw, zoals in Paragraaf 2.3.1, is de behandeling van krommen over een veld met karakteristiek 2 veel ingewikkelder dan voor karakteristiek p. We leggen ons nu toe op het geval waarbij $q = 2^n$, $n \geq 1$. Herinner dat de elliptische kromme vergelijking in bepaalde gevallen herleid kan worden tot de korte Weierstrass vorm (2.8)

$$E_{a,b} : y^2 + xy = x^3 + a_2x^2 + a_6$$

Onder zekere veronderstellingen (zie [22]) vereenvoudigen de formules van Stelling 2.3.2.1 tot:

$$-P_1 = (x_1, y_1 + x_1)$$

Wanneer $x_1 \neq x_2$ neem dan

$$\lambda = \frac{y_2 + y_1}{x_2 + x_1},\ \mu = \frac{y_1x_2 + y_2x_1}{x_2 + x_1}$$

en wanneer $x_1 = x_2 \neq 0$ neem dan

$$\lambda = \frac{x_1^2 + y_1}{x_1},\ \mu = x_1^2$$

Als

$$P_3 = (x_3, y_3) = P_1 + P_2 \neq \mathcal{O}$$

dan worden x_3 en y_3 gegeven door de formules

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + a_2 + x_1 + x_2 \\ y_3 &= (\lambda + 1)x_3 + \mu \\ &= (x_1 + x_3)\lambda + x_3 + y_1 \end{aligned}$$

Voorbeeld 2.3.2.2 *Beschouw het veld* $GF(2^3)$ *of* $GF(2)[x]/(x^3 + x + 1)$*. Neem* α *nulpunt van* $x^3 + x + 1$*, zodat* $\alpha^3 = \alpha + 1$*. Als basis over dit veld nemen we de veeltermbasis* $\{\alpha^2, \alpha^1 = \alpha, \alpha^0 = 1\}$*. De elementen van dit veld staan in Tabel 2.4.*

Tabel 2.4: Elementen van het veld $GF(2^3)$

–	0	$(0,0,0)$
α^0	1	$(0,0,1)$
α^1	α	$(0,1,0)$
α^2	α^2	$(1,0,0)$
α^3	$\alpha^3 = \alpha + 1$	$(0,1,1)$
α^4	$\alpha^4 = \alpha^3\alpha = \alpha^2 + \alpha$	$(1,1,0)$
α^5	$\alpha^5 = \alpha^3\alpha^2 = \alpha^3 + \alpha^2 = \alpha^2 + \alpha + 1$	$(1,1,1)$
α^6	$\alpha^6 = \alpha^3\alpha^3 = \alpha^2 + 1$	$(1,0,1)$

Beschouw nu de elliptische kromme

$$y^2 + xy = x^3 + \alpha^6 + 1$$

over dit veld $GF(2)[x]/(x^3 + x + 1)$*. De berekening van de punten op deze kromme is vrij rekenintensief. Hieronder vindt u dit voor enkele punten uitgeschreven:*

$$
\begin{aligned}
x = 0 = (0,0,0) &\Longrightarrow y^2 = 1 \\
&\Longrightarrow y = (0,0,1) \Longrightarrow P = (x,y) = ((0,0,0),(0,0,1)) \\
x = 1 = (0,0,1) &\Longrightarrow y^2 + y = 1 + \alpha^6 + 1 = \alpha^6 \\
&\Longrightarrow (y_2\alpha^2 + y_1\alpha + y_0)^2 + (y_2\alpha^2 + y_1\alpha + y_0) = \alpha^2 + 1 \\
&\Longrightarrow y_1\alpha^2 + (y_1 + y_2)\alpha = \alpha^2 + 1 \Longrightarrow \textit{geen oplossing} \\
x = \alpha = (0,1,0) &\Longrightarrow y^2 + \alpha y = \alpha^3 + \alpha^8 + 1 \\
&\Longrightarrow (y_2\alpha^2 + y_1\alpha + y_0)^2 + (y_2\alpha^2 + y_1\alpha + y_0)\alpha = 0 \\
&\Longrightarrow y_2\alpha^2 + y_0\alpha + (y_2 + y_0) = 0 \\
&\Longrightarrow Q = (x,y) = ((0,1,0),(0,1,0)), R = (x,y) = ((0,1,0),(0,0,0))
\end{aligned}
$$

Op deze kromme over $GF(2^3)$ liggen elf punten, waarvan we er hierboven drie berekenden. De punten zijn:

$$
\begin{gathered}
((0,0,0),(0,0,1)),\ ((0,1,0),(0,1,0)),\ ((0,1,0),(0,0,0)), \\
((0,1,1),(0,1,0)),\ ((0,1,1),(1,0,0)),\ ((0,1,1),(1,0,0)), \\
((1,1,0),(1,1,1)),\ ((1,1,1),(0,1,1)),\ ((1,1,1),(1,0,0)), \\
((1,0,1),(1,1,0)),\ ((1,0,1),(0,1,1))
\end{gathered}
$$

De optelling van de punten $((0,1,0),(0,1,0))$ en $((0,1,1),(0,0,1))$ verloopt als volgt:

$$
\begin{aligned}
\lambda &= \frac{y_2 + y_1}{x_2 + x_1} \\
&= \frac{(0,1,1)}{(0,0,1)} = (0,1,1) \\
\lambda^2 &= (0,1,1)^2 = (\alpha+1)^2 = \alpha^2 + 1 = (1,0,1) \\
\mu &= \frac{y_1x_2 + y_2x_1}{x_2 + x_1} \\
&= \frac{(1,1,0) + (0,1,0)}{(0,0,1)} = (1,0,0) \\
x_{sum} &= \lambda^2 + \lambda + a_2 + x_1 + x_2 \\
&= (1,0,1) + (0,1,1) + (1,0,1) + (0,1,0) + (0,1,1) = (0,1,0) \\
y_{sum} &= (\lambda+1)x_3 + \mu = (x_1 + x_3)\lambda + x_3 + y_1 \\
&= ((0,1,1) + (0,0,1))(0,1,0) + (1,0,0) = (1,0,0) + (1,0,0) = (0,0,0)
\end{aligned}
$$

De som van $((0,1,0),(0,1,0))$ en $((0,1,1),(0,0,1))$ bedraagt dus $((0,1,0),(0,0,0))$. Dit punt ligt ook op de kromme.

Het is duidelijk dat de berekening van de som van twee punten met één van de bovenstaande formules verschillende vermenigvuldigingen, inversies, kwadrateringen en optellingen in het onderliggende veld vraagt. De rekenkost van optellingen is zeer klein ten opzichte van de kost van vermenigvuldigingen. De kost van inversies daarentegen is enorm. Het aantal kwadrateringen en vermenigvuldigingen voor één inversie ligt in de grootte-orde van het aantal bits van de modulus. In de volgende paragraaf zullen we daarom overstappen naar het projectieve vlak, waarin deze inversies niet nodig zullen blijken.
In velden met even karakteristiek is kwadratering zeer eenvoudig en heeft dus een zeer lage kost, terwijl in velden met oneven karakteristiek een kwadratering evenveel kost als een vermenigvuldiging.
De kost van optelling en verdubbeling van twee punten wordt in Tabel 2.5 gegeven in termen van het aantal vermenigvuldigingen (V), kwadrateringen (K) en inversies (I), zowel voor velden met karakteristiek $p > 3$ als voor velden met even karakteristiek.

Tabel 2.5: Kost van optelling en verdubbeling in het affiene vlak. V staat voor vermenigvuldiging, I voor inversie, K voor kwadratering.

Bewerking	$char = p > 3$	$char = 2$
Gewone optelling	$3V + 1I$	$2V + 1K + 1I$
Verdubbeling	$4V + 1I$	$2V + 1K + 1I$

2.3.3 Efficiënte algoritmes en projectieve coördinaten

In de vorige paragraaf beschreven we dat, voor het uitvoeren van de bewerkingen puntoptelling en puntverdubbeling in het affiene vlak, het gebruik van de modulaire inversie nodig is. Een alternatief voor deze te dure inversie is het werken in het projectieve vlak. Dit wordt grondig behandeld in [6].

Definitie 2.3.3.1 *Het* projectieve vlak $\mathbb{P}^2(\mathbb{F})$ *is de verzameling van equivalentieklassen onder de relatie* $\sim$ *toegepast op de verzameling van alle vectoren van* $\mathbb{F}^3$*, verschillend van de nulvector. De equivalentierelatie* $\sim$ *wordt gedefinieerd als*

$$(a, b, c) \sim (a', b', c')$$

als en slechts als er een $\lambda \neq 0 \in \mathbb{F}$ *bestaat waarvoor geldt*

$$(a, b, c) = \lambda(a', b', c').$$

De equivalentieklasse die (a, b, c) *bevat noteren we als* $(a : b : c)$.

Er bestaat een natuurlijke betrekking tussen punten van $\mathbb{A}^2(\mathbb{F})$ en punten van $\mathbb{P}^2(\mathbb{F})$ beschreven door de injectie

$$P = (x, y) \rightarrow P^* = (x : y : 1).$$

Deze wordt de *homogenisatie afbeelding* genoemd. De inverse afbeelding

$$Q = (x : y : 1) \rightarrow Q^* = (x, y)$$

is dan de *dehomogenisatie afbeelding*, die enkel uitvoerbaar is op punten in $\mathbb{P}^2(\mathbb{F})$ met Z-coördinaten verschillend van nul. Dergelijke punten worden *affiene punten* genoemd. Het projectieve vlak kan dus gezien worden als een affien vlak waaraan extra punten, met $Z = 0$, werden toegevoegd. Dit zijn de punten op oneindig. In het vervolg van deze tekst zullen we een punt in het projectieve vlak met zijn X-, Y- en Z-coördinaten (X, Y, Z) noteren.

In het projectieve vlak kan een *projectieve Weierstrass vergelijking* worden geformuleerd. Bovendien is het mogelijk een elliptische kromme te definiëren door gebruik te maken van projectieve coördinaten. Beschouw opnieuw een eindig veld $\mathbb{F}$.

Definitie 2.3.3.2 *Een* projectieve Weierstrass vergelijking *is een homogene vergelijking van de derde graad van de vorm*

$$E : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \tag{2.10}$$

met $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$*. De Weierstrass vergelijking wordt* glad *of* niet-singulier *genoemd als voor alle projectieve punten* $P = (X : Y : Z) \in \mathbb{P}^2(\mathbb{F})$ *die voldoen aan*

$$F(X, Y, Z) = Y^2Z + a_1XYZ + a_3YZ^2 - X^3 - a_2X^2Z - a_4XZ^2 - a_6Z^3 = 0, \tag{2.11}$$

er ten minste één van de partiële afgeleiden $\frac{\delta F}{\delta X}$, $\frac{\delta F}{\delta Y}$, $\frac{\delta F}{\delta Z}$ *verschillend is van nul in P. Indien alle drie de partiële afgeleiden voor een bepaald punt P nul worden, is dit punt P een* singulier punt*, en de Weierstrass vergelijking* singulier*.*

Definitie 2.3.3.3 *Een* elliptische kromme *E is de verzameling van alle oplossingen in* $\mathbb{P}^2(\mathbb{F})$ *van een gladde Weierstrass vergelijking. Er bestaat exact één punt op E waarvan de Z-coördinaat gelijk is aan 0, namelijk het punt (0:1:0). We noemen dit punt het* punt op oneindig*, en noteren het als* $\mathcal{O}$*.*

Er bestaat een hele reeks van projectieve voorstellingswijzen. Ze worden uitvoerig besproken in [9]. Bij het gebruik van *conventionele projectieve coördinaten* komt een projectief punt (X, Y, Z), met $Z \neq 0$, dat voldoet aan vergelijking (2.10), overeen met het affiene punt $(\frac{X}{Z}, \frac{Y}{Z})$, dat op zijn beurt voldoet aan de affiene Weierstrass vergelijking (2.4). Vanzelfsprekend geldt ook in het projectieve vlak de vereenvoudigde vorm van de lange Weierstrass vergelijking. Bijvoorbeeld wordt voor velden van oneven karakteristiek, $p > 3$, Vergelijking (2.10) voor conventionele projectieve coördinaten herleid tot

$$E : Y^2Z = X^3 + aXZ^2 + bZ^3$$

Andere projectieve voorstellingswijzen leiden echter tot efficiëntere implementaties van de groepsbewerkingen (zie [9]). Een mogelijke kandidaat is de *gewogen projectieve voorstellingswijze* (ook de *Jacobiaanse voorstellingswijze* genoemd). Hier komt het triplet (X, Y, Z) met $Z \neq 0$ overeen met de affiene coördinaten $(\frac{X}{Z^2}, \frac{Y}{Z^3})$. Dit komt, voor velden met een oneven karakteristiek, $p > 3$, neer op het werken met een gewogen vergelijking van projectieve krommen van de vorm

$$E : Y^2 = X^3 + aXZ^4 + bZ^6$$

Het punt op oneindig $\mathcal{O}$ wordt voorgesteld door het triplet $(\alpha^2, \alpha^3, 0)$, $\alpha \in GF(p^n)^*$. Aangezien de coördinaten van dit punt nooit in bewerkingen worden gebruikt, voldoet in een praktische implementatie ieder triplet met $Z = 0$. De omzetting van affiene naar gewogen projectieve coördinaten is triviaal, terwijl de omgekeerde bewerking een inversie en enkele vermenigvuldigingen en kwadrateringen kost. We kiezen voor het gebruik van de gewogen projectieve coördinaten (Weighted Projective Coordinates, WPC) aangezien zowel puntoptelling als puntverdubbeling efficiënt met deze voorstellingswijze kunnen berekend worden. Deze maakt immers enkel gebruik van de modulaire vermenigvuldiging, zonder naar de inversie te moeten teruggrijpen. Op die manier worden de inversies uit het algoritme zelf geweerd. Enkel indien het uiteindelijke resultaat gewenst is in affiene coördinaten, is er één inversie nodig voor deze omzetting. De tol voor het elimineren van de inversies is een stijging van het aantal vermenigvuldigingen, zodat kiezen voor het al dan niet gebruiken van projectieve coördinaten sterk afhangt van de verhouding $I : V$. Krommen over het veld $GF(p^n)$ met karakteristiek $p > 3$ onderscheiden zich opnieuw van krommen over het veld $GF(2^n)$.

Krommen over $GF(q)$**,** $char(GF(q)) > 3$

Voor velden met karakteristiek $p > 3$ is de gewogen projectieve vereenvoudigde Weierstrass vergelijking van de vorm

$$E_{a,b} : Y^2 = X^3 + aXZ^4 + bZ^6$$

Omzetting van projectieve naar affiene coördinaten kost in dit geval $1I + 4V$. In Algoritme 2.3.3.1 wordt de som $P_3 = (X_3, Y_3, Z_3)$ van twee punten $P_1 = (X_1, Y_1, Z_1)$ en $P_2 = (X_2, Y_2, Z_2)$ berekend in WPC. Neem aan dat $P_1, P_2 \neq \mathcal{O}$, en dat $P_1 \neq \pm P_2$. Deze laatste voorwaarde kan al in een vroeg stadium van de berekening eenvoudig worden nagegaan, zoals hieronder uitgelegd.

Algoritme 2.3.3.1 : Gewone puntoptelling in WPC voor velden met $char(GF(q)) > 3$ (zie [6], p 59)

INPUT: Punten $P_1 = (X_1, Y_1, Z_1)$ en $P_2 = (X_2, Y_2, Z_2)$, $P_1, P_2 \neq \mathcal{O}$ en $P_1 \neq \pm P_2$.
OUTPUT: $P_3 = (X_3, Y_3, Z_3) = P_1 + P_2$

$\lambda_1 = X_1 Z_2^2$
$\lambda_2 = X_2 Z_1^2$
$\lambda_3 = \lambda_1 - \lambda_2$
$\lambda_4 = Y_1 Z_2^3$
$\lambda_5 = Y_2 Z_1^3$
$\lambda_6 = \lambda_4 - \lambda_5$
$\lambda_7 = \lambda_1 + \lambda_2$
$\lambda_8 = \lambda_4 + \lambda_5$
$Z_3 = Z_1 Z_2 \lambda_3$
$X_3 = \lambda_6^2 - \lambda_7 \lambda_3^2$
$\lambda_9 = \lambda_7 \lambda_3^2 - 2X_3$
$Y_3 = (\lambda_9 \lambda_6 - \lambda_8 \lambda_3^3)/2$

De totale kost voor de gewone puntoptelling is 16V. De voorwaarde $P_1 = \pm P_2$ is equivalent met $\lambda_3 = 0$ in Algoritme 2.3.3.1. Stel bovendien dat $\lambda_3 = 0$, dan wordt de voorwaarde $P_1 = P_2$ uitgedrukt door $\lambda_6 = 0$. Van zodra deze laatste voorwaarde wordt gedetecteerd, wordt de routine van de puntverdubbeling, uitgeschreven in Algoritme 2.3.3.2, gebruikt.

Algoritme 2.3.3.2 : Puntverdubbeling in WPC voor velden met $char(GF(q)) > 3$ (zie [6], p 60)

INPUT: Een punt $P_1 = (X_1, Y_1, Z_1)$, $P_1 \neq \mathcal{O}$
OUTPUT: $P_3 = (X_3, Y_3, Z_3) = P_1 + P_1 = [2]P_1$

$\lambda_1 = 3X_1^2 + aZ_1^4$
$Z_3 = 2Y_1 Z_1 \lambda_3$
$\lambda_2 = 4X_1 Y_1^2$
$X_3 = \lambda_1^2 - 2\lambda_2$
$\lambda_3 = 8Y_1^4$
$Y_3 = \lambda_1(\lambda_2 - X_3) - \lambda_3$

De berekening van een puntverdubbeling kost 10V (de kost om te detecteren dat $P_1 = P_2$, 8V, niet meegerekend). Deze kost kan verminderd worden tot 8V wanneer $a = -3$, aangezien in dit geval $\lambda_1 = 3(X_1 - Z_1^2)(X_1 + Z_1^2)$. Dit levert een verbetering tot 2V in plaats van de oorspronkelijke 4V (voor $\lambda_1 = 3X_1^2 + aZ_1^4$) op. Bij de beschrijving

van isomorfismen in Paragraaf 2.3.1 volgt uit Stelling 2.3.1.2 dat een kromme $E_{a,b}$ kan omgevormd worden in de $GF(q)$-isomorfe vorm $E_{a',b'}$ met $a' = -3$ als en slechts als $\frac{-3}{a}$ een vierdemachtswortel heeft in $GF(q)$. Dit geldt voor ongeveer een vierde van alle waarden van a als $q \equiv 1(\text{mod}4)$ en voor de helft van de waarden van a als $q \equiv 3(\text{mod}4)$.
Het verschil in kostprijs voor puntoptelling en puntverdubbeling in gewogen projectieve coördinaten is samengevat in Tabel 2.6. De waarden van deze tabel tonen duidelijk dat de kost van puntverdubbeling in WPC ongeveer de helft is van die van gewone optelling (wanneer $a = -3$), terwijl in affiene coördinaten het verdubbelen de duurste bewerking is. (De kostprijs van verdubbeling en optelling in affiene coördinaten kan worden afgelezen in Tabel 2.5.)

Krommen over $GF(q)$, $char(GF(q)) = 2$

Voor velden met karakteristiek $p = 2$ wordt de vereenvoudigde Weierstrass vergelijking in WPC van de vorm

$$E : Y^2 + XYZ = X^3 + a_2X^2Z^2 + a_6Z^6.$$

Het omzetten van projectieve naar affiene coördinaten kost in dit geval $1I + 3V + 1K$. In Algoritme 2.3.3.3 wordt de som $P_3 = (X_3, Y_3, Z_3)$ van twee punten $P_1 = (X_1, Y_1, Z_1)$ en $P_2 = (X_2, Y_2, Z_2)$ in WPC berekend. Neem aan dat $P_1, P_2 \neq \mathcal{O}$, en dat $P_1 \neq \pm P_2$. Deze laatste voorwaarde kan weer in een vroeg stadium van de berekening eenvoudig nagegaan worden.

Algoritme 2.3.3.3 : Gewone puntoptelling in WPC voor velden met $char(GF(q)) = 2$ (zie [6], p 62)

INPUT: Punten $P_1 = (X_1, Y_1, Z_1)$ en $P_2 = (X_2, Y_2, Z_2)$, $P_1, P_2 \neq \mathcal{O}$
en $P_1 \neq \pm P_2$.
OUTPUT: $P_3 = (X_3, Y_3, Z_3) = P_1 + P_2$

$\lambda_1 = X_1Z_2^2$
$\lambda_2 = X_2Z_1^2$
$\lambda_3 = \lambda_1 + \lambda_2$
$\lambda_4 = Y_1Z_2^3$
$\lambda_5 = Y_2Z_1^3$
$\lambda_6 = \lambda_4 + \lambda_5$
$\lambda_7 = Z_1\lambda_3$
$\lambda_8 = \lambda_6X_2 + \lambda_7Y_2$
$Z_3 = \lambda_7Z_2$
$\lambda_9 = \lambda_6 + Z_3$
$X_3 = a_2Z_3^2 + \lambda_6\lambda_9 + \lambda_3^3$
$Y_3 = \lambda_9X_3 + \lambda_8\lambda_7^2$

De totale kost voor gewone puntoptelling voor krommen over $GF(q)$ met $char(\mathbb{F}) = 2$ is $15V+5K$. Dit kan worden verminderd tot 14V+4K indien $a_2 = 0$, wat geldt voor één van de twee isomorfe klassen van niet-supersinguliere elliptische krommen in $GF(2^n)$. Net zoals in het geval van oneven karakteristiek is de voorwaarde $P_1 = \pm P_2$ equivalent met $\lambda_3 = 0$ en is $P_1 = P_2$ equivalent met $\lambda_6 = 0$. De puntverdubbelingsroutine

wordt uitgeschreven in Algoritme 2.3.3.4, waar het veldelement d_6 gedefinieerd is als $d_6 = \sqrt[4]{a_6} = a_6^{2^{n-2}}$.

Algoritme 2.3.3.4 : Puntverdubbeling in WPC voor velden met $char(GF(q)) = 2$ (zie [6], p 62)

INPUT: Een punt $P_1 = (X_1, Y_1, Z_1)$, $P_1 \neq \mathcal{O}$
OUTPUT: $P_3 = (X_3, Y_3, Z_3) = P_1 + P_1 = [2]P_1$

$Z_3 = X_1 Z_1^2$
$X_3 = (X_1 + d_6 Z_1^2)^4$
$\lambda = Z_3 + X_1^2 + Y_1 Z_1$
$Y_3 = X_1^4 Z_3 + \lambda X_3$

De bewerking puntverdubbeling kost $5V + 5K$ (de kost om te detecteren dat $P_1 = P_2$ niet meegerekend). Merk op dat aangezien kwadratering veel sneller is dan gewone vermenigvuldiging in karakteristiek twee, puntverdubbeling in WPC tot drie keer sneller is dan gewone puntoptelling. Dit in tegenstelling tot het affiene geval, waar beide bewerkingen van ongeveer dezelfde rekenkundige complexiteit zijn (zie Tabel 2.5).

We hebben hier dus geen rekening gehouden met de kost om te detecteren dat $P_1 = P_2$. In het elliptische kromme puntvermenigvuldigingsalgoritme is het namelijk steeds op voorhand duidelijk waar de op te tellen punten dezelfde zijn en waar niet. We grijpen hier dus rechtstreeks naar de puntverdubbeling en dus niet eerst naar de puntoptelling. Zie Paragraaf 3.2.6.

Het verschil in kost voor puntoptelling en puntverdubbeling voor beide karakteristieken $p > 3$ en $p = 2$ wordt samengevat in Tabel 2.6.

Tabel 2.6: Kostprijs van optelling en verdubbeling in WPC. V staat voor vermenigvuldiging, K voor kwadratering.

Bewerking	$char = p > 3$	$char = 2$
Gewone Optelling	$16V$	$(a_2 \neq 0) : 15V + 5K$ $(a_2 = 0) : 14V + 4K$
Verdubbeling	$(a \neq -3) : 10V$ $(a = -3) : 8V$	$5V + 5K$

2.3.4 Puntvermenigvuldiging

De belangrijkste bewerking in elliptische kromme cryptografie is zonder twijfel de puntvermenigvuldiging. Neem een punt P op de kromme en een geheel getal k. De puntvermenigvuldiging van P met k wordt dan genoteerd als $[k]P$. Dit komt neer op $k - 1$ keer een optelling van twee punten:

$$[k]P = \underbrace{P + P + \ldots + P}_{k-1\, keer}$$

De eenvoudigste efficiënte methode voor puntvermenigvuldiging is de binaire methode,

hier weergegeven als Algoritme 2.3.4.1. Dit algoritme is gebaseerd op de binaire expansie van k. Voor een overzicht van nog snellere methodes, zie [6].

Algoritme 2.3.4.1 : Binaire methode voor puntvermenigvuldiging

---

INPUT: Een punt P, een l-bit geheel getal $k = \Sigma_{j=0}^{l-1} k_j 2^j, k_j \in \{0, 1\}$
OUTPUT: $Q = [k]P$

$Q \leftarrow \mathcal{O}$
For j from $l-1$ to 0 by -1 **do**:
    $Q \leftarrow [2]Q$,
    If $k_j = 1$ **then** $Q \leftarrow Q + P$.
Return Q.

Voorbeeld 2.3.4.1 *Beschouw de elliptische kromme $y^2 = x^3 + x + 1$ in het veld $GF(23)$. Het is eenvoudig te verifiëren dat $P = (1, 16)$ een punt van de kromme is. In dit voorbeeld vermenigvuldigen we P met $k = 5$ of (in binaire notatie) $k_2k_1k_0 = 101$. De binaire methode wordt dan als volgt gebruikt om het resultaat van deze berekening te bekomen:*

$Q \leftarrow \mathcal{O}$
$j = 2$: $Q \leftarrow [2]Q = [2]\mathcal{O} = \mathcal{O}$
    $k_2 = 1 \Longrightarrow Q \leftarrow Q + P = \mathcal{O} + (1, 16) = (1, 16)$
$j = 1$: $Q \leftarrow [2]Q = [2](1, 16) = (7, 12)$
    $k_1 = 0 \Longrightarrow Q \leftarrow Q = (7, 12)$
$j = 0$: $Q \leftarrow [2]Q = [2](7, 12) = (17, 3)$
    $k_0 = 1 \Longrightarrow Q \leftarrow Q + P = (17, 3) + (1, 16) = (0, 22)$
Return Q.

Het resultaat $(0, 22)$ moet op de curve liggen, vermits de elliptische kromme groep gesloten is onder puntvermenigvuldiging.

Elliptische kromme cryptografie steunt op het feit dat het zeer moeilijk is om k te berekenen als P en $[k]P$ gegeven zijn. Dit wordt het elliptische kromme discrete logaritme probleem genoemd (Elliptic Curve Discrete Logarithm Problem, EKDLP), dat reeds aan bod kwam in het inleidende Hoofdstuk 1.

2.4 Besluit

Dit hoofdstuk bundelt de nodige wiskundige achtergrond voor een goed begrip van het rekenen met elliptische krommen en elliptische kromme cryptografie in het algemeen. Bouwend op de kennis van algebraïsche groepen- en veldentheorie, beschrijven de opeenvolgende paragrafen stap voor stap de verschillende bewerkingen nodig voor het uitvoeren van een puntvermenigvuldiging.
De eigenschappen en rekenregels van de velden $GF(2^n)$ en $GF(p)$ komen het eerst aan bod. Alle vervolgens opgesomde basisbewerkingen zijn in één van deze twee velden gedefinieerd. Tenslotte doen puntoptelling, puntverdubbeling en puntvermenigvuldiging, de hoofdbewerkingen voor elliptische kromme cryptografie, hun intrede. De

projectieve coördinaten bieden efficiëntere algoritmes voor puntverdubbeling en puntoptelling.

Hoofdstuk 3 beschrijft de gerealiseerde implementatie van het hele cryptosysteem op herprogrammeerbare hardware (FPGA). Vertrekkende van de specifieke algoritmes uit dit hoofdstuk zal iedere deelbewerking tot op poort-niveau worden uitgewerkt.

Hoofdstuk 3

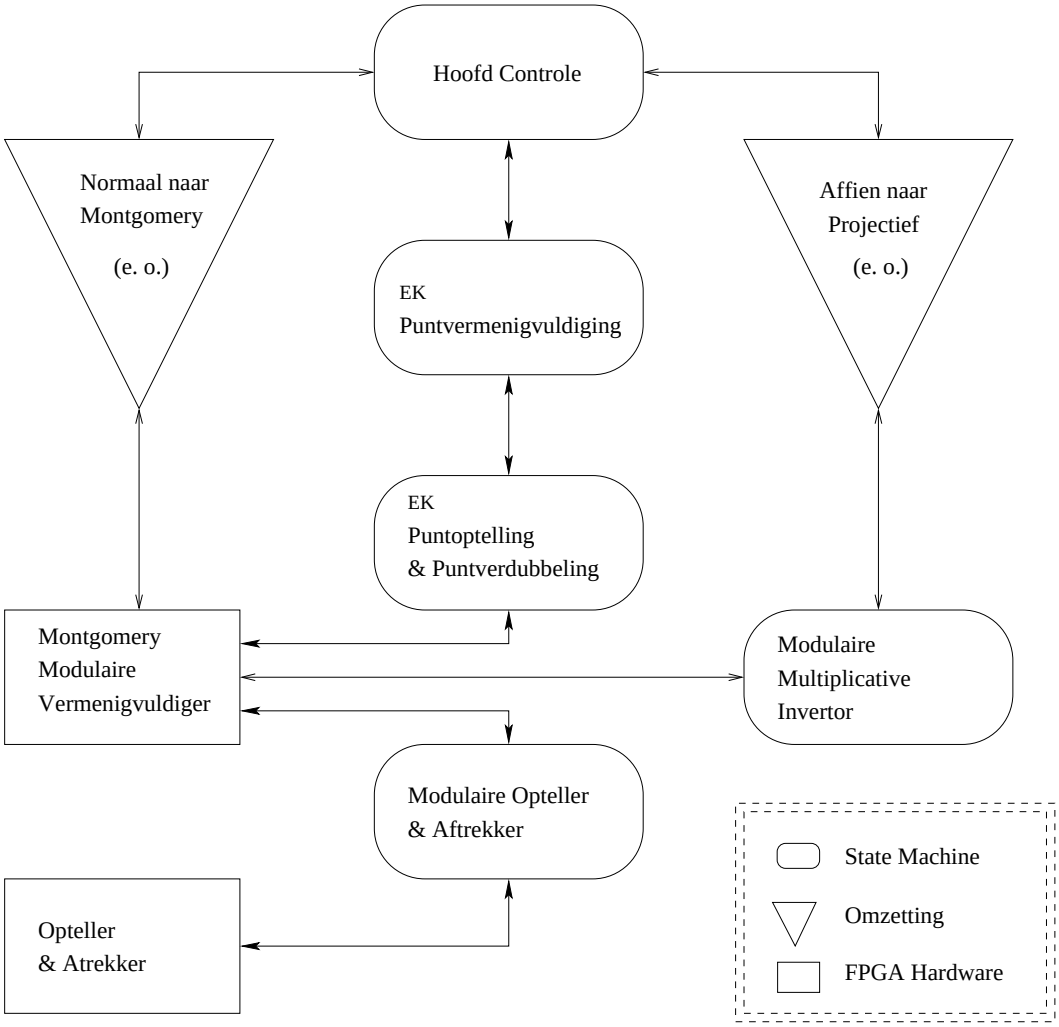
Implementatie

3.1 Inleiding

In de vakliteratuur [18] is er reeds veel aandacht besteed aan mogelijke tijds- en vermogengebaseerde aanvallen op een Smart-card. Met deze thesis wensen we te achterhalen hoe resistent een FPGA-implementatie van een elliptische kromme cryptosysteem is tegen dergelijke side-channel attacks. Om een mogelijke aanval te simuleren maken we gebruik van een VHDL-beschrijving van de puntvermenigvuldiging. Dit hoofdstuk beschrijft de ontwikkeling van deze VHDL-code, die uiteindelijk op een Xilinx Virtex FPGA geladen zal worden. Als ontwerp-omgeving maakten we gebruik van het softwarepakket Xilinx Project Manager.

Uit het vorige hoofdstuk weten we dat een elliptische kromme cryptosysteem gebaseerd is op de puntvermenigvuldiging. We zagen dat deze operatie bestaat uit een opeenvolging van puntoptellingen en -verdubbelingen, die op hun beurt weer op te delen zijn in een sequentie van basisbewerkingen.
Figuur 3.1 toont de hiërarchische blokstructuur van de verschillende bewerkingen nodig in de algoritmes van het elliptische kromme cryptosysteem. De rechthoeken stellen de blokken voor die het uiteindelijke rekenwerk verrichten: de "echte" hardware basisblokken. De ellipsen staan elk voor een state-machine, die de onderliggende blokken verschillende keren zal aanroepen. Een driehoek tenslotte geeft een conversie weer: van het affiene naar het projectieve vlak e.o. en van de normale representatie naar de Montgomery-representatie e.o. Dit hoofdstuk beschrijft de implementatie van elk van de blokken in detail, te beginnen onderaan in de hiërarchie. Paragraaf 3.2 beschrijft nauwkeurig iedere stap voor de bewerkingen in $GF(p)$. Paragraaf 3.3 behandelt diezelfde bewerkingen in $GF(2^n)$.

De VHDL-code besproken in dit hoofdstuk kan terug gevonden worden in Bijlage D. Merk op dat in deze code alle woordlengtes als parameters geprogrammeerd zijn. Dit verhoogt de flexibiliteit sterk en maakt hergebruik mogelijk. Er is tevens steeds voor gezorgd dat de benodigde klokfrequentie totaal onafhankelijk is van de gebruikte woordlengte. Op die manier zal eenzelfde bewerking uitgevoerd op woorden van 160 bits en op woorden van 8 bits gebruik maken van dezelfde klok, maar dus wel meer klokslagen vergen.



Figuur 3.1: Hiërarchische blokstructuur van de verschillende bewerkingen nodig voor de puntvermenigvuldiging. Ellipsen staan voor state-machines, driehoeken symboliseren een conversie en de rechthoeken geven de basisbewerkingen weer.

3.2 Implementatie van aritmetische bewerkingen in $GF(p)$

3.2.1 Opteller-Aftrekker

De basisbewerking voor ieder algoritme hogerop in de hiërarchische boomstructuur is de binaire optelling/aftrekking. Al deze algoritmes maken veelvuldig gebruik van optellingen van woorden waarvan de grootte varieert van enkele bits (4 of 8 bits bij de Montgomery vermenigvuldiging) tot vele bytes (bij de puntoptelling en -verdubbeling voor grote woordlengten).

Opdelen van de optelling

We vermeldden reeds het belang van een klokfrequentie onafhankelijk van de lengte van de op te tellen woorden. Daarvoor is het nodig optellingen van lange woorden op te splitsen in kleinere stukjes van een vaste lengte (de digit-lengte), elk uitgevoerd binnen één klokslag. De optelling van twee 8-bit woorden $A(a_7, \ldots, a_0)$ en $B(b_7, \ldots, b_0)$ kan bijvoorbeeld herschreven worden met een digit-lengte van 4:

$$klokslag1: \quad (a_3, a_2, a_1, a_0) + (b_3, b_2, b_1, b_0) + carry_{in} = carry_0 * 2^4 + (s_3, s_2, s_1, s_0)$$

$$klokslag2: \quad (a_7, a_6, a_5, a_4) + (b_7, b_6, b_5, b_4) + carry_0 = carry_1 * 2^4 + (s_7, s_6, s_5, s_4)$$

Deze oplossing vereenvoudigt de zoektocht naar een geschikte opteller. We zoeken nu namelijk een opteller voor slechts een beperkt aantal (digit-lengte, bv. 4 of 8) bits. We vergelijken hiervoor de eigenschappen van drie verschillende optellers. Meer uitleg in [27]. De *ripple carry opteller*, de *Brent Kung opteller* en de *carry look ahead opteller* komen in aanmerking. We besluiten deze paragraaf met de beschrijving van een vierde opteller, de *carry save opteller*. Deze laatste zullen we gebruiken voor de implementatie van de vermenigvuldiger in Paragraaf 3.2.3.

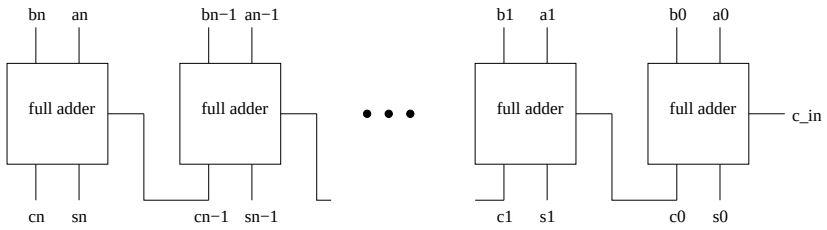
Ripple carry opteller

We beginnen met een van de eenvoudigste optellers: de ripple carry opteller. Deze opteller (Figuur 3.2) berekent de sombits één voor één, van minst beduidend naar meest beduidend, waarbij er telkens een overdrachtsbit (carry) naar het volgende niveau wordt doorgeschoven (rippelen). Beschouw twee woorden $A(a_n, \ldots, a_0)$ en $B(b_n, \ldots, b_0)$ en hun som $S(s_n, \ldots, s_0)$ met $c_{-1} = c_{in} = 0$ (met $\otimes$ de exor-operatie):

$$s_i = a_i \otimes b_i \otimes c_{i-1}$$

$$c_i = a_i * b_i + a_i * c_{i-1} + b_i * c_{i-1}$$

De totale vertraging is gelijk aan $n - 1$ maal de carryvertraging en 1 maal de somvertraging. De snelheid van deze opteller is dus lineair met zijn woordlengte.
Snellere optellers zijn mogelijk, maar dit gaat steeds ten koste van extra oppervlakte. We bespreken er twee: de carry look ahead opteller en de Brent Kung opteller.



Figuur 3.2: Schematische voorstelling van de ripple carry opteller. De ingangen a en b worden opgeteld tot de som s en de carry c.

Carry look ahead opteller

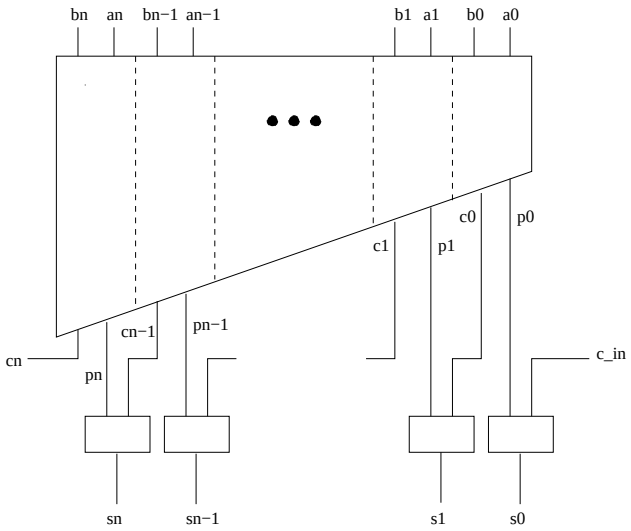
Beschouw twee woorden $A(a_n, \ldots, a_0)$ en $B(b_n, \ldots, b_0)$ en hun som $S(s_n, \ldots, s_0)$. Bij de ripple carry opteller hebben we voor de berekening van de i-de som-bit s_i de $(i-1)$-de overdrachtsbit c_{i-1} nodig. Iedere som-bit kan pas berekend worden van zodra alle voorgaande overdrachtsbits doorheen de hele opteller-keten geschoven (ripple) zijn. Dit maakt de ripple carry opteller erg traag. Met behulp van het *propagatie-generatie* mechanisme vermijdt de carry look ahead opteller het wachten op deze overdrachtsbits:

$$
\begin{aligned}
c_i &= g_i + p_i * c_{i-1} \\
&\text{met} \\
p_i &= a_i \otimes b_i \\
g_i &= a_i * b_i \\
&\text{en nog steeds} \\
s_i &= p_i \otimes c_{i-1}
\end{aligned}
$$

Waaruit volgt dat:

$$
\begin{aligned}
c_0 &= g_0 + p_0 * c_{in} \\
c_1 &= g_1 + p_1 * c_0 = g_1 + p_1 * g_0 + p_1 * p_0 * c_{in} \\
c_2 &= g_2 + p_2 * c_1 = g_2 + p_2 * g_1 + p_2 * p_1 * g_0 + p_2 * p_1 * p_0 * c_{in} \\
&\ldots
\end{aligned}
$$

Door het uitschrijven van deze formules wordt duidelijk dat er voor de berekening van de laatste overdrachtsbit c_n niet moet gewacht worden tot alle n vorige overdrachtsbits zijn berekend. De propagatie-bit p_i en de generatie-bit g_i zijn onmiddellijk uit de woorden A en B berekenbaar. Dit mechanisme wordt geïmplementeerd in de carry look ahead opteller (Figuur 3.3). Deze zal dus sneller zijn dan de ripple carry opteller, ten koste van meer oppervlakte (meer logische poorten). Het probleem bij deze opteller is echter dat de p_i's een zeer grote fan-out krijgen, evenredig met het kwadraat van de gebruikte woordlengte. p_0 bijvoorbeeld komt in de uitdrukking van elke c_i voor. Dit zorgt ervoor dat deze optellers enkel voor relatief korte woordlengtes gebruikt worden. Bovendien neemt de uitdrukking voor de overdrachtsbit toe met de woordlengte, wat ook te zien is aan de driehoekige vorm van de carry look ahead opteller op Figuur 3.3. Dit is helemaal niet gewenst.



Figuur 3.3: Schematische voorstelling van de carry look ahead opteller. De ingangen a en b worden opgeteld tot de som s en de carrybit c_n.

Brent Kung opteller

De Brent Kung opteller probeert de problemen van de carry look ahead opteller op te lossen door over te gaan van een ketting- naar een boomstructuur. Gebruik makend van de associativiteit van de *Brent Kung operator* zijn de n opeenvolgende bewerkingen herleidbaar tot een boomstructuur van diepte $log_2(n)$ (Figuur 3.4):

$$((\ldots((A_0A_1)A_2)A_3)\ldots) = ((\ldots(A_0A_1)(A_2A_3)\ldots))$$

De boomstructuur van de Brent Kung opteller belooft zo een snellere methode te realiseren dan de lineair groeiende uitdrukking voor de berekening van de overdrachtsbits $C(c_n, \ldots, c_1, c_0)$ van de carry look ahead opteller. Iedere cel van de boom (Figuur 3.4 onderaan) staat voor een *Brent Kung operatie* (genoteerd als $\bullet$):

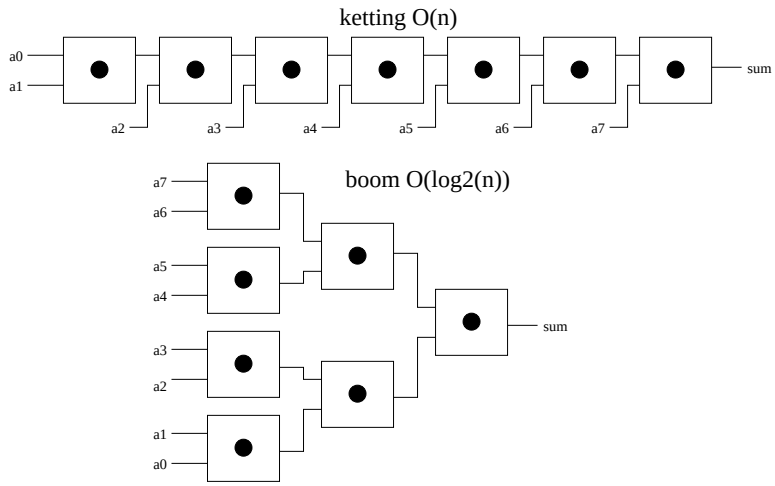
$$\begin{aligned} G(x_2 \bullet x_1) &= g_2 + p_2 * g_1 \\ P(x_2 \bullet x_1) &= p_2 * p_1 \\ met \quad x_i &= (g_i, p_i) \end{aligned}$$

De overgang van ketting- naar boomstructuur is mogelijk dankzij de associativiteit van deze operator $\bullet$. Eén enkele Brent Kung operatie met ingangen x_1 en x_2 en uitgang (G, P) is schematisch voorgesteld in Figuur 3.5.

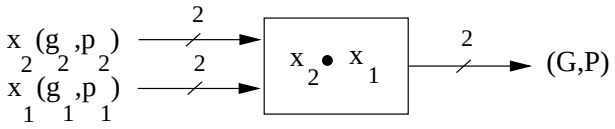
De generatie-propagatie uitdrukkingen voor de overdrachtsbit c_i zijn dan herschrijfbaar tot (voor $c_{in} = 0$):

$$\begin{aligned} c_0 &= g_0 + p_0 * 0 = G(x_0 \bullet 0) \\ c_1 &= g_1 + p_1 * g_0 = G(x_1 \bullet x_0) \\ c_2 &= g_2 + p_2 * g_1 + p_2 * p_1 * g_0 = G(x_2 \bullet x_1 \bullet x_0) \\ &\ldots \end{aligned}$$

Het kritische pad ligt ook bij de Brent Kung opteller in de berekening van de overdrachtsbits. Maar dankzij de boomstructuur vermindert de vertraging van $O(n^2)$ voor



Figuur 3.4: Door associativiteit kan overgegaan worden naar een boomstructuur.



Figuur 3.5: De Brent Kung operatie

de carry look ahead opteller tot $O(log_2(n))$ voor de Brent Kung realisatie. De prijs die je betaalt voor de winst aan snelheid is de toename aan benodigde oppervlakte voor de boomstructuur.

Keuze van de opteller

Het grote snelheidsvoordeel waarvoor de Brent Kung opteller kan zorgen, komt pas tot uiting voor grote woordlengtes. Vermits onze implementatie alle optellingen met veel bits uitspreidt over verschillende klokslagen, komt dit voordeel hier echter niet tot uiting. In dit geval geeft dus een simpele carry look ahead opteller een goede afweging tussen snelheid en benodigde oppervlakte.

Carry save opteller

We bespreken hier nog een vierde soort opteller: de carry save opteller. Deze zal verder in de tekst nodig blijken bij de implementatie van de modulaire vermenigvuldiger (Paragraaf 3.2.3).

De carry save opteller verschilt van de vorige optellers in het aantal uitgangen. Anders dan de vorige drie optellers geeft de carry save opteller niet de n som-bits en één overdrachtsbit als uitgang van de optelling van twee (of drie) n-bit getallen, maar n som-bits en n overdrachtsbits. Deze n-bit opteller is opgebouwd uit n afzonderlijke cellen, die elk maximaal drie bits binnennemen en een som- en overdrachtsbit uitgeven. Elke cel (optelling van 1-bit a, b en c) werkt onafhankelijk van het resultaat van de

overige cellen:

$$\begin{aligned} p &= a \otimes b \\ sum &= p \otimes c \\ carry &= \overline{p}a + pc \\ &= \overline{p}b + pc \end{aligned}$$

In Tabel 3.1 zie je een voorbeeld van deze operatie. De carry save opteller is bovendien zo gebouwd dat de vertraging van de som- en overdrachtsbits gelijk zijn. Dat was niet zo bij de vorige drie optellers. Daar was namelijk steeds de vertraging van de overdrachtsbit kritisch en dus geoptimaliseerd en de somvertraging niet. De berekening van het uiteindelijke resultaat van de optelling vraagt wel een bijkomende stap: resultaat = som + (rest << 1)

Tabel 3.1: Voorbeeld: optelling met behulp van carry save opteller

Ingang	A	10100
	B	10101
	C +	11111
Uitgang	som	11110
	rest	10101

3.2.2 Modulo optellen en aftrekken

Voor de berekeningen in het veld $GF(p)$ implementeren we een opteling/aftrekking modulo een geheel getal. Algoritme 3.2.2.1 geeft de procedure voor de modulaire optelling/aftrekking.

De reden waarom Algoritme 3.2.2.1 steeds eerst zowel stap 1 als stap 2 berekent (en niet enkel stap 2 indien dit nodig is), is tweeërlei. Ten eerste is vergelijken met nul (in stap 3) veel eenvoudiger dan vergelijken met P, waardoor er dus bij de modulaire optelling nood is aan $T2$. Een tweede, veel belangrijkere reden is dat er informatie zou lekken, indien men deze twee stappen niet altijd zou maken. Stel bijvoorbeeld dat bij de modulaire aftrekking stap 2 enkel uitgevoerd wordt indien $T1 < 0$. Een aanvaller

Algoritme 3.2.2.1 : Modulaire optelling (links) en aftrekking (rechts)

INPUT: P, X, Y $\in \mathbb{Z}$ met $0 \leq X, Y < P$

OUTPUT: T= X + Y mod P

1. T_1 = X + Y
2. T_2 = T_1 - P
3. **If** $T_2 < 0$
   $T = T_1$
4. **else**
   $T = T_2$
5. Return T

OUTPUT: T= X - Y mod P

1. T_1 = X - Y
2. T_2 = T_1 + P
3. **If** $T_1 < 0$
   $T = T_2$
4. **else**
   $T = T_1$
5. Return T

---

zou dan aan de duur van de operatie, of het geconsumeerde vermogen kunnen afleiden of deze stap uitgevoerd is en op die manier informatie over $T1$ kunnen vergaren. (Hoofdstuk 4 behandelt dit uitgebreid.) Dit wordt in het gebruikte algoritme vermeden door de twee stappen steeds uit te voeren en de beslissing uit te stellen tot bij het wegschrijven van het resultaat.

Een *finite state machine* (FSM) leent zich goed voor de implementatie van dit algoritme. Voor de uit te voeren optellingen en aftrekkingen gebruiken we steeds de opteller uit Paragraaf 3.2.1. In Bijlage C geeft Figuur C.1 het state-diagramma van de modulaire opteller/aftrekker. Het is opgebouwd uit 3 stappen. Het START signaal is een instructie van een hiërarchisch hogere component die de modulaire opteller/aftrekker aanroept. In elk van de toestanden S1, S2, S3, S4 wordt één van de bewerkingen, beschreven in Algoritme 3.2.2.1, uitgevoerd. Het DONE signaal van de gewone optelling stuurt de overgangen tussen de toestanden. De FSM geeft een DONE signaal naar buiten, wanneer de eigenlijke optelling/aftrekking beëindigd is. Eén modulo optelling vraagt $(2 * woordlengte/digitlengte + 1)$ klokslagen.

3.2.3 Montgomery vermenigvuldiging

Net zoals we bij de modulaire optelling eerst de normale optelling beschreven, gaan we eerst in op de gewone vermenigvuldiging om daarmee de modulaire Montgomery vermenigvuldiging te kunnen bouwen.

Wallace boom vermenigvuldiger

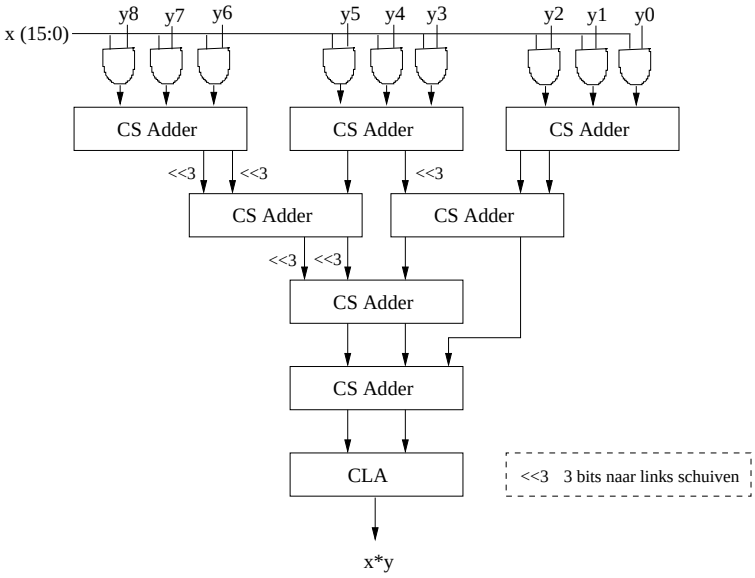
Bij de gewone vermenigvuldiging worden twee woorden $A(a_n, \ldots, a_0)$ en $B(b_m, \ldots, b_0)$ van een mogelijk verschillend aantal bits vermenigvuldigd. Iedere vermenigvuldiging van bitwoorden kan herschreven worden als het optellen van een aantal naar links verschoven termen $b_i * (a_n, \ldots, a_0)$. Tabel 3.2 geeft een voorbeeld van deze werkwijze. Deze aanpak vereist echter een groot aantal optellingen. Een efficiënte en snelle me-

Tabel 3.2: Voorbeeld: vermenigvuldiging van 10100 met 101

	10100	X
*	101	Y
	10100.	$X * y_1$
	00000 .	$X * y_2$
+	10100 .	$X * y_3$
	1100100.	X*Y

thode hiervoor biedt de *Wallace boom*. Een Wallace boom is een implementatie van een boomstructuur van optellers, ontworpen voor een minimale propagatie-vertraging. Een optimale schikking van de carry save optellers garandeert een maximale graad aan parallellisme (zoveel mogelijk optellingen tegelijkertijd). Elke carry save opteller heeft drie ingangen en twee uitgangen (een som- en een carry-uitgang). Zo reduceert elke laag van optellers het aantal op te tellen woorden met een factor 2/3 (compressiefactor). De vertraging is bijgevolg logaritmisch afhankelijk van het aantal op te tellen woorden.

Zoals het voorbeeld in Tabel 3.2 aantoont, realiseert deze boomstructuur van optellers efficiënt en snel de optelling van alle ingangswoorden. De boomstructuur maakt in onze implementatie de som van de partiële producten, gegenereerd door de 'AND'-poorten in de bovenste laag. Op het einde wordt een normale (bij voorkeur snelle) opteller gebruikt om de laatste som en carry tot één resultaat te herleiden. Dit is meestal een carry look ahead opteller (zoals in onze implementatie) of Brent Kung opteller. We noemen dit geheel de Wallace boom vermenigvuldiger. De vertraging van deze vermenigvuldiger zal ongeveer logaritmisch zijn met de woordlengte van één van de twee factoren, wat hem de snelst mogelijke geometrisch-gestructureerde vermenigvuldiger maakt. Figuur 3.6 toont de structuur van deze boom voor een 16*9 bit vermenigvuldiging. Op deze figuur is duidelijk te zien dat de ingangen en tussenresultaten telkens



Figuur 3.6: Wallace vermenigvuldiger voor 16*9 bit: X(16)*Y(9)

per drie worden samengenomen om opgeteld te worden. Hierbij is het belangrijk dat alle ingangen van een carry save opteller de juiste alignatie hebben. Dit is de reden dat sommige tussenresultaten verschoven dienen te worden (zie figuur).

Modulaire vermenigvuldiging

Voor de vermenigvuldiging in een veld maken we gebruik van de Montgomery vermenigvuldiging. Het principe en het basis-algoritme voor deze operatie is reeds besproken in Paragraaf 2.2.2 (Algoritme 2.2.2.1). De implementatie gaat echter uit van een licht gewijzigde vorm ervan, namelijk Algoritme 3.2.3.1.

Deze gewijzigde vorm verschilt van het oorspronkelijke algoritme op drie punten:

1. De reductiestap (stap 3) op het einde valt weg.
2. De ingangsignalen x, y zijn 1 bit langer ($n + 1$ bits).
3. Er zijn twee bijkomende iteratiestappen ($n + 2$ iteraties).

Algoritme 3.2.3.1 : Aangepaste Montgomery vermenigvuldiging

INPUT: Gehele getallen $p = (0, p_{n-1} \dots p_1, p_0)_b$, $x = (\mathbf{0,}\ \boldsymbol{x}_n, \dots, x_1, x_0)_b$,
$y = (\mathbf{0,}\ \boldsymbol{y}_n, \dots, y_1, y_0)_b$ met $0 \leq x, y < \boldsymbol{2p}$,
$R = b^{n+2}$, $ggd(p, b) = 1$ en $p' = -p^{-1} \bmod b$.
OUTPUT: $xyR^{-1} \bmod \boldsymbol{2p}$.

1. $T \leftarrow 0$. (met $T = (t_n, t_{n-1} \dots t_1, t_0)_b$)
2. **For** i from 0 to **(n+1) do**:
   2.1 $u_i \leftarrow (t_0 + x_i y_0) p' \bmod b$.
   2.2 $T \leftarrow (T + x_i y + u_i p)/b$.
3. Return T

De reden voor deze wijziging is opnieuw (net als in Paragraaf 3.2.2) het vermijden dat er informatie lekt naar de buitenwereld. De reductiestap zou immers enkel uitgevoerd worden indien aan een bepaalde voorwaarde voldaan is. Dit is een zwakke plek in het algoritme en mogelijk kwetsbaar voor tijds- en vermogenaanvallen. (zie Hoofdstuk 4) Deze reductiestap weghalen levert wel een uitgang modulo $2p$ in plaats van modulo p op. Dit is niet echt een probleem, gesteld dat de ingangen ook modulo $2p$ mogen zijn. Vaak dient immers het resultaat van een vermenigvuldiging op zijn beurt weer vermenigvuldigd te worden. Dit laatste verklaart waarom het nieuwe algoritme ingangen van $n + 1$ in plaats van n bits heeft. De twee bijkomende iteratiestappen zijn hiervan een rechtstreeks gevolg: onze ingangen zijn beide maximaal b keer groter en de lus (met telkens een deling door b) moet dus twee maal meer uitgevoerd worden (in totaal $n + 2$ maal). Meer uitleg hieromtrent is te vinden in [26].

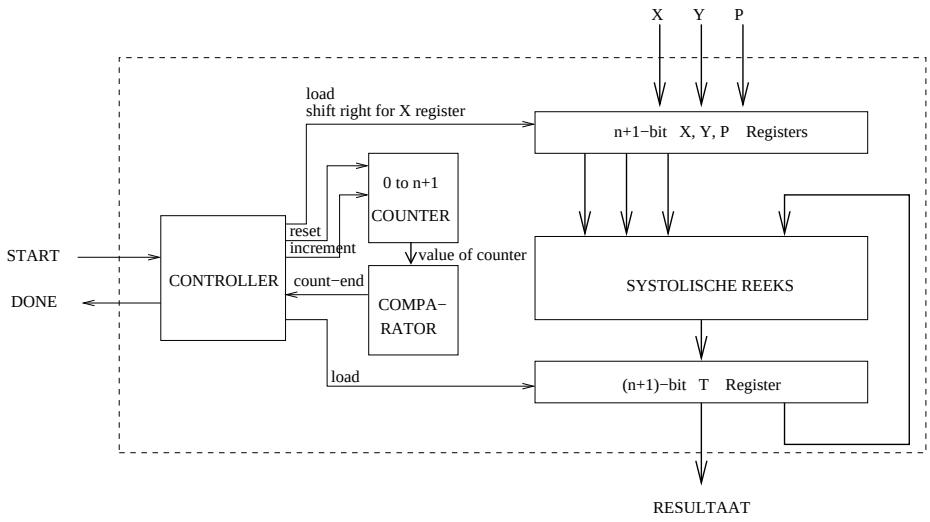
Een voor de hand liggende alternatieve oplossing is de modulaire aftrekking in stap 3 van Algoritme 2.2.2.1 te behouden, maar onafhankelijk van de sprongvoorwaarde het resultaat weg te schrijven naar een register. Enkel in het geval dat $T \geq p$, is dit het register T, anders een 'blind' register dat verder niet meer gebruikt wordt. Dit alternatief is echter niet te verkiezen boven Algoritme 3.2.3.1. Het zou namelijk een modulaire opteller binnenin de Montgomery vermenigvuldiger vereisen, wat een groot gevolg heeft voor de benodigde hoeveelheid transistoren. Aangezien een modulaire vermenigvuldiging en optelling vaak in parallel werken, kan de aanwezige modulaire opteller hier immers niet voor aangewend worden. Een tweede opteller zou dus nodig zijn.

Algoritme 3.2.3.1 bestaat eigenlijk uit een kern (stappen 2.1 en 2.2), die telkens herhaald wordt voor de verschillende digits van de ingang X. In Figuur 3.7 vormt het blokje *systolische reeks* deze kern. Hierbij krijgt het (uitgangs)register T steeds een nieuwe waarde, berekend uit de oude T. Dit zal nog belangrijk blijken bij het bestuderen van het vermogenverbruik. Het schuifregister voor de ingang X, zorgt ervoor dat er telkens een volgende digit van X wordt binnengeschoven.

We gaan nu dieper in op het centrale blokje. Binnenin de lus worden de volgende bewerkingen uitgevoerd:

$$\begin{aligned} u_i &\leftarrow (t_0 + x_i y_0) * p' \bmod b \qquad Stap\ \ 2.1 \\ T &\leftarrow (T + x_i y + u_i p)/b \qquad Stap\ \ 2.2 \end{aligned}$$

Alvorens deze functies te gaan realiseren in hardware, dient eerst de basis b duidelijk



Figuur 3.7: Architectuur Montgomery vermenigvuldiger

gedefinieerd te worden. Zoals vaak in de digitale wereld, is een macht van twee te verkiezen als basis. In het bijzonder in dit algoritme maakt deze basis de deling in stap 2.2 zeer eenvoudig. Een deling door een macht van twee is immers een shift over "exponent"-aantal plaatsen. We stellen b voortaan voor door 2^α. De implementatie zal verschillen afhankelijk van de gekozen macht α. Wij kozen voor een uitwerking met $\alpha = 1$ en $\alpha = 4$.

Radix 1

Radix 1 wil zeggen dat onze basis gelijk is aan 2^1, of $\alpha = 1$. Stappen 2.1 en 2.2 herleiden zich dan tot:

$$u_i \leftarrow (t_0 + x_i y_0) * p' \bmod 2 \qquad Stap \quad 2.1 \tag{3.1}$$

$$T \leftarrow (T + x_i y + u_i p)/2 \qquad Stap \quad 2.2 \tag{3.2}$$

Vermits van p steeds oneven is, zal steeds $p'_0 = 1 \mod 2$. Stap 2.1 herleidt dus verder tot

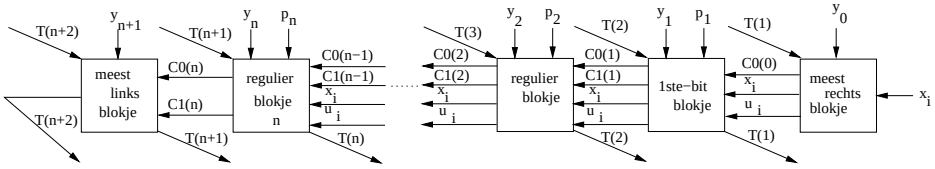
$$u_i \leftarrow (t_0 + x_i y_0) \bmod 2.$$

In de tweede formule kan de deling door twee gezien worden als een verschuiven naar rechts over één positie. Deze regel (stap 2.2) kan nu herschreven worden tot een meer bruikbare vorm:

$$2^2 * c1_{i,j} + 2 * c0_{i,j} + T_{i,j} =$$
$$T_{i-1,j+1} + x_i * y_j + u_i * p_j + 2 * c1_{i,j-1} + c0_{i,j-1} \tag{3.3}$$

met $i = 0, \ldots, n+1$ het aantal ronden van de *for*-lus; $j = 0, \ldots, n+1$ het aantal blokjes in de systolische reeks; $c1_{i,-1} = 0$ en $c0_{i,-1} = 0$. In (3.3) wordt in iedere lus i voor iedere bit y_j herhaald om alle resultaatbits van T_i te berekenen. De implementatie hiervan gebeurt met behulp van een *systolische reeks*. Deze bestaat uit een heel aantal (quasi identieke) blokjes achter elkaar. In plaats van een matrix-structuur van verschillende systolische reeksen onder elkaar (één blokje per bit en per iteratie), hergebruiken

we hier dus één rij van cellen. De werking steunt op het feit dat de j-de cel van de rij in elke i-de iteratie de rol van de (i, j)-de cel van de matrix vervult. Elk blokje voert namelijk de operatie (3.3) uit. Hierbij krijgt het de juiste ingangsbits (y_j en p_j) van buitenaf, de juiste bit van de vorige T-waarde van het T-register en de overdrachtsbits $c0$ en $c1$ van het vorige blokje. Als uitgang produceert het een bit van de nieuwe T-waarde en nieuwe overdrachtsbits $c0$ en $c1$ voor het volgende blokje. De $T_{i,j+1}$-uitgang van het $j+1$-de blokje wordt een ingang voor het j-de blokje gedurende de volgende $(i+1$-ste) iteratie. Dit realiseert de deling door 2 (shift over één positie naar rechts) in Stap 2.2 van het algoritme. De bits x_i en u_i worden door het eerste blokje (dat inwendig zal verschillen van de anderen) ingelezen, respectievelijk geproduceerd. Deze systolische reeks is te zien op Figuur 3.8.

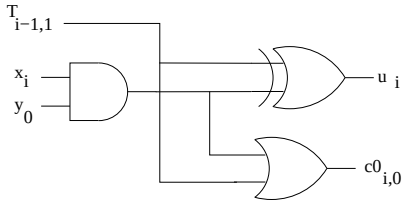


Figuur 3.8: Architectuur systolische reeks. De $n+2$ blokjes produceren iedere klokslag de nieuwe waarden voor de tussenresultaten $T_{n+2}, \ldots, T_1$.

We gaan nu dieper op de interne architectuur van de vier verschillende blokjes in. We beginnen met het *meest rechtse* blokje dat eenmalig uit de ingangen $x_i, y_0, T_{i-1,1}$ het resultaat u_i berekent:

$$u_i = (T_{i-1,1} + x_i y_0) \bmod 2$$

Deze waarde u_i wordt in alle verdere blokjes van de systolische reeks hergebruikt. Zoals Figuur 3.9 illustreert, komt dit overeen met één AND-poort (vermenigvuldiging) en één XOR-poort (optelling modulo twee). Naast u_i geeft dit blokje ook een eerste



Figuur 3.9: 'Meest rechtse' basisbouwblok voor de systolische reeks Radix 1

overdrachtsbit $c0_{i,0}$ terug. De berekening van deze bit volgt uit de Vergelijking (3.3), herschreven voor $j = 0$:

$$2 * c0_{i,0} + T_{i,0} = T_{i-1,1} + x_i * y_0 + u_i \tag{3.4}$$

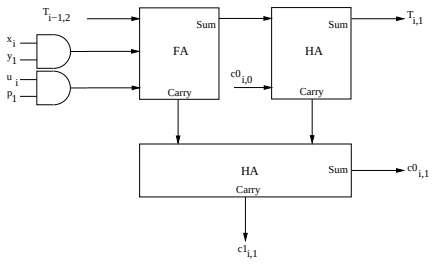
met $i = 0, \ldots, n-1$ het aantal ronden in de *for*-lus; $j = 0, \ldots, n+1$ het aantal blokjes in de systolische reeks; $T_{-1,1} = 0$ het eerste (tussen)resultaat. Tabel 3.3 geeft de waarheidstabel van $c0_{i,0}$ en $T_{i,0}$.

Hieruit volgt $T_{i,0} = 0$ (altijd) en de volgende vergelijkging voor $c0_{i,0}$:

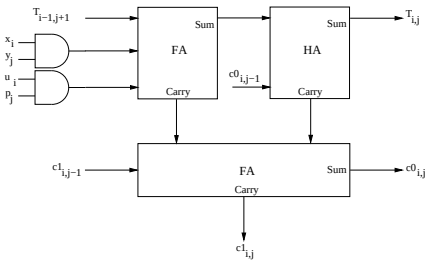
$$c0_{i,0} = T_{i-1,1} + x_i * y_0 \tag{3.5}$$

Tabel 3.3: Waarheidstabel van $c0_{i,0}$ en $T_{i,0}$ (x: 'don't care' condities)

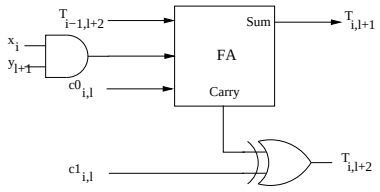
$T_{i-1,1}$	$x_i * y_0$	m_i	$c0_{i,0}$	$T_{i,0}$
0	0	0	0	0
0	0	1	x	x
0	1	0	x	x
0	1	1	1	0
1	0	0	x	x
1	0	1	1	0
1	1	0	1	0
1	1	1	x	x



(a) 1ste bit



(b) regulier



(c) meest links

Figuur 3.10: Architectuur van de blokjes '1ste bit', 'regulier' en 'meest links' voor radix 1

met $i = 0, \ldots n-1$; $\quad j = 0, \ldots, n+1$; $\quad T_{-1,1} = 0$. De implementatie hiervan is een OR-poort, terug te vinden op Figuur 3.9.

Eens u_i en $c0_{i,0}$ gekend zijn, vervolledigen de daaropvolgende $n+1$ blokjes de iteratiestap 2.2 uit Algoritme 3.2.3.1. Ieder blokje berekent één bit van het resultaat T voor het uitgangsregister. De architectuur van ieder van deze $n+1$ blokjes is identiek, behalve die van het eerste (*1ste bit*) blokje en van het laatste (*meest linkse*) blokje in de reeks, die lichtjes vereenvoudigd kunnen worden. Figuur 3.10 toont de drie verschillende architecturen van het eerste, het reguliere en het laatste blokje. De bovenste (1ste bit) onderscheidt zich enkel van de middelste (reguliere) blokjes door zijn verschil in aantal te verwerken overdrachtsbits. Een regulier blokje krijgt twee overdrachtsbits c_0 en c_1 van zijn voorganger. Het blokje '1ste bit' krijgt enkel c_0 en heeft voldoende aan een *Half Adder* (HA) waar voor een regulier blokje een *Full Adder* (FA) nodig is.

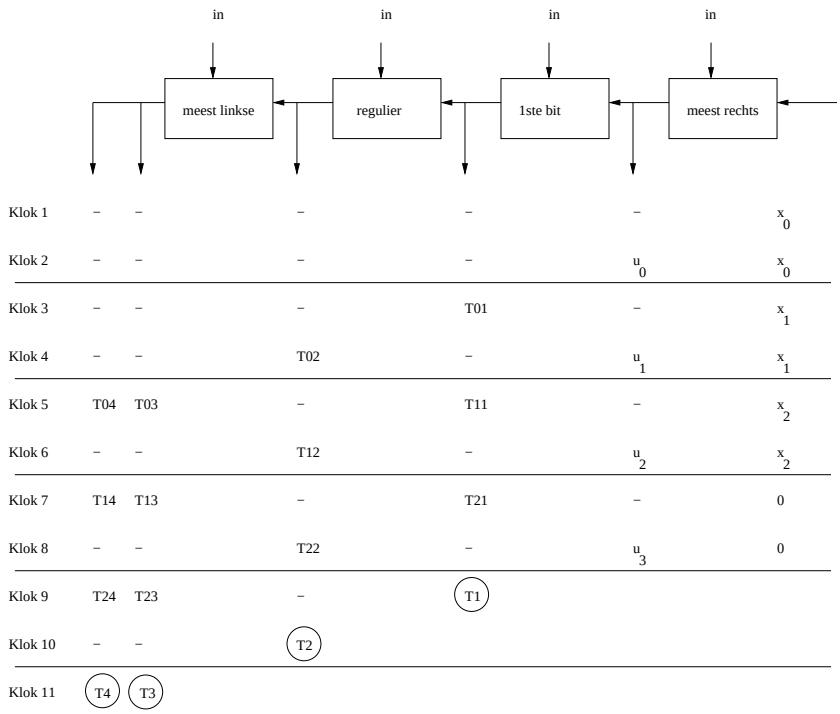
Het laatste blokje van de systolische reeks berekent de laatste twee bits T_{n+2} en T_{n+1} voor het uitgangsregister. Dit is in één blokje mogelijk: aangezien telkens Stap 2.2 uitgevoerd wordt, geldt dat $y_{n+2} = 0$, $p_{n+2} = 0$ en $p_{n+1} = 0$. De berekening van de laatste twee bits van T is dan te vereenvoudigen tot de architectuur van Figuur 3.10(c). Dit blok realiseert de volgende vergelijking:

$$\begin{aligned} 2 * T_{i,n+2} + T_{i,n+1} = \\ T_{i-1,n+2} + x_i * y_{n+1} + 2 * c1_{i,n} + c0_{i,n} \end{aligned} \tag{3.6}$$

We kennen nu de volledige inwendige architectuur van het centrale blok van de Montgomery vermenigvuldiger uit Figuur 3.7. Iedere bit van $X(x_n, \ldots, x_1, x_0)$ uit het schuifregister doorloopt de hele systolische reeks om alle uitgangsbits van het tussenresultaat $T(T_{n+2}, \ldots, T_1)$ te berekenen. Zonder registers tussen de verschillende blokjes van de systolische reeks zouden we dus moeten wachten met het invoegen van de volgende bit van X totdat de laatste resultaatbit T_{n+2} in het uitgangsregister gelezen is. Dit groot kritisch pad zou verplichten aan een lage klokfrequentie te werken. Bovendien zou de lengte van het kritisch pad zo weer afhankelijk zijn van de woordlengte van de factoren X en Y. De $n+1$ pijplijn registers tussen de $n+2$ blokjes van de systolische reeks lossen dit probleem op. Het kritisch pad is nu hetzelfde als het kritisch pad van één reguliere cel en is onafhankelijk van de woordlengte van de operanden.

Deze extra registers maken het uitlezen van het juiste resultaat $T(T_{n+2}, \ldots, T_1)$ wat ingewikkelder. Door de gepijplijnde structuur zullen de verschillende blokjes immers niet op hetzelfde moment in dezelfde iteratie zitten en bits van hetzelfde tussenresultaat T_i uitrekenen. Figuur 3.11 illustreert dit uitleesmechanisme aan de hand van een voorbeeld met $n = 2$. Bijlage D bevat bovendien de volledige hardware-beschrijving in VHDL. De vier blokken bovenaan Figuur 3.11 stellen de $n + 2$ blokken van de systolische reeks voor. Op klokslag 1 wordt rechts de eerste bit van $X(x_n, \ldots, x_1, x_0)$ ingelezen. Het meest rechtse blokje berekent de waarde u_0 en geeft deze op de volgende klokslag door aan het '1ste bit'-blokje. Met deze informatie begint dit blokje te rekenen en geeft dus op de volgende (derde) klokslag zijn eerste uitgang $T_{0,1}$ uit. $T_{0,1}$ is de minst beduidende bit van ons eerste tussenresulataat (resultaat eerste iteratie, $i = 0$). Aangezien de resultaatbit $T_{0,1}$ voor de volgende iteratiestap als ingang van het voorgaande, meest rechtse blokje wordt ingelezen om de nieuwe u_1 te berekenen, heeft dit meest rechtse blokje tussen klokslag 2 en 3 geen functie. Het blokje werkt natuurlijk wel, maar de uitgangen zijn onbelangrijk en worden nergens gebruikt. Ook bij de andere blokjes treedt hetzelfde fenomeen op. Een blokje levert enerzijds ingangen aan een volgend blokje ($c0$ en $c1$), en gebruikt anderzijds resultaten, die dat (volgende) blokje met deze overdrachtsbits berekent. Dit maakt dat niet alleen het meest rechtse, maar alle blokjes één op de twee klokslagen niet relevant zijn (zie de streepjes op Figuur 3.11). Als gevolg hiervan klokken we slechts om de twee klokflanken rechts de volgende bit x_i binnen. Op die manier lezen we dus de laatste ingangsbit $x_n = x_2$ op klokslag $2n + 1 = 5$ in. De eerste resultaatbit $T_{n+1,1} = T_1$ verschijnt op klokflank 9. Tijdens ieder van de daaropvolgende klokflanken zal precies één volgende bit van het eindresultaat $T(T_{n+2}, \ldots, T_2, T_1)$ in het uitgangsregister gelezen worden. Behalve bij het laatste blokje, dat twee resultaatbits berekent. Het duurt na het inlezen van de laatste x-waarde dus nog eens $n + 4 = 6$ klokslagen voordat we de laatste bitwaarden T_{n+1} en T_{n+2} kunnen uitlezen (op klokslag 11).
De gehele berekening, vanaf het inschuiven van de eerste bit van X, tot het uitlezen van de laatste bit van het resultaat, neemt dus exact **3n+5 klokslagen** in beslag (van klokslag 1 tot 11).

Radix 4
In het eerste deel van deze paragraaf bespraken we in detail de implementatie van



Figuur 3.11: Uitleesmechanisme voor de systolische reeks. De omcirkelde waarden van T zijn het uiteindelijke resultaat van de vermenigvuldiging. Alle andere T-waarden zijn tussenresultaten.

de Montgomery vermenigvuldiging met 2^1 als basis en gaven deze werkwijze de naam 'radix 1'. In wat volgt gaan we in op een tweede mogelijke implementatie van hetzelfde algoritme maar met een basis 2^4, of $\alpha = 4$. Aangezien aan de algemene architectuur niets fundamenteel wijzigt, verklaren we hier enkel de verschillen die ontstaan in de implementatie van radix 4.

Stappen 2.1 en 2.2 van de aangepaste Montgomery vermenigvuldiging, Algoritme 3.2.3.1, herschrijven we tot:

$$\begin{aligned} u_i &\leftarrow (t_0 + x_i y_0) * p' \bmod 2^4 \\ T &\leftarrow (T + x_i y + u_i p)/2^4 \end{aligned}$$

De basis van 2^4 (4 bits) heeft consequenties voor de werking van de systolische reeks. Terwijl voor radix 1 ieder blokje uit de systolische reeks steeds één ingangsbit van $Y(0, y_n, \ldots, y_1, y_0)$ en $P(p_{n-1}, \ldots, p_1, p_0)$ inlas en ook precies één resultaatbit T_i berekende, zal voor radix 4 ieder blokje met groepen van **vier** ingangs- en uitgangsbits werken. Opnieuw moet vergelijking 3.3 gerealiseerd worden in ieder blokje. Elk symbool in deze vergelijking stelt nu echter een woord van 4 bits voor. Concreet leest ieder blokje van de systolische reeks 4 opeenvolgende bits van $Y(y_n, \ldots, y_1, y_0)$ en $P(p_n, \ldots, p_1, p_0)$ in, krijgt het meest rechtse blokje aan het begin van iedere iteratiestap steeds 4 opeenvolgende bits van $X(x_n, \ldots, x_1, x_0)$ en hebben ook alle uitgangen $T_{i,j}$ en u_i een lengte van 4 bits. De grotere bitlengte van in- en uitgangen heeft consequenties voor de interne signalen van de systolische reeks. We zetten de directe gevolgen even op een rijtje:

- De overdrachtsbits $c0$ en $c1$ zijn nu overdrachtswoorden van 4 respectievelijk 2 bits lang.

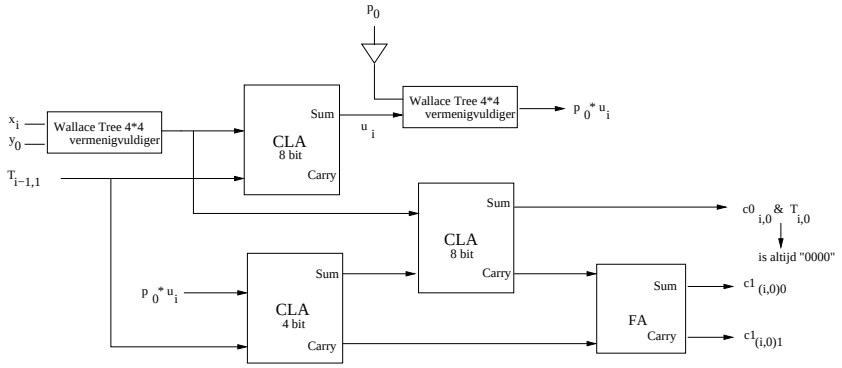
- Ook het meest rechtse blokje genereert nu twee overdrachtswoorden $c0_0$ en $c1_0$ die het eerste van de reguliere blokjes binnenleest (in plaats van enkel een $c0_0$).

Ook op de interne structuur van de verschillende blokjes heeft deze verhoging van radix veel invloed. Per blokje zal er meer (complexere) hardware nodig zijn. Elk blokje moet immers woorden van vier bits verwerken, in plaats van ingangen van één enkele bit zoals bij radix 1. De structuur van deze nieuwe blokjes is te vinden in Figuur 3.12. Deze figuur toont het 'meest rechtse'-, 'regulier'- en 'meest linkse'[1]-blokje. Een '1ste bit'-blokje, zoals bij radix 1, is er hier niet meer. De reden dat dit bij radix 1 vereenvoudigd kon worden, was dat het 'meest rechtse'-blokje geen restbit $c1$ uitgaf. Vermits dit bij radix 4 wel het geval is, valt het onderscheid tussen dit '1ste bit'- en 'regulier'-blokje hier weg. De grootste veranderingen, die aangebracht moesten worden aan de inwendige architectuur van de blokjes is het vervangen van de 'AND'-poorten, die in de radix 1 blokjes een 1-bit vermenigvuldiging implementeerden, door Wallace boom vermenigvuldigers om de 4x4 vermenigvuldiging uit te voeren. Ook zijn op de meeste plaatsen de 'Full Adders' vervangen door carry look ahead optellers, die optellingen van twee 4- of 8-bit woorden aankunnen.
De 4-bit woorden $T_{i,0}$, aangemaakt door het 'meest rechtse'-blokje, zijn allemaal gelijk aan "0000". Dit kan op dezelfde manier bewezen worden, als we aantoonden dat $T_{i,0}$ bij radix 1 steeds "0" is (met een waarheidstabel, zie Tabel 3.3). De deling door 2^4 in Stap 2.2 kan dan herleid worden tot het laten vallen van de 4 minst beduidende ("0000") bits of een shift naar rechts over vier posities. Vermits elk blokje bij radix 4 vier posities vertegenwoordigt, kan deze shift opnieuw geïmplementeerd worden door alle uitgangswoorden $T_{i,j}$ als ingang van het vorige blokje te laten dienen in de volgende klokperiode. Aangezien nu ieder blokje op één klokslag 4 ingangsbits verwerkt en 4 resultaatbits berekent, zal het totale aantal klokcycli voor het uitvoeren van een vermenigvuldiging voor de methode met radix 4 kleiner zijn dan voor die met radix 1. Anderzijds zal het langste pad tussen twee opeenvolgende registers voor radix 4 groter zijn dan voor de eenvoudigere combinatorische schakelingen van radix 1. Een langer kritisch pad betekent vanzelfsprekend een tragere maximale klokfrequentie voor radix 4. De radix 4 methode belooft dus in minder klokslagen dezelfde berekening uit te voeren, maar werkt met een tragere klokfrequentie. Tabel 3.4 vergelijkt de resultaten van radix 1 en radix 4 voor de berekening van een 160x160 vermenigvuldiging. Zo blijkt duidelijk dat de winst in totaal aantal klokslagen voor radix 4 het verlies door een tragere klokfrequentie weer meer dan goed maakt. Voor deze totale snelheidswinst betaalt het wel de prijs van een grotere oppervlakte (aantal transistoren).

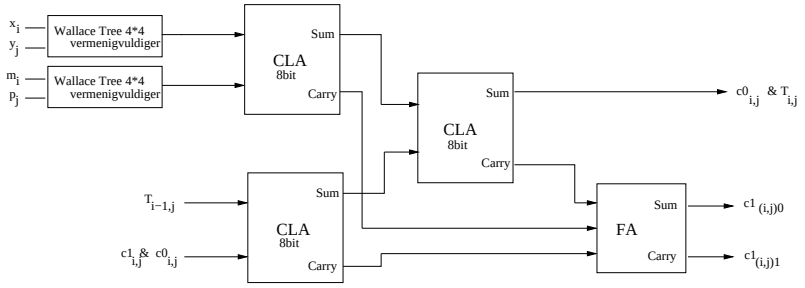
Tabel 3.4: Vergelijkend overzicht radix 1 en radix 4 voor 160x160 vermenigvuldiging in $GF(p)$

	radix 1	radix 4
Totaal aantal klokslagen ($\sharp$)	485	122
Periode klok (P)	7 ns	19 ns
Totale vertraging ($\sharp * P$)	3,39 μs	2,32 μs
Oppervlakte (aantal gates)	1334	4547

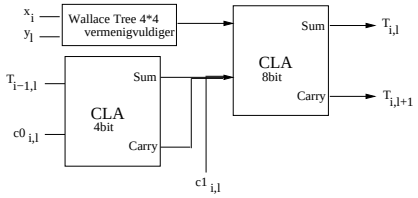
[1] In Figuur 3.12(c) bestaat de tweede ingang van de 8bit CLA uit de concatenatie van ($c1 + Carry(CLA\ 4\ bit)$) en de $Som(CLA\ 4\ bit)$.



(a) meest rechts



(b) regulier



(c) meest links

Figuur 3.12: Architectuur van de blokjes 'meest rechts', 'regulier' en 'meest links' voor radix 4

Het loont zeker de moeite om verder te zoeken naar implementaties voor de Montgomery vermenigvuldiging met hogere radices, zoals 8, 16,.... Dit laat toe het totaal aantal klokslagen (daalt met stijgende radix) en de periode van de klok (stijgt met stijgende radix) duidelijk in kaart te brengen, om zo de optimale radix-lengte te kunnen bepalen, die de totale vertraging minimaliseert.

3.2.4 Omzettingen

De vorige Paragrafen 3.2.1 tot 3.2.3 behandelden de hardware-implementaties van achtereenvolgens optelling, modulaire optelling en Montgomery vermenigvuldiging. Om deze bewerkingen efficiënt te kunnen uitvoeren, definieerden we in het vorige hoofdstuk verschillende voorstellingswijzen. We bespreken in deze paragraaf de omzettingen tussen deze voorstellingswijzen. De driehoeken in het hiërarchisch overzicht van Figuur 3.1 stellen omzettingen van een bepaalde voorstellingswijze naar een andere voor. Voor de puntvermenigvuldiging is de *affien* $\leftrightarrow$ *projectief omzetting* en de *normaal* $\leftrightarrow$ *Montgomery omzetting (e.o.)* nodig. Beide worden bondig toegelicht.

Normale versus Montgomery voorstelling

De aangepaste Montgomery vermenigvuldig, Algoritme 3.2.3.1, geeft voor de ingangen x, y, p

$$xyR^{-1} \bmod (2p)$$

als resultaat. Verschillende opeenvolgende vermenigvuldigingen waarbij de bekomen producten hergebruikt worden als ingang, zouden zo telkens de factor R^{-1} accumuleren. De *Montgomery voorstelling* biedt een oplossing. Ieder ingangswoord van onze berekeningen initialiseren we eerst door er een Montgomery vermenigvuldiging met R^2 op toe te passen:

$$x \rightarrow Mont(x, R^2) = xR^2R^{-1} \bmod p = xR \bmod p \tag{3.7}$$
$$y \rightarrow Mont(y, R^2) = yR^2R^{-1} \bmod p = yR \bmod p \tag{3.8}$$

De bekomen waarde is de Montgomery voorstelling van het woord. Na deze omzettingsstap kunnen we op de resultaten een onbeperkt aantal Montgomery vermenigvuldigingen (en andere bewerkingen bv. modulaire optelling) toepassen. Er is geen opstapeling meer van de factor R^{-1} en de tussenresultaten zijn dus telkens in de Montgomery voorstellingswijze.

$$T \rightarrow Mont(xR, yR) = xR * yR * R^{-1} \bmod p = xyR \bmod p \tag{3.9}$$

Wanneer we het uiteindelijke resultaat van onze opeenvolgende modulaire vermenigvuldigingen in de normale voorstellingswijze willen uitlezen, volstaat een enkele naberekening:

$$resultaat \rightarrow Mont(T, 1) = xyR * 1 * R^{-1} \bmod p = xy \bmod p \tag{3.10}$$

De omgekeerde omzetting is namelijk niets meer dan de Montgomery vermenigvuldiging met 1. Dit zorgt ervoor dat de R^{-1} verdwijnt.

De implementatie van dit omzettingsalgoritme gebeurt met behulp van een state-machine. Bij het doorlopen van de state-machine wordt voor iedere ingangswaarde het hardware blok voor de Montgomery vermenigvuldiging opgeroepen. Voor de puntvermenigvuldiging zullen deze ingangen de affiene coördinaten x, y en de parameter a van de korte Weierstrass Vergelijking (2.7) en 1 zijn. De 1 is de Z-coördinaat van het affiene punt (x, y). De R^2, nodig als bijkomende ingang voor de omzetting naar Montgomery voorstellingswijze, wordt niet intern berekend, maar is een externe ingang.

In Bijlage C geeft Figuur C.2(a) de architectuur van de state machine voor deze omzetting.

Affiene versus projectieve coördinaten

Zoals reeds vermeld in Paragraaf 2.3.3 van het vorige hoofdstuk, is het vaak nuttig over te stappen van affiene coördinaten (x- en y-coördinaat) naar projectieve coördinaten (X-, Y- en Z-coördinaat). Werken in het projectieve in plaats van het affiene vlak vermindert namelijk het aantal moeilijk te berekenen inversies.
Er bestaan tal van projectieve voorstellingswijzen, die elk een andere implementatie van de puntvermenigvuldiging met zich meebrengen. Voor een overzicht, zie [26]. Wij kozen in onze implementatie voor de *aangepaste gewogen projectieve voorstellingswijze* (ook wel de *aangepaste Jacobiaanse voorstellingswijze*).

De Jacobiaanse voorstellingswijze herschrijft de affiene coördinaten $(\frac{X}{Z^2}, \frac{Y}{Z^3})$ als het triplet (X, Y, Z) met $Z \neq 0$ (Paragraaf 2.3.3).
Wij werken met een licht gewijzigde vorm van de gewogen projectieve voorstellingswijze. Naast een X-, Y- en Z-coördinaat, houden we ook de waarde aZ^4 bij (met a de parameter a uit Vergelijking 2.7, de korte Weierstrass vorm van een elliptische kromme). De invoering van deze vierde coördinaat zal het rekenwerk in de algoritmes voor puntverdubbeling en puntoptelling (Algoritme 3.2.5.1) erg vereenvoudigen.
Een punt in *aangepaste gewogen projectieve voorstellingswijze* of *aangepaste Jacobiaanse voorstellingswijze* is dus van de vorm:

$$(X, Y, Z, aZ^4)$$

De omzetting van affiene coördinaten naar aangepaste Jacobiaanse is triviaal:

$$\text{affien} : (x, y) \rightarrow \text{projectief} : (x, y, 1, a)$$

De omgekeerde omzetting van Jacobiaans naar affien is moeilijker:

$$\text{projectief} : (X, Y, Z, aZ^4) \rightarrow \text{affien} : (\frac{X}{Z^2}, \frac{Y}{Z^3})$$

Dit impliceert namelijk dat de coördinaat Z geïnverteerd moet worden, wat een moeilijke wiskundige bewerking is. Deze ene inversie (per uitgangswoord) weegt echter niet op tegen de talloze inversies die nodig zijn indien we de hele berekening in het affiene vlak zouden uitvoeren (Paragraaf 2.3.3).

Ook de implementatie van deze omzetting is een eenvoudige state-machine, die de Montgomery vermenigvuldiger en het invertor-blok oproept. In Bijlage C toont Figuur C.2(b) de architectuur van de state-machine. Opnieuw komt de instructie START van de hiërarchisch hogere component die de omzetting oproept. De overgangen tussen de 5 verschillende states wordt naast het START signaal ook door het DONE signaal van de Montgomery modulaire vermenigvuldiging (MMM) en het DONE signaal van het inversieblok (INV) gestuurd. DONE gaat hoog van zodra de het resultaat van de omzetting berekend is.

De modulaire inversie, die nodig is bij de omzetting, maakt gebruik van het Theorema van Fermat [21], $a^{-1} = a^{p-2} \bmod p$, waarbij $\text{ggd}(a, p) = 1$ en de waarde p een priemgetal is. Aangezien de krommen in $GF(p)$ liggen met p priem, laat het theorema toe de inverse van a modulo p te vinden. Daarvoor volstaat het op a een modulaire machtsverheffing met exponent $p - 2$ uit te voeren. Het algoritme voor de modulaire machtsverheffing is opgebouwd uit kwadrateringen en vermenigvuldigingen. De verschillende stappen staan beschreven in Algoritme 3.2.4.1.

Algoritme 3.2.4.1 : Modulaire machtsverheffing

INPUT: Gehele getallen M en E, $0 \leq M < N$, $0 < E < N$, $E = (e_{t-1} \ldots, e_0)_2$, $e_{t-1} = 1$ en $N \in \mathbb{N}$
OUTPUT: $M^E \mod N$.

1. $A \leftarrow M$
2. **For** i from $t-2$ to 0 **do**:
3. $\quad A \leftarrow AA \mod N$
4. $\quad$ **If** $e_i = 1$ **then**
5. $\quad\quad A \leftarrow AM \mod N$
6. Return A

In Bijlage C geeft Figuur C.3 de architectuur van de state machine. Opnieuw komt de instructie START van de hiërarchisch hogere component die de inversie-bewerking oproept. De overgangen tussen de 4 verschillende states wordt naast het START signaal ook door het DONE-signaal van de Montgomery modulaire vermenigvuldiging (MMM) gestuurd. DONE gaat hoog van zodra het resultaat van de inversie berekend is.

3.2.5 Puntoptelling en puntverdubbeling

Eens de bewerkingen optelling en vermenigvuldiging en de bijhorende omzettingen uit de voorgaande paragrafen volledig geïmplementeerd zijn, is het uitvoeren van de puntoptelling en -verdubbeling zeer eenvoudig. Beide bestaan ze uit een 'finite state machine' met elk 14 states. Deze 14-stappen algoritmes zijn bekomen door de formules voor puntverdubbeling en -optelling te herschrijven, zodanig dat:

- er maximum één modulaire optelling en één modulaire vermenigvuldiging in parallel plaatsvinden
- de data-afhankelijkheden gerespecteerd worden
- het totale aantal stappen minimaal is.

Algoritme 3.2.5.1 geeft voor iedere state de uit te voeren bewerking.

De puntoptelling (links) heeft één ingang P_1 met Z-coördinaat = 1. De reden hiervoor zal duidelijk worden in Paragraaf 3.2.6. De puntoptelling zullen we namelijk enkel gebruiken voor de in de hiërarchie hogere puntvermenigvuldiging. (Zie Algoritme 3.2.6.1.) Eén van de twee ingangen van de puntoptelling zal steeds P zijn, waarvan we weten dat $Z_p = 1$ (want P is een ingang, gegeven in affiene coördinaten). Deze kleine aanpassing vereenvoudigt het algoritme sterk.
Het algoritme voor de puntoptelling werkt niet voor het geval $P_1 = P_2$ of $P_i = \mathcal{O}$, $i = 1, 2$. Dit vormt geen probleem voor onze toepassing. De scalar k uit Algoritme 3.2.6.1 voor de puntvermenigvuldiging is steeds kleiner dan de orde van de elliptische kromme. De tussenresultaten in dit algoritme, en ook de tweede ingang van de puntoptelling, kunnen dus nooit terug het punt P of $\mathcal{O}$ zijn.
De FSM heeft slechts zes tijdelijke registers nodig. De modulaire optellingen/aftrekkingen en verdubbelingen worden uitgevoerd door het modulair optel-blok (Paragraaf

Algoritme 3.2.5.1 : Puntoptelling en -verdubbeling

INPUT: $P_1 = (X_1, Y_1, 1, a)$		INPUT: $P_1 = (X_1, Y_1, Z_1, aZ_1^4)$	
$P_2 = (X_2, Y_2, Z_2, aZ_2^4)$			
OUTPUT: $P_1 + P_2 = P_3$		OUTPUT: $2P_1 = P_3$	
$= (X_3, Y_3, Z_3, aZ_3^4)$.		$= (X_3, Y_3, Z_3, aZ_3^4)$	
1. $T_1 \leftarrow Z_2^2$		1. $T_1 \leftarrow Y_1^2$	$T_2 \leftarrow 2X_1$
2. $T_2 \leftarrow X_1 T_1$		2. $T_3 \leftarrow T_1^2$	$T_2 \leftarrow 2T_2$
3. $T_1 \leftarrow T_1 Z_2$	$T_3 \leftarrow X_2 - T_2$	3. $T_1 \leftarrow T_2 T_1$	$T_3 \leftarrow 2T_3$
4. $T_1 \leftarrow Y_1 T_1$		4. $T_2 \leftarrow X_1^2$	$T_3 \leftarrow 2T_3$
5. $T_4 \leftarrow T_3^2$	$T_5 \leftarrow Y_2 - T_1$	5. $T_4 \leftarrow Y_1 Z_1$	$T_3 \leftarrow 2T_3$
6. $T_2 \leftarrow T_2 T_4$		6. $T_5 \leftarrow T_3(aZ_1^4)$	$T_6 \leftarrow 2T_2$
7. $T_4 \leftarrow T_4 T_3$	$T_6 \leftarrow 2T_2$	7.	$T_2 \leftarrow T_6 + T_2$
8. $Z_3 \leftarrow Z_2 T_3$	$T_6 \leftarrow T_4 + T_6$	8.	$T_2 \leftarrow T_2 + (aZ_1^4)$
9. $T_3 \leftarrow T_5^2$		9. $T_6 \leftarrow T_2^2$	$Z_3 \leftarrow 2T_4$
10. $T_1 \leftarrow T_1 T_4$	$X_3 \leftarrow T_3 - T_6$	10.	$T_4 \leftarrow 2T_1$
11. $T_6 \leftarrow Z_3^2$	$T_2 \leftarrow T_2 - X_3$	11.	$X_3 \leftarrow T_6 - T_4$
12. $T_3 \leftarrow T_5 T_2$		12.	$T_1 \leftarrow T_1 - X_3$
13. $T_6 \leftarrow T_6^2$	$Y_3 \leftarrow T_3 - T_1$	13. $T_2 \leftarrow T_2 T_1$	$aZ_3^4 \leftarrow 2T_5$
14. $aZ_3^4 \leftarrow aT_6$		14.	$Y_3 \leftarrow T_2 - T_3$
15. Return X_3		15. Return X_3	
16. Return Y_3		16. Return Y_3	
17. Return Z_3		17. Return Z_3	

3.2.2); de Montgomery vermenigvuldiger (Paragraaf 3.2.3) neemt de modulaire vermenigvuldigingen en kwadrateringen voor zijn rekening. Omdat het uitvoeren van een modulaire optelling altijd sneller (in minder klokslagen) gaat dan het uitvoeren van een Montgomery vermenigvuldiging, zal de vertraging van één state in het optel-algoritme steeds dezelfde zijn als die van één Montgomery vermenigvuldiging. De totale duur van één elliptische kromme puntoptelling is dus gelijk aan $14T_{MMM}$, met T_{MMM} de duur van één Montgomery vermenigvuldiging. De totale duur van één puntverdubbeling is $8T_{MMM} + 6T_{MAS}$, met T_{MAS} de duur van één modulaire optelling/aftrekking.

3.2.6 Puntvermenigvuldiging

Ook de implementatie van de puntvermenigvuldiging gebeurt met een eenvoudige state-machine. Voor elke bit van de scalar k starten we een iteratie in de *for*-lus. In elke iteratie wordt de puntverdubbeling aangeroepen en enkel indien $k_i = 1$ ook de puntoptelling. In Paragraaf 2.3.4 werd al een algoritme voor deze puntvermenigvuldiging aangehaald. We gebruiken voor de implementatie een licht gewijzigd, zij het equivalent, algoritme (Algoritme 3.2.6.1). De implementatie bestaat uit een FSM met 4 states. Figuur C.4(a) in Bijlage C geeft schematisch de structuur weer.

Dit algoritme zal onveilig blijken voor de tijds- en vermogenaanvallen op elliptische kromme cryptosystemen. Hoofdstuk 4 en volgende komen hier op terug.

Tabel 3.5 geeft een overzicht van de resultaten voor de implementaties van de verschillende bewerkingen. Alle bewerkingen werden uitgevoerd met 160 bit woorden. De nodige chipoppervlakte voor radix 4 is telkens groter dan die voor radix 1. Van zodra de implementatie paste op de FPGA, werd geen aandacht besteed aan verdere optimalisatie van snelheid en oppervlaktegebruik.

Algoritme 3.2.6.1 : Puntvermenigvuldiging (tweede versie)

INPUT: Een punt P, geheel getal $k = (k_{l-1}, k_{l-2}, \ldots, k_0)_2$, $k_{l-1} = 1$
OUTPUT: $Q = [k]P$

1. $Q \leftarrow P$
2. **For** i from $l-2$ to 0 **do**:
3. $\quad Q \leftarrow 2Q$
4. $\quad$ **If** $k_i = 1$ **then**
5. $\quad\quad Q \leftarrow Q + P$
6. Return Q

Tabel 3.5: Overzicht van resultaten voor de implementaties van verschillende bewerkingen. Iedere bewerking gebeurt met 160 bits. Afkortingen: gebruikte aantal bitslices (# b), het equivalente aantal poorten (# p) en de maximale klokfrequentie (max freq)

	radix 1			radix 4		
Implementatie	# b	# p	max freq	# b	# p	max freq
Montg. Verm.	1 684	37 216	51,5 MHz	3 240	52 876	39,9 MHz
Puntopt.	5 820	128 900	20,65 MHz	7 372	144 340	29,2 MHz
Puntverd.	5 490	121 329		7 042	137 320	41MHz
Puntverm.	6 912	152 755		8 728	164 724	31,1 MHz

3.2.7 Hoofd controller

De hoofd controller zit helemaal bovenaan in de hiërarchie van de hardware blokken. Hij heeft drie belangrijke taken:

- Conversie van normale naar Montomery voorstelling van de ingangen en van Montgomery naar normaal van het resultaat.
- Conversie van affiene naar projectieve coördinaten van het inganspunt (triviaal) en van affien naar projectief voor het resultaat.
- Aanroepen van het puntvermenigvuldigingsalgoritme en hierbij de juiste argumenten meegeven.

Dit alles gebeurt in een state-machine met vijf states. Figuur C.4(b) in Bijlage C geeft deze weer.

Het beschreven ontwerp in deze paragraaf bouwt verder op de implementatie gepubliceerd in [29].

3.3 Implementatie van aritmetische bewerkingen in $GF(2^n)$

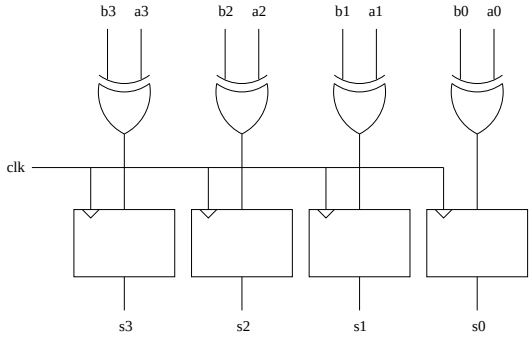
De implementatie van de puntvermenigvuldiging in $GF(2^n)$ is in verschillende opzichten gelijkaardig aan de implementatie in $GF(p)$. Daarom verwijzen we in dit gedeelte meermaals naar Paragraaf 3.2.

Zoals reeds aangehaald in Paragraaf 2.2.1 stellen we de veldelementen van $GF(2^n)$ voor als veeltermen in x. De modulus is dan een irreduceerbare veelterm $p(x)$ van graad n. De coëfficiënten van de gebruikte veeltermen zijn elementen van $GF(2)$, m.a.w. de coëfficiënten zijn steeds gelijk aan 0 of 1, het zijn *bits*. Deze eigenschap maakt aritmetische bewerkingen in $GF(2^n)$ uiterst geschikt voor implementatie in hardware. De bits kunnen immers worden bijgehouden in vectoren. De meest beduidende bit (Most Significant Bit, MSB) is de meest linkse bit van de vector en komt overeen met de veeltermcoëfficiënt bij de hoogste macht van x. De minst beduidende bit (Least Significant Bit, LSB) is de meest rechtse, het is de constante term van de veelterm. De lengte van een vector is steeds 1 hoger dan de graad van de bijhorende veelterm.

3.3.1 Modulo optellen en aftrekken

De optelling/aftrekking van twee veeltermen in $GF(2^n)$ komt overeen met de optelling/aftrekking van de coëfficiënten in $GF(2)$. Er is dus geen onderscheid tussen optellen en aftrekken in $GF(2^n)$; de bewerking is steeds een simpele bitgewijze XOR. Het resultaat van een optelling/aftrekking is bijgevolg reeds na één klokpuls beschikbaar. Figuur 3.13 toont een schematische voorstelling van de geklokte optelling/aftrekking in $GF(2^4)$:

$$(s_3x^3 + s_2x^2 + s_1x + s_0) = (a_3x^3 + a_2x^2 + a_1x + a_0) + (b_3x^3 + b_2x^2 + b_1x + b_0)$$



Figuur 3.13: Optelling/aftrekking in $GF(2^4)$. Op de stijgende klokflank worden de uitgangen van de XOR-poorten in de vier registers ingelezen.

3.3.2 Montgomery vermenigvuldiging

Net zoals in oneven velden is het ook in $GF(2^n)$ efficiënter om gebruik te maken van de Montgomery vermenigvuldiging:

$$c(x) = a(x) * b(x) * r(x)^{-1} \bmod p(x)$$

Hierin zijn de ingangen en de uitgang veeltermen van graad $n-1$:

$$a(x) = a_{n-1}x^{n-1} + \ldots + a_2x^2 + a_1x + a_0$$

$$b(x) = b_{n-1}x^{n-1} + \ldots + b_2x^2 + b_1x + b_0$$

$$c(x) = c_{n-1}x^{n-1} + \ldots + c_2x^2 + c_1x + c_0$$

De modulus is een irreduceerbare veelterm van graad n:

$$p(x) = p_nx^n + p_{n-1}x^{n-1} + \ldots + p_2x^2 + p_1x + p_0$$

De voorwaarde waaraan $r(x)$ moet voldoen is $ggd(r(x), p(x)) = 1$. Vermits $p(x)$ irreduceerbaar is en bijgevolg niet deelbaar door x, is x^n een goede keuze voor $r(x)$.

Afhankelijk van de vereiste snelheid en oppervlakte zijn er verschillende algoritmes mogelijk om de Montgomery vermenigvuldiging te realiseren. In deze paragraaf komt eerst de implementatie van het *bit-level algoritme* aan bod, waarin de evaluatie van één van de ingangsveeltermen bit per bit gebeurt. Daarna volgt een uiteenzetting van het *word-level algoritme*, dat w bits tegelijk verwerkt.

Bit-level algoritme

Algoritme 3.3.2.1 (gebaseerd op [7]) gebruikt in elke lusherhaling één bit van $a(x)$ terwijl $b(x)$ vanaf de eerste herhaling in zijn geheel nodig is. Een eerste implementatie van

Algoritme 3.3.2.1 : Bit-level algoritme voor Montgomery vermenigvuldiging

---

INPUT: $a(x), b(x), p(x)$
OUTPUT: $c(x) = a(x) * b(x) * x^{-n} \bmod p(x)$
1. $c(x) \leftarrow 0$
2. **For** i from 0 to $(n-1)$ **do**:
   2.1 $m(x) \leftarrow c(x) + a_i * b(x)$
   2.2 $c(x) \leftarrow m(x) + m_0 * p(x)$
   2.3 $c(x) \leftarrow c(x)/x$
3. Return $c(x)$

---

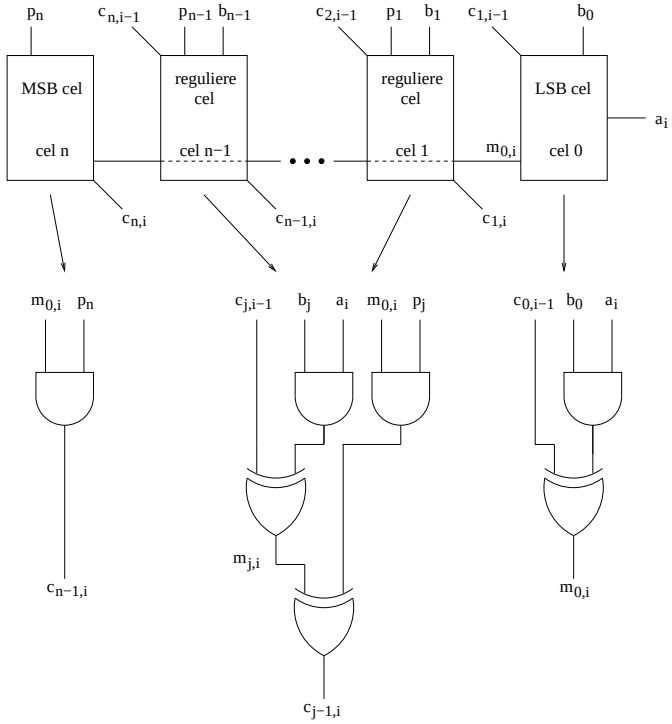
de Montgomery vermenigvuldiging steunt op dit bit-level algoritme. De implementatie bestaat uit een systolische reeks (Figuur 3.14) en een finite state machine (Figuur C.5 in Bijlage C).

De bewerkingen in $GF(2^n)$ op de veeltermen in het algoritme kunnen herleid worden tot bewerkingen in $GF(2)$ op de veeltermcoëfficiënten:

- Een optelling van twee veeltermen is een bitgewijze XOR (zoals in Paragraaf 3.3.1).
- Een vermenigvuldiging van een veelterm met een constante is een bitgewijze AND van de coëfficiënten van de veelterm met deze constante. Zo geldt bv. in stap 2.1 van het algoritme:

$$a_i * b(x) = (a_i * b_{n-1})x^{n-1} + \ldots + (a_i * b_2)x^2 + (a_i * b_1)x + (a_i * b_0)$$

- Een deling door x is een schuifbewerking naar rechts over één positie van de veelterm-coëfficiënten, waarbij de MSB op 0 wordt gezet (de graad van de veelterm wordt 1 lager).



Figuur 3.14: Bit-level algoritme systolische reeks

Een analyse van de i-de herhaling van stap 2 in het algoritme verklaart de inhoud van de $n+1$ cellen in de systolische reeks.
In **stap 2.1** berekent elke cel een coëfficiënt van $m(x)$. Vermits $m(x)$ een veelterm is met n coëfficiënten ($m(x)$ is van graad $n-1$), is er geen MSB cel voor deze stap nodig. In cel j van de systolische reeks (met $0 \leq j \leq n-1$) gebeurt de bewerking

$$m_j = c_{j,i-1} + a_i * b_j.$$

Hierin is $c_{j,i-1}$ de j-de coëfficiënt van $c(x)$ uit de vorige herhaling (de $(j-1)$-de herhaling).
In **stap 2.2** voert elke cel een AND uit van $m_{0,i}$, de constante term in $m(x)$, met een coëfficiënt van $p(x)$. Het resultaat is een veelterm van graad n. Die veelterm wordt bitgewijs ge-XOR-d met $m(x)$ uit stap 2.1. Op deze manier berekent cel j de j-de coëfficiënt van $c(x)$ in stap 2.2. Omdat de graad van $m(x)$ één lager is dan de graad van $m_0 * p(x)$, geldt

$$c_n = 0 + m_0 * p_n = m_0 * p_n.$$

Voor de overige coëfficiënten van $c(x)$ is

$$c_j = m_j + m_0 * p_j.$$

Opdat $p(x)$ irreduceerbaar zou zijn, moet $p_0 = 1$. Hieruit volgt

$$c_0 = m_0 + m_0 * p_0 = m_0 + m_0 = 0.$$

Omdat deze bewering altijd geldt, is er in de LSB cel geen hardware voorzien voor de uitvoering van stap 2.2. De veelterm die in deze stap ontstaat, is dus van graad n en

heeft geen constante term:

$$c(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_2 x^2 + c_1 x$$

Stap 2.3 voert een deling van deze veelterm door x uit. Het resultaat doet dienst als ingang van de volgende herhaling (de $(i+1)$-de herhaling) van stap 2. De deling door x wordt gerealiseerd door de uitgang van cel j in herhaling i aan te leggen aan de ingang van cel $j-1$ in herhaling $i+1$.

Elke klokpuls wordt er een herhaling van stap 2 uitgevoerd, waardoor een Montgomery vermenigvuldiging n klokpulsen duurt.

Een schuifregister *a_temp* zorgt ervoor dat de systolische reeks in elke klokcyclus de juiste coëfficiënt van $a(x)$ krijgt.
Figuur C.5 in Bijlage C toont dat *a_temp* na het START signaal gevuld wordt met de coëfficiënten van $a(x)$. De a_i-ingang van de systolische reeks is verbonden met de LSB van het schuifregister. Vlak na het START signaal ligt a_0 dus aan de ingang van de systolische reeks. Op dit moment krijgt de tellervector *counter* de waarde $(10\ldots0)$ toegekend. De lengte van deze vector is n.
De FSM voert daarna bij elke klokpuls een controle uit op de LSB van *counter*. Als deze niet 1 is, gebeurt er een schuifbewerking naar rechts van zowel *counter* als *a_temp*. Ook zullen de coëfficiënten van $c(x)$, gevormd in de vorige klokcyclus, aangelegd worden als nieuwe ingangen van de systolische reeks. Deze toekenning wordt in de FSM voorgesteld door $c_{in} \leftarrow c_{uit}$.
Als de controle van de FSM uitwijst dat de LSB van counter gelijk is aan 1, betekent dit dat alle coëfficiënten van *a_temp* geëvalueerd zijn. De volgende $c(x)$-uitgang van de systolische reeks moet dan niet meer worden teruggevoerd naar de ingang, maar vormt de uiteindelijke uitgang van het algoritme. In de FSM wordt dit genoteerd als $c \leftarrow c_{uit}$.
Een VALID signaal geeft aan dat het resultaat van de Montgomery vermenigvuldiging klaar is.
Een speciale uitleesmethode zoals in Figuur 3.11 van Paragraaf 3.2.3 is hier niet nodig omdat alle bits van het resultaat gelijktijdig geldig zijn.

Word-level algoritme

In het word-level algoritme (Algoritme 3.3.2.2) gebeurt de evaluatie van $a(x)$ in *digits* van w bits. Een digit is een veelterm van graad $w-1$, w wordt daarom de *digitlengte* of *radix* genoemd. Volgende notaties zijn van toepassing op het algoritme:

Algoritme 3.3.2.2 : Word-level algoritme voor Montgomery vermenigvuldiging

---

INPUT: $a(x), b(x), p(x), P_0^{-1}(x)$
OUTPUT: $c(x) = a(x) * b(x) * x^{-n} \bmod p(x)$
1. $c(x) \leftarrow 0$
2. **For** i from 0 to $(s-1)$ **do**:
   2.1 $d(x) \leftarrow c(x) + A_i(x) * b(x)$
   2.2 $M_0(x) \leftarrow D_0(x) * P_0^{-1}(x) \bmod x^w$
   2.3 $c(x) \leftarrow d(x) + M_0(x) * p(x)$
   2.4 $c(x) \leftarrow c(x)/x^w$
3. Return $c(x)$

- Een hoofdletter is een digit, afgeleid van de veelterm gegeven door de overeenkomstige kleine letter. De coëfficiënten worden bepaald door het subscript:

$$A_i(x) = \sum_{j=0}^{w-1} a_{iw+j}x^j$$

Voor $w = 4$ geldt bijvoorbeeld

$$\begin{aligned} A_0(x) &= a_3x^3 + a_2x^2 + a_1x + a_0 \\ A_1(x) &= a_7x^3 + a_6x^2 + a_5x + a_4 \\ &\dots \end{aligned}$$

- $P_0^{-1}(x)$ is de inverse van $P_0(x)$ modulo x^w:

$$P_0^{-1}(x) * P_0(x) \bmod x^w = 1$$

- s is het aantal digits in een veelterm van graad $n-1$, waarbij n is steeds een veelvoud is van w: $s = \frac{n}{w}$

Verderop in de tekst wordt deze eigen gekozen notatie gebruikt:

- Een hoofdletter met een accent is een digit waarvan de hoogste graadsterm is weggelaten:

$$A'_i(x) = \sum_{j=0}^{w-2} a_{iw+j}x^j$$

Voor $w = 4$ geldt bijvoorbeeld

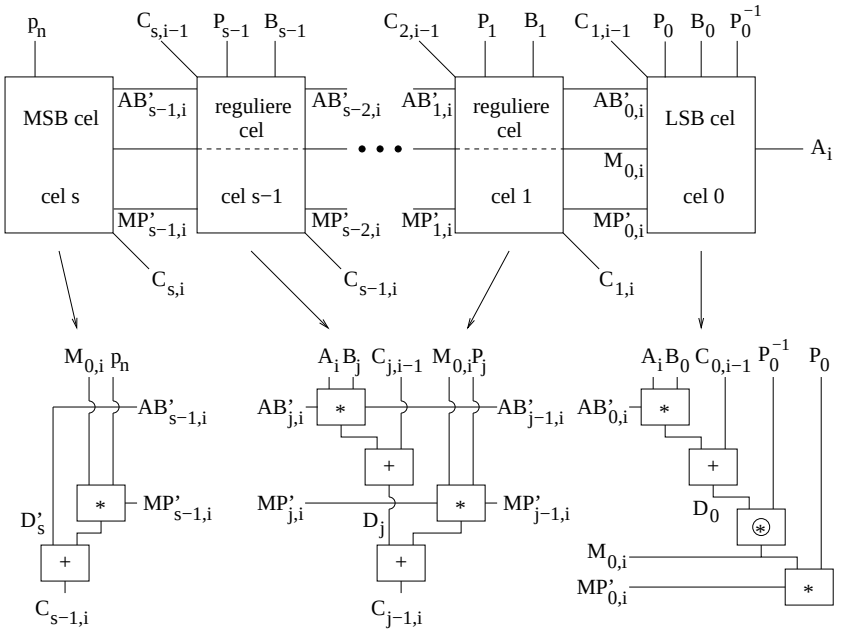
$$\begin{aligned} A'_0(x) &= a_2x^2 + a_1x + a_0 \\ A'_1(x) &= a_6x^2 + a_5x + a_4 \\ &\dots \end{aligned}$$

De implementatie van het word-level algoritme maakt opnieuw gebruik van een systolische reeks. Figuur 3.15 toont de gebruikte hardware blokjes en Figuur C.6 van Bijlage C de finite state machine.

In het algoritme komen de volgende vier bewerkingen voor:

- Een optelling van 2 veeltermen is een bitgewijze XOR van de coëfficiënten (zoals in Paragraaf 3.3.1).
- Een vermenigvuldiging van 2 veeltermen bestaat uit verschillende XOR- en AND-bewerkingen op de coëfficiënten van de veeltermen. Bijvoorbeeld:

$$\begin{aligned} &(a_3x^3 + a_2x^2 + a_1x + a_0) * (b_3x^3 + b_2x^2 + b_1x + b_0) \\ &= \\ &(a_3 * b_3)x^6 + \\ &(a_3 * b_2 + a_2 * b_3)x^5 + \\ &(a_3 * b_1 + a_2 * b_2 + a_1 * b_3)x^4 + \\ &(a_3 * b_0 + a_2 * b_1 + a_1 * b_2 + a_0 * b_3)x^3 + \\ &(a_2 * b_0 + a_1 * b_1 + a_0 * b_2)x^2 + \\ &(a_1 * b_0 + a_0 * b_1)x + \\ &(a_0 * b_0) \end{aligned}$$



Figuur 3.15: Word-level algoritme systolische reeks

Het resultaat van de vermenigvuldiging van een veelterm van graad $k-1$ (met k coëfficiënten) met een veelterm van graad $l-1$ (met l coëfficiënten) is een veelterm van graad $k+l-2$ (met $k+l-1$ coëfficiënten).

- Een vermenigvuldiging van 2 veeltermen modulo x^w (aangeduid met $\circledast$ in Figuur 3.15) is een gewone vermenigvuldiging van 2 veeltermen waarbij alle termen met een macht van x groter dan $w-1$ wegvallen. Bijvoorbeeld:

$$
\begin{aligned}
&(a_3x^3 + a_2x^2 + a_1x + a_0) * (b_3x^3 + b_2x^2 + b_1x + b_0) \bmod x^4 \\
&= \\
&(a_3 * b_0 + a_2 * b_1 + a_1 * b_2 + a_0 * b_3)x^3 + \\
&(a_2 * b_0 + a_1 * b_1 + a_0 * b_2)x^2 + \\
&(a_1 * b_0 + a_0 * b_1)x + \\
&(a_0 * b_0)
\end{aligned}
$$

- Een deling door x^w is een schuifbewerking van de coëfficiënten naar rechts over w plaatsen. Aan de linkerkant worden w nullen ingevoerd (de graad van de veelterm vermindert met w).

Een analyse van de i-de herhaling van stap 2 in Algoritme 3.3.2.2 laat toe om de werking van de $s+1$ cellen in de systolische reeks beter te begrijpen.
In **stap 2.1** gebeurt eerst een vermenigvuldiging van $A_i(x)$, de i-de digit van $a(x)$, met $b(x)$. Het resultaat, een veelterm met $n+w-1$ coëfficiënten, wordt opgeteld bij $c(x)$, een veelterm met n coëfficiënten. De som is $d(x)$, een veelterm met $n+w-1$ coëfficiënten. $d(x)$ kan worden opgesplitst in s veeltermen met w coëfficiënten (digits) en één veelterm met $w-1$ coëfficiënten:

$$d(x) = D'_s(x)x^{sw} + D_{s-1}(x)x^{(s-1)w} + \ldots + D_j(x)x^{jw} + \ldots + D_0(x)$$

Cel j van de systolische reeks (met $0 \leq j \leq s-1$) berekent $D_j(x)$ en cel s (de MSB cel) $D'_s(x)$. Dit zijn de bewerkingen die plaatsvinden in de cellen:

$$\begin{aligned}
D_0(x) &= C_{0,i-1}(x) + A_i(x) * B_0(x) \bmod x^w \\
D_1(x) &= C_{1,i-1}(x) + A_i(x) * B_1(x) \bmod x^w + AB'_0(x) \\
&\dots \\
D_j(x) &= C_{j,i-1}(x) + A_i(x) * B_j(x) \bmod x^w + AB'_{j-1}(x) \\
&\dots \\
D_{s-1}(x) &= C_{s-1,i-1}(x) + A_i(x) * B_{s-1}(x) \bmod x^w + AB'_{s-2}(x) \\
D'_s(x) &= AB'_{s-1}(x)
\end{aligned}$$

Hierin is $C_{j,i-1}$ een digit van $c(x)$ uit de vorige herhaling (de $(i-1)$-de herhaling) van stap 2.
De minst beduidende digit van het product $A_i(x) * B_j(x)$ wordt in cel j verwerkt. De $w-1$ overige bits worden in $AB'_j(x)$ geplaatst. Cel $j+1$ telt $AB'_j(x)$ op bij de minst beduidende digit van $A_i(x) * B_{j+1}(x)$. Omdat cel s (de MSB cel) de $w-1$ meest beduidende bits van $d(x)$ moet vormen, moet deze cel geen bits van $a(x)$, $b(x)$ en $c(x)$ verwerken. $D'_s(x)$ is daarom rechtstreeks gelijk aan AB'_{s-1}.
De uitvoering van **stap 2.2** vindt enkel plaats in de LSB cel. Hier is $D_0(x)$ beschikbaar uit stap 2.1 en wordt $P_0^{-1}(x)$ als ingang aangelegd. De modulo bewerking zorgt ervoor dat de overige cellen niet nodig zijn bij de berekening van $M_0(x)$.
In **stap 2.3** voert de systolische reeks een vermenigvuldiging uit van $M_0(x)$ met $p(x)$. Het resultaat, een veelterm van $n+w$ bits, wordt opgeteld bij $d(x)$, bestaande uit $n+w-1$ bits. De som is de $(n+w)$-bit lange veelterm $c(x)$. De opsplitsing van $c(x)$ in $s+1$ digits ziet er als volgt uit:

$$c(x) = C_s(x)x^{sw} + C_{s-1}(x)x^{(s-1)w} + \ldots + C_j(x)x^{jw} + \ldots + C_0(x)$$

Cel j van de systolische reeks (met $0 \leq j \leq s$) berekent $C_j(x)$:

$$\begin{aligned}
C_0(x) &= D_0(x) + M_0(x) * P_0(x) \bmod x^w \\
C_1(x) &= D_1(x) + M_0(x) * P_1(x) \bmod x^w + MP'_0(x) \\
&\dots \\
C_j(x) &= D_j(x) + M_0(x) * P_j(x) \bmod x^w + MP'_{j-1}(x) \\
&\dots \\
C_{s-1}(x) &= D_{s-1}(x) + M_0(x) * P_{s-1}(x) \bmod x^w + MP'_{s-2}(x) \\
C_s(x) &= D'_s(x) + M_0(x) * p_n + MP'_{s-1}(x)
\end{aligned}$$

Op dezelfde manier als in stap 2.1 geeft elke cel (behalve de MSB cel) de $w-1$ meest beduidende bits van het product $M_0(x) * P_j(x)$ door aan de volgende cel. De MSB cel moet nog 1 bit van $p(x)$ vermenigvuldigen met $M_0(x)$. In deze cel is de vermenigvuldiging dus een eenvoudige bitgewijze AND van p_n met de coëfficiënten van $M_0(x)$.
Het uitschrijven van de toekenning aan $C_0(x)$ maakt duidelijk dat de LSB cel in stap

2.3 enkel $MP'_0(x)$ moet genereren en niet $C_0(x)$:

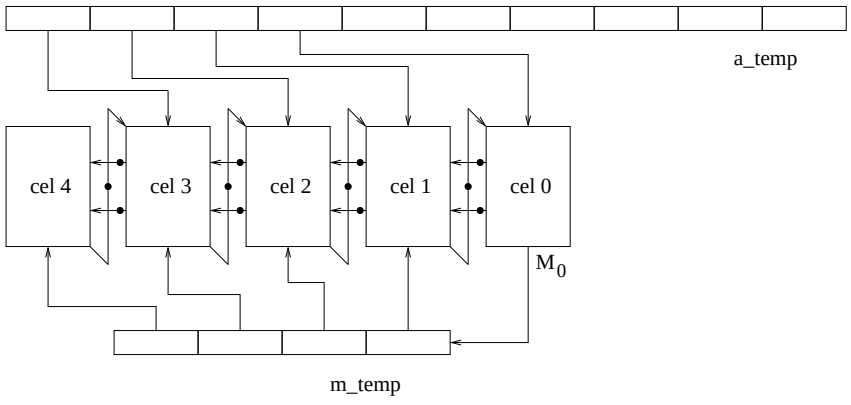
$$\begin{aligned} C_0(x) &= D_0(x) + M_0(x) * P_0(x) \bmod x^w \\ &= D_0(x) + D_0(x) * P_0^{-1}(x) * P_0(x) \bmod x^w \\ &= D_0(x) + D_0(x) \\ &= 0 \end{aligned}$$

$c(x)$ is dus een veelterm van graad $n + w - 1$ waarvan de w minst beduidende bits 0 zijn:

$$c(x) = c_{n+w-1}x^{n+w-1} + \ldots + c_w x^w$$

Stap 2.4 deelt deze veelterm door x^w. De implementatie realiseert dit door de uitgangsdigit van cel j in herhaling i aan te leggen aan de ingang van cel $j-1$ in herhaling $i+1$.

Indien de uitvoering van elke herhaling van stap 2 binnen 1 klokcyclus zou plaatsvinden, zou de vertraging van $C_{0,i-1}$ naar $C_{s-1,i}$ veel te groot worden. De MSB cel moet dan immers wachten op AB'_{s-1} en MP'_{s-1} van cel $s-1$, terwijl die eerst AB'_{s-2} en MP'_{s-2} van cel $s-2$ moet ontvangen, ... De minimaal vereiste klokperiode zou in dit geval ook groter worden bij stijgende n. Omdat het de bedoeling is om de klokperiode niet te groot en onafhankelijk van n te maken, bevat de systolische reeks geklokte registers tussen de cellen. De klokperiode is nu enkel nog afhankelijk van de digitlengte w, omdat die de hoeveelheid hardware binnen 1 cel bepaalt.
Het plaatsen van de tussenliggende registers zorgt ervoor dat elke cel van de systolische reeks een andere digit van $a(x)$ verwerkt binnen dezelfde klokcyclus. Het aanleggen van de juiste digits van $a(x)$ aan de juiste cellen gebeurt met een schuifregister *a_temp*. Op dezelfde manier wordt ook $M_0(x)$ in een schuifregister *m_temp* geplaatst zodat elke cel de juiste versie van $M_0(x)$ krijgt. Aan de hand van Figuur C.6 (Bijlage C) en Figuur 3.16 illustreert Voorbeeld 3.3.2.1 het schuifmechanisme in $GF(2^{16})$ met $w = 4$. De zwarte bolletjes in Figuur 3.16 zijn registers.



Figuur 3.16: Word-level algoritme schuifmechanisme in $GF(2^{16})$

Voorbeeld 3.3.2.1 (t) : Schuifmechanisme in $GF(2^{16})$ met $w = 4$ en $s = 4$

- *RST: alle registers* $\leftarrow 0$

- *START (klokpuls 1):*

  $a\_temp \leftarrow a\_start = (0000\ 0000\ 0000\ A_0\ 0000\ A_1\ 0000\ A_2\ 0000\ A_3)$

- *klokpuls 2:*

  $a\_temp \leftarrow (0000\ 0000\ A_0\ 0000\ A_1\ 0000\ A_2\ 0000\ A_3\ 0000)$
  $m\_temp \leftarrow (0000\ 0000\ 0000\ M_{0,0})$

- *klokpuls 3:*

  $a\_temp \leftarrow (0000\ A_0\ 0000\ A_1\ 0000\ A_2\ 0000\ A_3\ 0000\ 0000)$
  $m\_temp \leftarrow (0000\ 0000\ M_{0,0}\ XXXX)$

- *klokpuls 4:*

  $a\_temp \leftarrow (A_0\ 0000\ A_1\ 0000\ A_2\ 0000\ A_3\ 0000\ 0000\ 0000)$
  $m\_temp \leftarrow (0000\ M_{0,0}\ XXXX\ M_{0,1})$

- ...

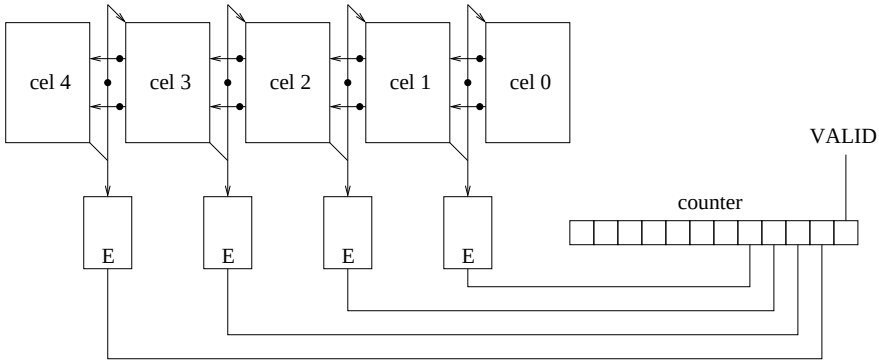
- *klokpuls 10:*

  $a\_temp \leftarrow (A_3\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)$
  $m\_temp \leftarrow (XXXX\ M_{0,3}\ XXXX\ XXXX)$

- *klokpuls 11:*

  $a\_temp \leftarrow (0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)$
  $m\_temp \leftarrow (M_{0,3}\ XXXX\ XXXX\ XXXX)$

Op klokpuls 1, de eerste stijgende klokflank na het START signaal, krijgt cel 0 $A_0(x)$ als ingang. In klokcyclus 1 (de klokcyclus tussen klokpuls 1 en 2) verwerkt cel 0 $A_0(x)$. $AB'_0(x)$, $MP'_0(x)$ en $M_{0,0}(x)$ zijn dan klaar op klokpuls 2, wanneer cel 1 start met de evaluatie van $A_0(x)$. Cel 0 kan op dat moment echter nog niet starten met $A_1(x)$, omdat de uitgang $C_0(x)$ van cel 1 eerst klaar moet zijn. In klokcyclus 3 maakt cel 0 gebruik van $C_0(x)$ en $A_1(x)$. Ondertussen evalueert cel 2 $A_0(x)$ en wacht cel 1 op $M_{0,1}(x)$, $AB'_0(x)$ en $MP'_0(x)$ van cel 0 en $C_1(x)$ van cel 2. In klokcyclus 4 zullen cel 3 en cel 1 respectievelijk $A_0(x)$ en $A_1(x)$ verwerken, ...
X duidt op een waarde die niet relevant is voor de werking van het algoritme. Zo maakt het bijvoorbeeld op klokpuls 3 niet uit welke waarde er aan de M_0-uitgang van cel 0 verschijnt, omdat cel 0 op dat moment nog niet bezig is met de verwerking van $A_1(x)$.
Omdat de digits van de uiteindelijke uitgang $c(x)$ niet allemaal op hetzelfde moment geldig zijn, is er een uitleesmechanisme voorzien dat hiermee rekening houdt. Figuur 3.17 toont de s uitgangsregisters (elk w bits breed) van de systolische reeks. Er wordt enkel een waarde naar een register geschreven als de enable ingang E gelijk is aan 1. De laatste bits van het schuifregister *counter* zijn verbonden met de *enable* ingangen. Op die manier worden enkel geldige resultaten naar de uitgangsregisters geschreven. *counter* zorgt er eveneens voor dat er na het juiste aantal klokpulsen een VALID signaal wordt uitgestuurd.
Het totaal aantal klokpulsen dat nodig is voor één Montgomery vermenigvuldiging is gelijk aan $3 * s$. Voor $GF(2^{16})$ en $w = 4$ duurt het 12 klokpulsen eer de uitgang geldig is: de 11 klokpulsen uit Voorbeeld 3.3.2.1 plus één klokpuls om naar het meest beduidende uitgangsregister te schrijven.



Figuur 3.17: Word-level algoritme uitleesmechanisme

Tabel 3.6 geeft een samenvatting van de eigenschappen van de gerealiseerde 160 bit vermenigvuldiger. De tabel toont dat voor deze radices de snelheid van één vermenigvuldiging stijgt, naarmate w stijgt. Het nadeel is echter dat de oppervlakte in dit geval ook groter wordt.

Tabel 3.6: Vergelijkend overzicht radix 1, radix 4 en radix 8 voor 160x160 vermenigvuldiging in $GF(2^n)$

	radix 1	radix 4	radix 8
Totaal aantal klokslagen ($\sharp$)	160	120	60
Periode klok (P)	10,375 ns	11,742 ns	13,040 ns
Totale vertraging ($\sharp * P$)	1,66 μ s	1,41 μ s	0,782 μ s
Oppervlakte (aantal gates)	1427	2018	3691

3.3.3 Omzettingen

Normale versus Montgomery voorstelling

Om alle vermenigvuldigingen te kunnen vervangen door Montgomery vermenigvuldigingen, is er eerst een omzetting nodig van de ingangsveeltermen naar de Montgomery voorstelling. Net zoals bij de omzetting in oneven velden (Paragraaf 3.2.4), is er ook hier een vermenigvuldiging met $r(x)$ nodig:

$$a(x)_{Montgomery} = a(x)r(x) \bmod p(x) = a(x)x^n \bmod p(x)$$

Dit komt overeen met een Montgomery vermenigvuldiging met $r(x)^2$:

$$a(x)_{Montgomery} = a(x)r(x)^2r(x)^{-1} \bmod p(x) = a(x)x^{2n}x^{-n} \bmod p(x)$$

Het terug omzetten van het uiteindelijke resultaat naar de normale voorstelling is een Montgomery vermenigvuldiging met 1:

$$a(x) = a(x)_{Montgomery}r(x)^{-1} \bmod p(x)$$

Affiene versus projectieve coördinaten

Om inversies te vermijden is het nuttig om van affiene coördinaten over te stappen op projectieve. Paragraaf 3.2.4 maakt gebruik van een voorstelling met 4 coördinaten. De voorstelling in $GF(2^n)$ heeft 3 coördinaten X, Y en Z. Zij voldoen aan volgende omzettingsregels tussen affiene en projectieve coördinaten:

$$\text{affien} : (x, y) \rightarrow \text{projectief} : (X, Y, Z) = (x, y, 1)$$
$$\text{projectief} : (X, Y, Z) \rightarrow \text{affien} : (x, y) = (\frac{X}{Z}, \frac{Y}{Z^2})$$

De omzetting van projectieve naar affiene coördinaten vraagt een inversie van Z. In $GF(2^n)$ geldt als gevolg van het theorema van Fermat [21]

$$a(x)^{-1} = a(x)^{2^n-2} \bmod p(x).$$

Op basis van deze gelijkheid kan een inversie uitgevoerd worden door herhaaldelijk te kwadrateren en vermenigvuldigen. Algoritme 3.3.3.1 realiseert de inversie op die manier. De implementatie van het algoritme maakt gebruik van de Montgomery ver-

Algoritme 3.3.3.1 : Algoritme voor modulaire inversie

INPUT: $a(x)$
OUTPUT: $b(x)$ zodat $a(x) * b(x) \bmod p(x) = 1$
1. $b(x) \leftarrow a(x)$
2. **For** i from 1 to $(n-2)$ **do**:
   2.1 $b(x) \leftarrow b(x) * b(x) \bmod p(x)$
   2.2 $b(x) \leftarrow a(x) * b(x) \bmod p(x)$
3. $b(x) \leftarrow a(x) * b(x) \bmod p(x)$
4. Return $b(x)$

---

menigvuldiger in Paragraaf 3.3.2. Het resultaat van de inversie zal dus in Montgomery voorstelling staan. Daarom zal omzetting van projectieve naar affiene coördinaten plaatsvinden vóór de omzetting van Montgomery naar normale voorstelling.

3.3.4 Puntoptelling en puntverdubbeling

Voor de puntoptelling en -verdubbeling wordt er gebruik gemaakt van projectieve coördinaten zoals in Paragraaf 3.3.3. Deze coördinaten stemmen niet overeen met de in Paragraaf 2.3.1 voorgestelde projectieve coördinaten. De Weierstrass vergelijking in $GF(2^n)$ wordt nu

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4.$$

Publicatie [19] geeft zowel de algoritmes voor de puntoptelling en de puntverdubbeling in deze projectieve coördinaten als de praktische realisatie ervan. Algoritme 3.3.4.1 is hierop gebaseerd en beschrijft onze puntoptelling en -verdubbeling. T_1, T_2, T_3 en T_4 zijn tijdelijke registers. Het naast elkaar plaatsen van twee bewerkingen wijst op een parallelle uitvoering. Een modulaire optelling en een Montgomery vermenigvuldiging kunnen parallel berekend worden, omdat ze door twee verschillende hardware blokken worden uitgevoerd. Figuur C.7 in Bijlage C toont de finite state machines die de puntoptelling en de puntverdubbeling uitvoeren.

Algoritme 3.3.4.1 : Puntoptelling en -verdubbeling

INPUT: 2 punten op de curve:
$P_1(X_1, Y_1, Z_1), P_2(X_2, Y_2, 1)$
coëfficiënt van de curve:
a
OUTPUT: $P_3(X_3, Y_3, Z_3)$
$= P_1 + P_2$

1. $T_1 \leftarrow Z_1 * Z_1$	
2. $T_2 \leftarrow Y_2 * T_1$	
3. $T_3 \leftarrow X_2 * Z_1$	$T_2 \leftarrow T_2 + Y_1$
4. $T_1 \leftarrow a * T_1$	$T_3 \leftarrow T_3 + X_1$
5. $T_4 \leftarrow Z_1 * T_3$	
6. $T_3 \leftarrow T_3 * T_3$	$T_1 \leftarrow T_4 + T_1$
7. $T_1 \leftarrow T_3 * T_1$	
8. $Z_3 \leftarrow T_4 * T_4$	
9. $T_4 \leftarrow T_2 * T_4$	
10. $T_2 \leftarrow T_2 * T_2$	$T_1 \leftarrow T_1 + T_4$
11. $T_1 \leftarrow X_2 * Z_3$	$X_3 \leftarrow T_2 + T_1$
12. $T_2 \leftarrow Y_2 * Z_3$	$T_1 \leftarrow X_3 + T_1$
13. $T_1 \leftarrow T_4 * T_1$	$T_2 \leftarrow X_3 + T_2$
14. $T_3 \leftarrow Z_3 * T_2$	
15. $Y_3 \leftarrow T_1 + T_3$	
16. Return X_3	
17. Return Y_3	
18. Return Z_3	

INPUT: een punt op de curve:
$P_1(X_1, Y_1, Z_1)$
coëfficiënten van de curve:
a en b
OUTPUT: $P_3(X_3, Y_3, Z_3)$
$= [2]P_1$

1. $T_1 \leftarrow Z_1 * Z_1$	
2. $T_2 \leftarrow X_1 * X_1$	
3. $Z_3 \leftarrow T_1 * T_2$	
4. $T_1 \leftarrow T_1 * T_1$	
5. $T_2 \leftarrow T_2 * T_2$	
6. $T_1 \leftarrow b * T_1$	
7. $T_2 \leftarrow Y_1 * Y_1$	$X_3 \leftarrow T_1 + T_2$
8. $T_3 \leftarrow a * Z_3$	$T_2 \leftarrow T_2 + T_1$
9. $T_1 \leftarrow T_1 * Z_3$	$T_3 \leftarrow T_3 + T_2$
10. $T_3 \leftarrow X_3 * T_3$	
11. $Y_3 \leftarrow T_3 + T_1$	
12. Return X_3	
13. Return Y_3	
14. Return Z_3	

3.3.5 Puntvermenigvuldiging

Het algoritme voor de puntvermenigvuldiging in $GF(2^n)$ is hetzelfde als in $GF(p)$. Daarom verwijzen we voor de puntvermenigvuldiging naar Algoritme 3.2.6.1.

Tabel 3.7 geeft een overzicht van de resultaten voor de implementaties van de verschillende bewerkingen. Alle bewerkingen werden uitgevoerd met 160 bit woorden. De nodige chipoppervlakte voor radix 4 is telkens groter dan die voor radix 1. Van zodra de implementatie paste op de FPGA, werd geen aandacht besteed aan verdere optimalisatie van snelheid en oppervlaktegebruik.

Tabel 3.7: Overzicht van resultaten voor de implementaties van verschillende bewerkingen. Iedere bewerking gebeurt met 160 bits. Afkortingen: gebruikte aantal bitslices (# b), het equivalente aantal poorten (# p) en de maximale klokfrequentie (max freq)

	radix 1			radix 4		
Implementatie	# b	# p	max freq	# b	# p	max freq
Montg. Verm.	473	12 150	63,7 MHz	1 064	26 113	60,0 MHz
Puntopt.	3 004	62 202	25,5 MHz	3 596	73 635	50,3 MHz
Puntverd.	2 138	47 618	38,4 MHz	2 725	59 051	54,2 MHz
Puntverm.	5 695	117 358	39,3 MHz	6 282	129 454	31,6 MHz

3.3.6 Hoofd controller

De hoofd controller in $GF(2^n)$ is dezelfde als in $GF(p)$ (Paragraaf 3.2.7).

3.4 Besluit

Dit hoofdstuk geeft de uitwerking van de verschillende blokken in de implementatie van het hele cryptosysteem op herprogrammeerbare hardware (FPGA). Twee delen maken een duidelijk onderscheid tussen de bewerkingen in $GF(2^n)$ en $GF(p)$. Dezelfde hiërarchie van bewerkingen, geïllustreerd met Figuur 3.1, vormt de rode draad doorheen beide delen. Vertrekkende van de relatief eenvoudige structuren voor optellers en vermenigvuldigers, komen achtereenvolgens de modulaire optelling en Montgomery vermenigvuldiging, de puntoptelling en de puntverdubbeling aan bod. Bijzondere aandacht gaat in elk van de delen uit naar alternatieve implementaties van de Montgomery vermenigvuldiging. Enkele bijkomende omzettingen vervolledigen de nodige bewerkingen voor het uitvoeren van de puntvermenigvuldiging.

Eens het volledige elliptische kromme cryptosysteem op herprogrammeerbare hardware is geïmplementeerd, kunnen de metingen voor een vermogen- en tijdsaanval op de chip beginnen. Hiervoor is er nood aan een specifiek voor deze aanvallen ontwikkelde meetopstelling. Vooraleer in te gaan op de metingen zelf, geeft Hoofdstuk 4 een gedetailleerde beschrijving van deze gebruikte meetopstelling.

Hoofdstuk 4

Inleiding tot tijds- en vermogenaanvallen

4.1 Principe

Al van bij het ontstaan van de cryptografie, nemen mensen de uitdaging aan de cryptografische algoritmes te breken. Zij proberen het systeem aan te vallen om zo de geheime sleutel ervan te achterhalen. Naast de puur wiskundige aanvallen op de algoritmes, die de cryptanalisten uitvoeren, duikt er de laatste jaren een tweede vorm op: de aanvallen op de onderliggende hardware, ook wel implementatie-aanvallen genoemd. Zwakheden op dit niveau vormen immers een steeds groter gevaar, nu er meer en meer kleine, mobiele cryptosystemen op de markt zijn.
Elektronische systemen lekken op verschillende manieren informatie over wat er in hun binnenste gebeurt en daar maken de implementatie-aanvallen dankbaar gebruik van. Naast de aanvallen gebaseerd op tijds- en vermogenmeting, die we in deze thesis bespreken, is ook de meting van de elektromagnetische straling die het systeem uitzendt, mogelijk. Men gaat er bij implementatie-aanvallen van uit dat de variatie op de opgemeten parameter (vermogen, tijd,...) sterk verband houdt met de operaties die de chip intern uitvoert en de data die hij verwerkt. Indien met deze informatie kan achterhaald worden wat er in de chip gebeurt, kan dit leiden tot het ontdekken van de sleutel.

De tijdsaanvallen en de eenvoudige vermogenaanvallen zijn gebaseerd op hetzelfde basisprincipe. Ze maken beide gebruik van de conditionele takken in het algoritme om te achterhalen wat er in die chip gebeurt. Veronderstel dat de cryptochip op een bepaald moment het volgende stuk code uitvoert:

```
...
if (e == 1) then
   A = B + C mod N
end if
...
```

In dit stuk code is er dus een operatie, de modulaire optelling, die enkel uitgevoerd wordt indien de conditie ($e == 1$) waar is. Deze operatie zal natuurlijk de nodige tijd vergen en het nodige vermogen verbruiken. Men kan dus, door de totale vertraging

of het verbruikte vermogen in de tijd op te meten, achterhalen of de operatie al dan niet is uitgevoerd en dus of de conditie ($e == 1$) al dan niet waar was. Indien het uitgevoerde algoritme verschillende sprongcondities bevat, wordt het toepassen van dit principe natuurlijk iets complexer. Toch zal het nog steeds mogelijk zijn er informatie uit te halen.
Een sprongconditie is dus onveilig en lekt informatie als:

- in de ene ('if'-) tak een operatie wordt uitgevoerd met een verschillende duur of een verschillend vermogenverbruik dan in de andere, eventueel ontbrekende, ('else'-) tak en
- de sprongconditie data-afhankelijk is.

In het binaire algoritme voor de puntvermenigvuldiging komt een dergelijke sprongconditie voor. Er wordt namelijk in de i-de iteratie van de *for*-lus enkel een puntoptelling uitgevoerd, indien de i-de bit van de sleutel gelijk is aan 1. (zie Algoritme 3.2.6.1). Dit is een uiterst zwak punt van dit cryptosysteem, vermits het informatie over de sleutel blootgeeft.
De Hoofdstukken 5 en 6 bespreken uitgebreid de aanval gericht op deze zwakheid met behulp van tijds- respectievelijk vermogenanalyse. Het is echter niet zo moeilijk een systeem robuust te maken tegen deze aanvallen. Een meer verfijnde aanvalsmethode dringt zich daarom op, de differentiële vermogenanalyse. Hoofdstuk 7 gaat hier dieper op in. Alvorens over te gaan tot deze aanvallen, bespreekt Paragraaf 4.2 de gebruikte meetopstelling voor de tijds- en vermogenmetingen.

4.2 Meetopstelling

Vermogenaanvallen baseren zich op het vermogenverbruik van het cryptosysteem. Ze analyseren de vorm van de opgemeten vermogencurve. Tijdsaanvallen gebruiken de tijdsduur van de uitgevoerde bewerking om het cryptosysteem te breken. Het bepalen van deze tijdsduur kan op twee manieren:

- Door het observeren van de I/O signalen. In systemen met een *start*-ingang en een *done*-uitgang kan de totale duur van de bewerking makkelijk achterhaald worden.
- Door het analyseren van het vermogenverbruik. Wanneer het cryptosysteem actief is, verbruikt het veel meer vermogen dan wanneer er geen bewerkingen moeten worden uitgevoerd. Met dit in het achterhoofd kan op basis van de vermogencurve de tijdsduur van de bewerking bepaald worden.

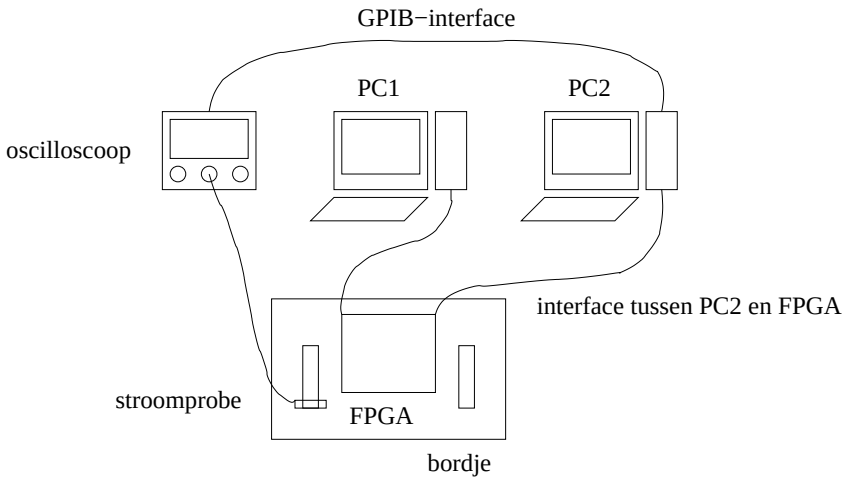
In deze thesis is gekozen voor de laatste mogelijkheid. De vermogen- en tijdsaanvallen baseren zich dus allebei op de waarde van het vermogenverbruik in functie van de tijd. Het vermogenverbruik P van de FPGA is op elk moment gelijk aan:

$$P = VCC * I_{VCC}$$

met VCC de voedingsspanning en I_{VCC} de stroom die uit de voeding wordt getrokken. VCC is constant en gekend. Het meten van I_{VCC} vraagt echter een niet-triviale meetopstelling die uit volgende onderdelen bestaat (Figuur 4.1):

- een **FPGA** die de VHDL implementatie van het cryptosysteem bevat. Het programmeren van de FPGA gebeurt met een parallelle kabel vanuit een eerste computer **PC1**.

- een **bordje** waarop de FPGA gemonteerd is. Dit bordje is zodanig gebouwd dat het verbruikte vermogen van de FPGA op elk moment observeerbaar is. Het vermogenverbruik wordt met behulp van een **stroomprobe** gemeten.
- een **interface tussen een tweede computer PC2 en de FPGA** voor het zenden van I/O data en controle- en statussignalen naar en vanuit de FPGA.
- een **oscilloscoop** die het signaal van de stroomprobe zichtbaar maakt. De oscilloscoop communiceert met PC2 via een **GPIB-interface** die zowel het instellen van de oscilloscoop als het opslaan van het gemeten vermogen op PC2 mogelijk maakt.



Figuur 4.1: Schematische voorstelling van de meetopstelling

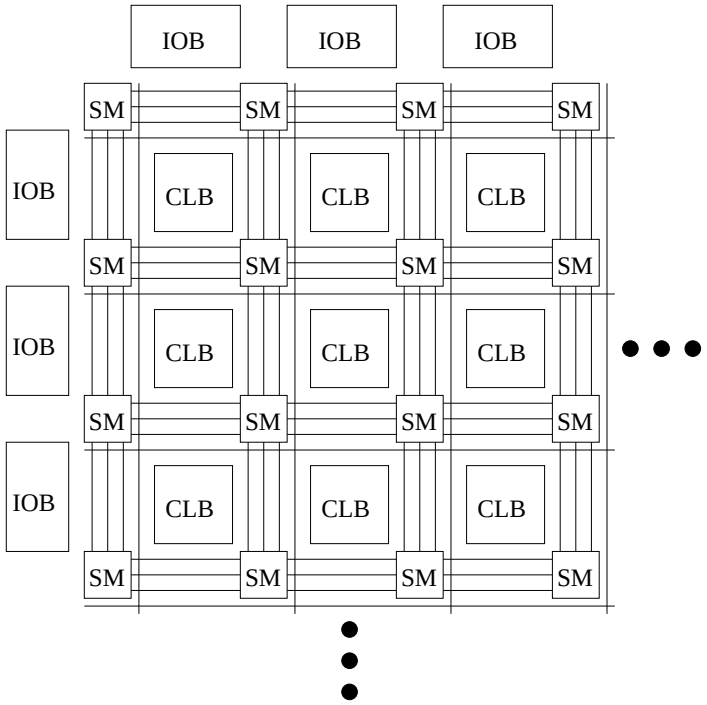
In deze opstelling zijn er twee verschillende PC's nodig, omdat zowel voor het programmeren van de FPGA als voor het zenden en ontvangen van data naar en vanuit de FPGA een parallelle poort nodig is. Elk van deze PC's beschikt over slechts één parallelle poort.
Een meer uitgebreide toelichting van de belangrijkste onderdelen bevindt zich in de volgende vier paragrafen. Paragraaf 4.2.5 geeft het resultaat van een eerste meting.

4.2.1 FPGA

Een FPGA (Field Programmable Gate Array) is een herprogrammeerbare hardware component. Het elliptische kromme cryptosysteem is geïmplementeerd in een XCV800-6-HQ240 Virtex van Xilinx. De volledige specificatie van deze FPGA zijn te vinden in [30]. Deze paragraaf vermeldt enkel de belangrijkste eigenschappen.
Eerst komt de algemene structuur van een Xilinx FPGA aan bod (zie ook [20]). Deze structuur verklaart waarom vermogen- en tijdsmetingen op FPGA en op ASIC dezelfde resultaten zullen geven. Daarna volgen enkele eigenschappen die specifiek zijn voor de in deze thesis gebruikte FPGA.

Structuur van de FPGA

Figuur 4.2 toont de algemene structuur van een Xilinx FPGA.



Figuur 4.2: Structuur van een Xilinx FPGA met configureerbare logische blokken (CLB), ingang-uitgang blokken (IOB) en schakelmatrices (SM)

De FPGA bestaat uit volgende basisblokken:

- De functionele logica bevindt zich in programmeerbare logische blokken (Configurable Logic Blocks, **CLB**'s).
- Ingangs-/uitgangsblokken (Input/Output Blocks, **IOB**'s) zorgen voor de interface tussen de pinnen van de verpakking en de CLB's.
- De CLB's zijn onderling verbonden via een netwerk van programmeerbare schakelmatrices (**SM**'s), die zorgen voor de routering van de interne signalen.

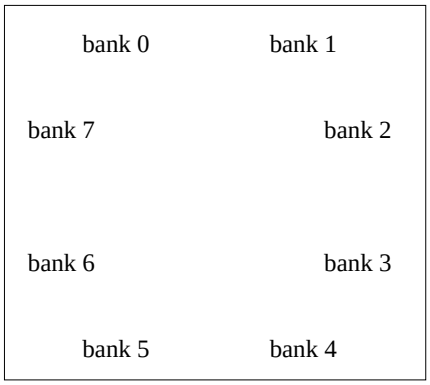
Een CLB bevat zowel combinatorische als sequentiële logica. Het sequentieel gedeelte bestaat uit 4 flipflops (FF's) die geconfigureerd kunnen worden als flank-getriggerde D-FF's of als niveau-gevoelige latches. Het veranderen van waarde of *omklappen* van deze FF's zorgt er voor dat er stroom uit de voeding wordt getrokken. Het aantal FF's dat omklapt bepaalt bijgevolg het vermogenverbruik van de FPGA op een bepaald tijdstip. Opdat tijds- en vermogenaanvallen op een FPGA hetzelfde resultaat zouden geven als op een ASIC mogen enkel de gebruikte logische poorten bijdragen tot het vermogenverbruik. De SM's in de FPGA garanderen dit. Ze zorgen er immers voor dat de ingangen van de ongebruikte CLB's niet aangesloten zijn. De FF's in deze CLB's kunnen daarom onmogelijk omklappen en beïnvloeden het vermogenverbruik niet. Een herprogrammeerbare FPGA implementatie kan dus gebruikt worden om het cryptosysteem bestand te maken tegen vermogen- en tijdsaanvallen alvorens het definitief te implementeren op een ASIC.

Specifieke eigenschappen van de XCV800-6-HQ240

De door ons gebruikte FPGA bevat 888 439 logische poorten ondergebracht in 56x84 CLB's.
De voedingsspanning voor de interne logica (VCCINT) is 2,5V, terwijl de uitgangsbuffers een voeding van 3,3V krijgen (VCCO). De grondspanning GND is gemeenschappelijk voor de hele FPGA.
De FPGA heeft 240 pinnen waarvan er maximaal 166 als I/O pinnen beschikbaar zijn. De overige pinnen zijn als volgt opgedeeld:

- 32 GND pinnen
- 16 VCCO pinnen
- 12 VCCINT pinnen
- 4 klok pinnen
- een aantal controle- en statuspinnen met betrekking tot het programmeren van de FPGA

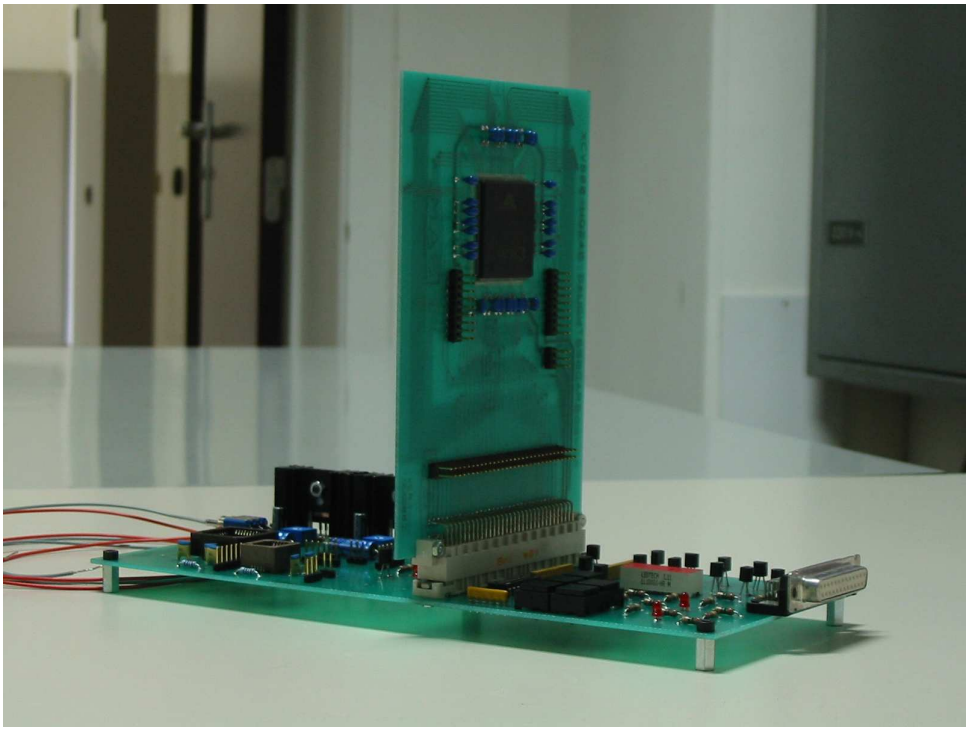
De CLB's en SM's in de FPGA zijn gegroepeerd in 8 *banken*. Figuur 4.3 geeft de positie van de banken in de FPGA. Elke bank krijgt haar VCCO en VCCINT via een aparte lijn (gaande van VCCO0 en VCCINT0 t.e.m. VCCO7 en VCCINT7). Elke lijn is met één of meer VCCO of VCCINT pinnen verbonden. Zo is elk van de 16 VCCO pinnen verbonden met één van de 8 VCCO lijnen (VCC0-VCC7) en is elk van de 12 VCCINT pinnen verbonden met één van de 8 VCCINT lijnen (VCCINT0-VCCINT7). De meetopstelling laat toe om zowel het totale vermogenverbruik als het vermogenverbruik in elke bank apart te meten. Details over de realisatie ervan bevinden zich in Paragraaf 4.2.2. Paragraaf 4.2.5 toont aan dat de gevulde banken van de FPGA meer vermogen verbruiken dan de lege.
De communicatie tussen de CLB's en de SM's over de banken heen gebeurt op dezelfde manier als de communicatie binnen een bank.



Figuur 4.3: Positie van de 8 banken in de FPGA

4.2.2 Bordje en stroomprobe

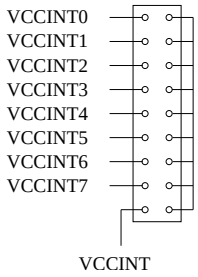
Figuur 4.4 toont het bordje dat in combinatie met een stroomprobe de vermogenmeting mogelijk maakt. De resultaten van de eerste metingen uitgevoerd met behulp van dit



Figuur 4.4: Het bordje waarop de FPGA (verticaal) gemonteerd is. Het horizontale gedeelte verzorgt de voeding en communicatie met de PC's.

bordje zijn terug te vinden in [25].

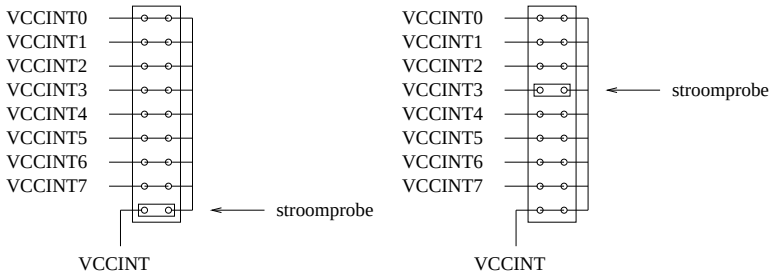
Zowel de stroom uit VCCINT als uit VCCO kan gemeten worden en dit zowel voor elke bank apart als voor alle banken samen.
Omdat de aanvallen in deze thesis gericht zijn op de FF's in de CLB's, meten ze enkel de stroom uit VCCINT. Figuur 4.5 toont de connector waarmee dit gebeurt. Links bevinden zich de contacten met de 8 VCCINT pinnen. Aan de rechterkant zijn alle lijnen met elkaar verbonden.



Figuur 4.5: Connector voor de vermogenmeting. De acht VCCINT-pinnen leveren de acht banken stroom

Het meten van de totale stroom gebeurt door de 8 bovenste contacten kort te sluiten en de stroomprobe over het onderste contact te plaatsen. De linkerkant van Figuur 4.6 geeft schematisch weer hoe de totale VCCINT stroom gemeten wordt.

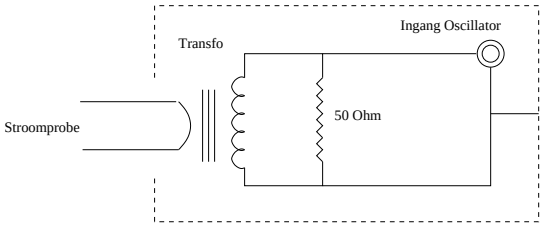
Om de VCCINT stroom van één bank te meten, wordt de stroomprobe over het overeenkomstige deel van de connector geplaatst. De overige lijnen worden kortgesloten. De rechterkant van Figuur 4.6 toont de opstelling voor het meten van bank 3.



(a) Totale vermogen (b) Vermogen in bank 3

Figuur 4.6: Configuraties van de connector voor het meten van het totale vermogen en het meten van het vermogen in bank 3

De stroomprobe vormt, net zoals de 8 andere verbindingen, een kortsluiting tussen twee contacten. Op die manier zorgt het bordje ervoor dat alle VCCINT lijnen met elkaar verbonden zijn. De stroom die door de stroomprobe vloeit, veroorzaakt door inductie een elektromagnetisch veld. De transformator, schematisch voorgesteld in Figuur 4.7 tussen stroomprobe en oscilloscoop meet de grootte van dit veld en geeft een spanning aan de oscilloscoop. De conversiefactor is zodanig dat een stroom I_{VCCINT} van 1mA wordt omgezet in een spanning van 5mV. Omdat het vermogen gelijk is aan $I_{VCCINT} * VCCINT$, komt een spanning van 5mV op de oscilloscoop overeen met een vermogen van $1mA * 2,5V = 2,5mW$. De geplotte resultaten van de metingen in Hoofdstuk 5, 6, en 7 zijn steeds spanningen. Om het vermogen hieruit te berekenen moeten deze spanningen vermenigvuldigd worden met 0,5.



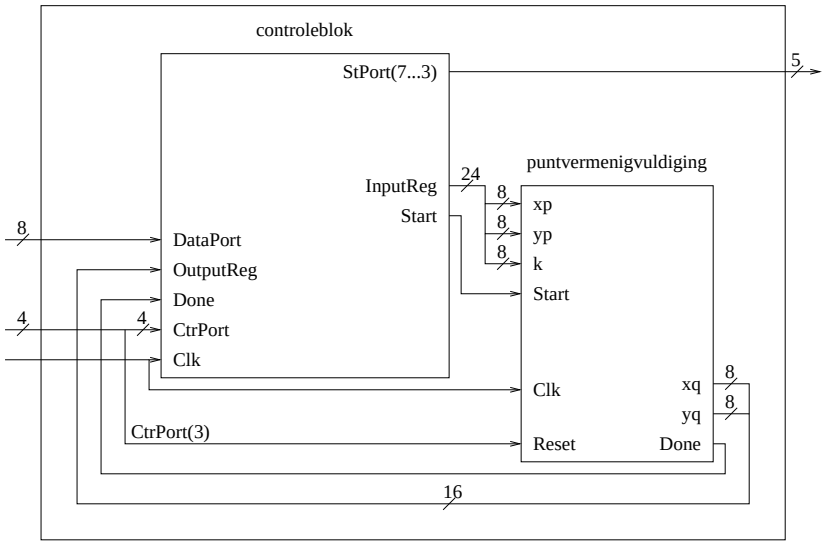
Figuur 4.7: Schematische voorstelling van verbinding stroomprobe naar oscillator

4.2.3 Interface tussen PC2 en FPGA

PC2 stuurt de gewenste ingangssignalen naar de FPGA en leest de uitgangssignalen uit. Het aantal in- en uitgangen van het cryptosysteem is echter veel groter dan het aantal beschikbare I/O pinnen van de FPGA. Daarom is het cryptosysteem niet rechtstreeks in de FPGA geïmplementeerd. Het bevindt zich samen met een tweede component in een overkoepelend hardware systeem met slechts 18 I/O signalen. Het

is dit systeem dat in de FPGA geïmplementeerd wordt. Hetzelfde probleem duikt op bij PC2. Deze computer zendt en ontvangt alle data via een parallelle poort met een beperkt aantal datapinnen. Een C-programma op PC2 verzorgt daarom de communicatie met de FPGA.

Het systeem dat in de FPGA wordt geïmplementeerd bestaat uit het elliptische kromme cryptosysteem en een controleblok. Figuur 4.8 toont deze hardware blokken voor een 8 bit puntvermenigvuldiging met een 8 bit sleutel. De uitgevoerde bewerking is $(xq, yq) = [k](xp, yp)$.

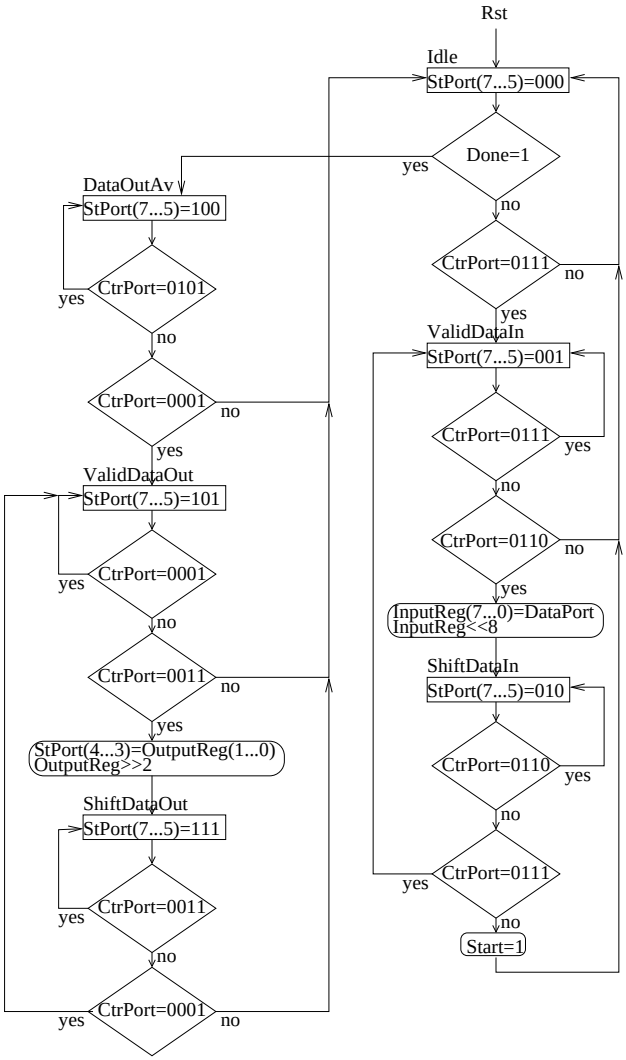


Figuur 4.8: Geïmplementeerde hardware in de FPGA

Het controleblok is een FSM die een schuifmechanisme aanstuurt om de data vanuit PC2 per 8 bits in te lezen. Op dezelfde manier verzorgt deze FSM het wegschrijven van de data naar PC2 per 2 bits (*StPort(4...3)*). De FSM bevindt zich in Figuur 4.9.

De signalen die via de 3 meest beduidende bits van de statuspoort, *StPort(7...5)*, naar PC2 worden gestuurd, geven aan in welke toestand de FSM zich bevindt. PC2 stuurt controlesignalen naar de FPGA via de controlepoort *CtrPort(3...0)*.
De communicatie tussen de FPGA en PC2 verloopt als volgt:

- PC2 zorgt ervoor dat de FSM in de *Idle*-toestand terecht komt door het resetsignaal *Rst* via *CtrPort(3)* hoog te maken.
- De FPGA geeft aan dat *Idle* bereikt is door *StPort(7...5)* op 000 te zetten.
- De FSM merkt dat *Done* gelijk is aan 0 omdat de puntvermenigvuldiging nog niet is uitgevoerd.
- PC2 geeft aan klaar te zijn voor het sturen van data door *CtrPort(3)* laag te maken. Op die manier is *Rst* niet langer hoog.
- Als de FSM zich in de *ValidDataIn*-toestand bevindt, wordt *StPort(7...5)* op 001 gezet om aan te geven dat de FPGA klaar is voor het ontvangen van data.
- PC2 zendt de 8 eerste databits naar *DataPort* op een dalende flank van *CtrPort(0)*.



Figuur 4.9: Controleblok state-machine in de FPGA. De 6 mogelijke toestanden worden door rechthoeken voorgesteld. De controle-signalen in de ruiten bepalen de toestandsovergangen.

- De databits worden in het minst beduidende gedeelte van *InputReg* geplaatst. *InputReg* is het register dat met de ingangen van de puntvermenigvuldiging is verbonden. De lengte van het register is gelijk aan de som van de lengtes van de ingangen. Een schuifbewerking over 8 plaatsen naar links zorgt ervoor dat de reeds opgeslagen data in *InputReg* niet overschreven worden in een van de volgende stappen. Daarna komt de FSM in de *ShiftDataIn*-toestand en wordt *StPort(7...5)* op 010 gezet.

- Als PC2 klaar is om de volgende 8 databits te zenden, wordt *CtrPort(0)* weer hoog gemaakt en worden de toestanden *ValidDataIn* en *ShiftDataIn* steeds opnieuw doorlopen.

- Wanneer PC2 alle ingangsdata in *InputReg* geschreven heeft, wordt *CtrPort* op 0000 gezet, waardoor het *Start*-signaal voor de puntvermenigvuldiging hoog wordt en de FSM in de *Idle*-toestand terecht komt.

- De puntvermenigvuldiging geeft een *Done*-signaal wanneer de uitgangen berekend zijn. De FSM komt in de *DataOutAv*-toestand en *StPort(7...5)* wordt op 100 gezet.

- *CtrPort* is ondertussen gelijk aan 0101, maar bij een dalende flank van *CtrPort(2)* komt de FSM in de *ValidDataOut*-toestand, waarbij *StPort(7...5)* op 101 wordt gezet.

- Een stijgende flank van *CtrPort(1)* gaat gepaard met het uitlezen van de 2 minst beduidende bits van *OutputReg* via *StPort(4...3)*. *OutputReg* is het register dat de uitgangen van de puntvermenigvuldiging bevat en waarvan de lengte gelijk is aan de som van de lengtes van de uitgangen. Na het uitlezen van deze bits wordt er een schuifbewerking over 2 plaatsen naar rechts uitgevoerd zodat de volgende 2 uitgangsbits klaar staan om uitgelezen te worden in een van de volgende stappen. Daarna komt de FSM in de *ShiftDataOut*-toestand en wordt *StPort(7...5)* op 111 gezet.

- Als PC2 klaar is om de volgende 2 uitgangsbits te ontvangen, wordt *CtrPort(1)* weer laag gemaakt en worden de toestanden *ValidDataOut* en *ShiftDataOut* steeds opnieuw doorlopen.

- Wanneer PC2 alle uitgangsdata uit *OutputReg* ingelezen heeft, wordt *CtrPort* op 0000 gezet en komt de FSM in de *Idle*-toestand.

4.2.4 Oscilloscoop en GPIB-interface

De gebruikte oscilloscoop is een TDS741L van Tektronix. Hij heeft een maximale bemonsteringsfrequentie van 500 Msamples/s en een intern geheugen van 8 MB. Op kanaal 1 wordt de externe trigger aangesloten. Dit is het signaal op ControlPort(0), dat van laag naar hoog overgaat bij het begin van de meting en terug laag wordt op het einde. Kanaal 2 wordt verbonden met de stroomprobe. Een GPIB-kaart in PC2 verzorgt de interface met de oscilloscoop.

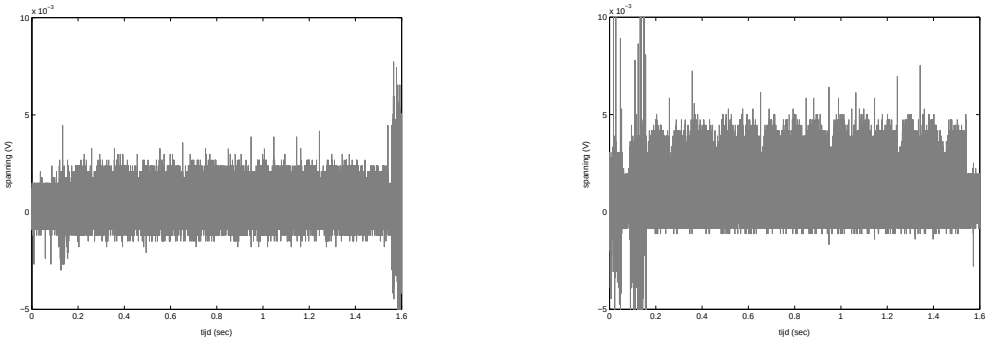
4.2.5 Meten in banken en 'proof of concept'

Om het concept in 4.2.1 kracht bij te zetten, beschrijft deze paragraaf de vermogenmeting van een 160 bit puntoptelling in $GF(p)$. Deze implementatie zal slechts een

Figuur 4.10: Gedeelte van de FPGA in beslag genomen door de modulaire opteller/aftrekker (grijs)

beperkt deel van de oppervlakte van de FPGA in beslag nemen. Figuur 4.10 toont het gebruikte gedeelte van de FPGA. De implementatie bezet vooral banken 4, 5, 6 en 7.

De resultaten van vermogenmetingen in bank 1 en bank 6 bevinden zich in Figuur 4.11.



(a) Bank 1

(b) Bank 6

Figuur 4.11: Vermogencurves voor een 160 bit puntoptelling in $GF(p)$ in de banken 1 en 6 met een klokfrequentie van 5 kHz. Gezien het vermogenverbruik vertoont de gebruikte bank 6 een duidelijk hogere activiteit.

Het valt op dat de dichter bezette bank 6 meer vermogen uit VCCINT trekt dan bank 1. Hetgeen in Paragraaf 4.2.1 werd voorspeld, blijkt voor dit simpele voorbeeld dus te kloppen: Het gemeten vermogen is evenredig met het aantal actieve FF's.

4.3 Besluit

Dit hoofdstuk overloopt de verschillende onderdelen van de meetopstelling nodig voor het uitvoeren van de tijds- en vermogenaanvallen op het elliptische kromme cryptosysteem. Alvorens op de functionaliteit van de afzonderlijke onderdelen in te gaan, leidt het bondig de begrippen tijds- en vermogenaanvallen in. De werking, onderlinge communicatie en eigenschappen van FPGA, meetbordje, stroomprobe, oscilloscoop en PC's worden dan na elkaar belicht. Tenslotte geeft een eerste meting van een lege en een volle bank van de FPGA in werking het bewijs van het concept van het meten van 'activiteit'. Bovendien is aangetoond dat meetresultaten op FPGA en op ASIC vergelijkbaar zijn.

Hoofdstuk 5 vat de resultaten van de tijdsanalyse uitgevoerd op een elliptische kromme cryptosysteem in het veld $GF(2^n)$ samen. Hoofdstuk 6 en 7 geven vervolgens de resultaten van de eenvoudige respectievelijk de differentiële vermogenanalyse uitgevoerd op een elliptische kromme cryptosysteem in het veld $GF(p)$.

Hoofdstuk 5

Tijdsanalyse in $GF(2^n)$

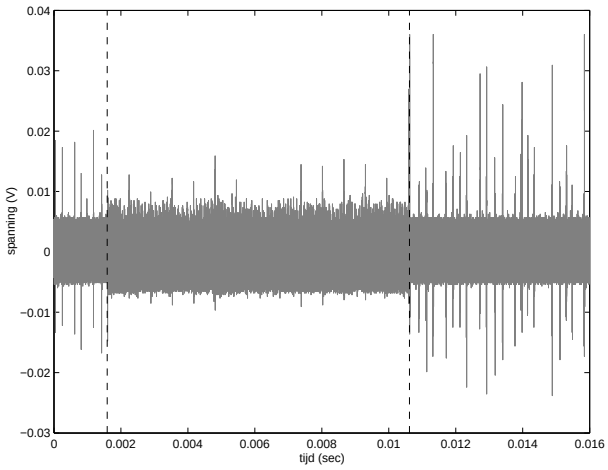
5.1 Inleiding

Zoals reeds aangehaald in Hoofdstuk 4 is de totale tijdsduur van een bewerking in een cryptosysteem afhankelijk van het al dan niet uitvoeren van bepaalde conditionele sprongen in het algoritme. Wanneer de sprongvoorwaarden opgelegd worden door de sleutelbits, kan het meten van de totale tijdsduur informatie over de geheime sleutel onthullen. Algoritme 3.2.6.1 doet vermoeden dat uit de totale tijdsduur van een puntvermenigvuldiging enkel het Hamming gewicht van de sleutel kan worden afgeleid. In [16] en [12] wordt echter een methode aangereikt om de volledige sleutel te achterhalen. Deze methode gaat er van uit dat de tijdsduur niet enkel afhankelijk is van de sleutel, maar ook van de andere ingangsdata. Dit geldt niet voor de puntvermenigvuldiging in $GF(2^n)$ door het ontbreken van modulaire reducties. Daarom beperkt de tijdsanalyse in deze thesis zich tot het bepalen van het *Hamming gewicht* (aantal 1-en) van de sleutel. Paragrafen 5.2 en 5.3 bespreken de meetresultaten voor de radix 1- respectievelijk de radix 4-implementatie. Hieruit blijkt dat de oorspronkelijke implementatie van de puntvermenigvuldiging niet bestand is tegen tijdsaanvallen die het Hamming gewicht proberen te achterhalen. Paragraaf 5.4 introduceert daarom enkele nieuwe implementaties voor de puntvermenigvuldiging. Om de resultaten duidelijk zichtbaar te maken, worden alle metingen uitgevoerd aan een zeer lage klokfrequentie van 250 kHz. Zowel voor het testen van de functionaliteit van de implementatie als voor het voorspellen van het vermogenverbruik wordt gebruik gemaakt van Matlab programma's. Deze zijn ontwikkeld in het kader van dit eindwerk en terug te vinden in Bijlage D.

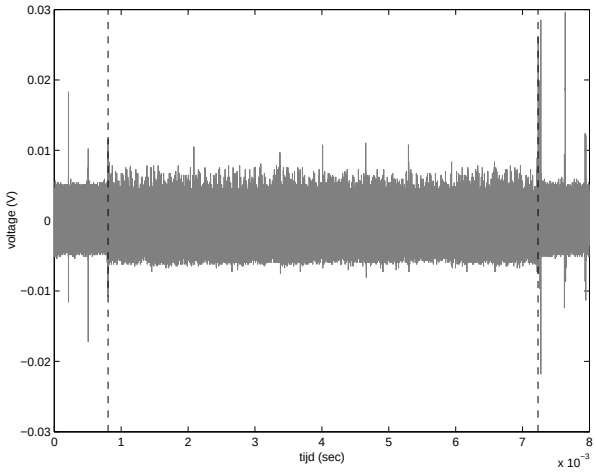
5.2 Metingen op de radix 1-implementatie

5.2.1 Puntoptelling en puntverdubbeling

Figuur 5.1 toont de vermogencurves van een 160 bit radix 1 puntoptelling en puntverdubbeling. De stippellijnen geven het begin en het einde van de bewerkingen aan. Vooraleer een bewerking start, zorgt het controleblok, beschreven in Paragraaf 4.2.3, er voor dat de ingangsdata via een schuifregister worden klaargezet. Het omklappen van de bits in dit register veroorzaakt hoge pieken in het gedeelte vóór de eerste stippellijn. Om dezelfde reden ontstaan de pieken na de tweede stippellijn bij het wegschrijven van de uitgangsdata. Het vermogenverbruik in de gebieden vóór de eerste en na de tweede



(a) Puntoptelling



(b) Puntverdubbeling

Figuur 5.1: Vermogencurves voor 160 bit radix 1 puntoptelling en puntverdubbeling, klokfrequentie 250 kHz. De stippellijnen geven begin en einde van de bewerking aan.

stippellijn is, met uitzondering van de hoge pieken, kleiner dan het vermogenverbruik tijdens de uitvoering van de bewerking. De reden hiervoor is dat de puntoptelling en -verdubbeling constant waarden naar registers schrijven, terwijl de schuifbewerking voor het inlezen en wegschrijven van data dat enkel op bepaalde momenten doet.

Tabel 5.1 geeft op basis van Algoritme 3.3.4.1 een overzicht van de deelbewerkingen in de algoritmes voor puntoptelling en puntverdubbeling. Volgens Paragraaf 3.3.2 is het aantal klokcycli voor een Montgomery vermenigvuldiging in de radix1-implementatie gelijk aan de woordlengte. In dit geval zal een Montgomery vermenigvuldiging dus 160 cycli in beslag nemen. Omdat de klokperiode gelijk is aan $\frac{1}{250\ kHz} = 4\ \mu s$, is de duur van een Montgomery vermenigvuldiging $160 * 4\mu s$ of 640 μs. Een modulaire optelling

Tabel 5.1: Volgorde van de bewerkingen in een puntoptelling en -verdubbeling ('TEL OP' staat voor een modulaire optelling, 'VERM' voor een Montgomery vermenigvuldiging)

	puntoptelling	puntverdubbeling
1	VERM	VERM
2	VERM	VERM
3	VERM en TEL OP	VERM
4	VERM en TEL OP	VERM
5	VERM	VERM
6	VERM en TEL OP	VERM
7	VERM	VERM en TEL OP
8	VERM	VERM en TEL OP
9	VERM	VERM en TEL OP
10	VERM en TEL OP	VERM
11	VERM en TEL OP	TEL OP
12	VERM en TEL OP	
13	VERM en TEL OP	
14	VERM	
15	TEL OP	

in even velden duurt slechts 1 klokcyclus of 4 μs in dit geval. De totale tijdsduur van een puntoptelling kan daarom als volgt berekend worden:

$$\begin{aligned} t_{puntoptelling} &= 14 * t_{Montgomery\ vermenigvuldiging} + t_{modulaire\ optelling} \\ &= 14 * 640\ \mu s + 4\ \mu s \\ &= 8964\ \mu s \end{aligned}$$

Voor de puntverdubbeling geldt

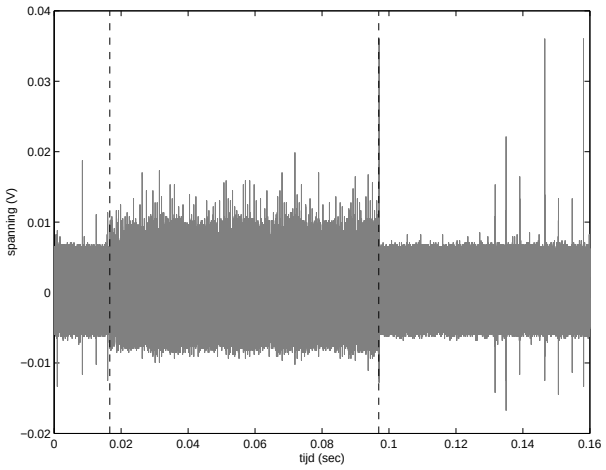
$$\begin{aligned} t_{puntverdubbeling} &= 10 * t_{Montgomery\ vermenigvuldiging} + t_{modulaire\ optelling} \\ &= 10 * 640\ \mu s + 4\ \mu s \\ &= 6404\ \mu s. \end{aligned}$$

Deze waarden zijn ook af te leiden uit de vermogencurves.

5.2.2 Puntvermenigvuldiging

De vermogencurve voor een 160 bit radix 1 puntvermenigvuldiging met een 8 bit sleutel wordt getoond in Figuur 5.2. De gebruikte sleutel is '10101101'. Voor elke 1 in de sleutel voert Algoritme 3.2.6.1 een puntverdubbeling en -optelling uit, voor elke 0 enkel een puntverdubbeling. De implementatie van de puntvermenigvuldiging gaat er van uit dat de MSB steeds gelijk is aan 1, waardoor de eerste lusherhaling van het algoritme wordt herleid tot een simpele toekenning van P aan Q. De overige 7 bits zorgen er voor dat de sequentie van puntverdubbelingen en -optellingen in Tabel 5.2 wordt uitgevoerd.

Op basis van de tabel kan de totale duur van de puntvermenigvuldiging berekend



Figuur 5.2: Vermogencurve voor 160 bit radix 1 puntvermenigvuldiging met sleutel '10101101', klokfrequentie 250 kHz. De stippellijnen geven begin en einde van de bewerking aan.

Tabel 5.2: Sequentie van puntoptellingen en -verdubbelingen bij vermenigvuldiging met scalar $'10101101'$

PUNTVERDUBBELING	0
PUNTVERDUBBELING PUNTOPTELLING	1
PUNTVERDUBBELING	0
PUNTVERDUBBELING PUNTOPTELLING	1
PUNTVERDUBBELING PUNTOPTELLING	1
PUNTVERDUBBELING	0
PUNTVERDUBBELING PUNTOPTELLING	1

worden:

$$\begin{aligned} t_{puntvermenigvuldiging} &= 7 * t_{puntverdubbeling} + 4 * t_{puntoptelling} \\ &= 7 * 6,404\ ms + 4 * 8,964\ ms \\ &= 80,684\ ms \end{aligned}$$

Deze tijdsduur is ook af te leiden uit de curve.

Een aanvaller die op basis van een tijdsanalyse het Hamming gewicht van de sleutel probeert te achterhalen, heeft enkel de vermogencurve in Figuur 5.2 en het gebruikte algoritme ter beschikking. Hij kan Tabel 5.3 opstellen, die de totale tijdsduur van de puntvermenigvuldiging geeft in functie van het Hamming gewicht. Dit is minstens gelijk aan 1, omdat de MSB steeds 1 is. De aanvaller leidt uit de vermogencurve een tijd van ongeveer 80 ms af en besluit succesvol dat het Hamming gewicht van de sleutel gelijk is aan vijf.

Tabel 5.3: Totale tijdsduur van een radix 4 puntvermenigvuldiging met 8 bit sleutel in functie van het Hamming gewicht (HG) van de sleutel

HG	duur (ms)
1	44,828
2	53,792
3	62,756
4	71,720
5	80,684
6	89,648
7	98,612
8	107,576

5.3 Metingen op de radix 4-implementatie

5.3.1 Puntoptelling en puntverdubbeling

Figuur 5.3 toont de vermogencurves van een 160 bit radix 4 puntoptelling en puntverdubbeling. Net zoals in Paragraaf 5.2 bevinden de uitgevoerde bewerkingen zich tussen de twee stippellijnen en kan de totale tijdsduur bepaald worden aan de hand van Tabel 5.1. Volgens Paragraaf 3.3.2 is het aantal klokcycli voor een Montgomery vermenigvuldiging in de radix4-implementatie gelijk aan $3 * \frac{woordlengte}{digitlengte}$ of 120 in dit geval. Dit komt overeen met $120 * 4\mu s$ of 480 μs. Een modulaire optelling duurt, net zoals in de radix1-implementatie, 1 klokcyclus of 4 μs. De totale tijdsduur van een puntoptelling is

$$\begin{aligned} t_{puntoptelling} &= 14 * t_{Montgomery\ vermenigvuldiging} + t_{modulaire\ optelling} \\ &= 14 * 480\ \mu s + 4\ \mu s \\ &= 6724\ \mu s. \end{aligned}$$

Voor de puntverdubbeling geldt

$$\begin{aligned} t_{puntverdubbeling} &= 10 * t_{Montgomery\ vermenigvuldiging} + t_{modulaire\ optelling} \\ &= 10 * 480\ \mu s + 4\ \mu s \\ &= 4804\ \mu s. \end{aligned}$$

Deze waarden komen overeen met de tijd tussen de stippellijnen in de vermogencurves.

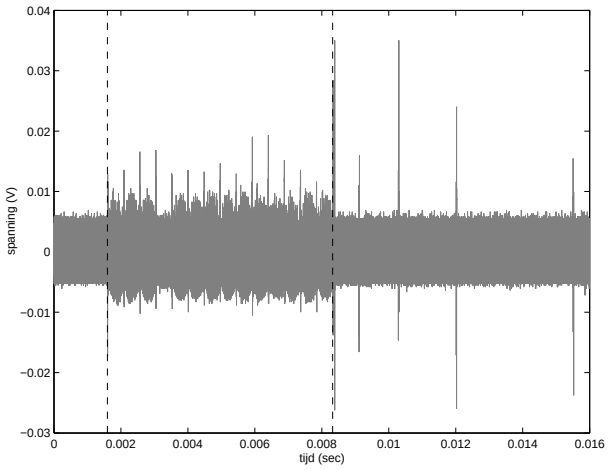
5.3.2 Puntvermenigvuldiging

Figuur 5.4 toont de vermogencurve voor een 160 bit radix 4 puntvermenigvuldiging met de sleutel '10101101'. Op dezelfde manier als in Paragraaf 5.2 wordt de totale duur van de puntvermenigvuldiging met behulp van Tabel 5.2 berekend:

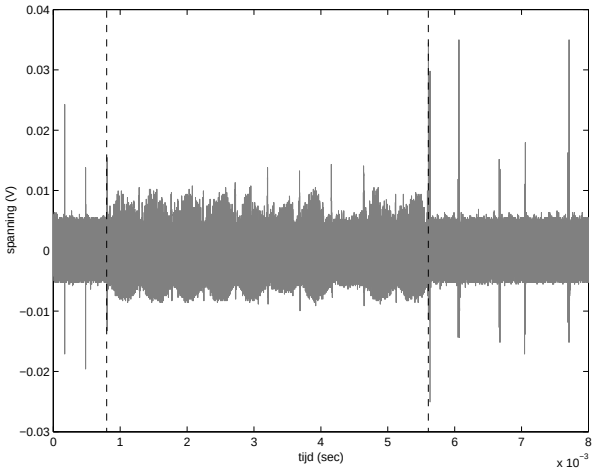
$$\begin{aligned} t_{puntvermenigvuldiging} &= 7 * t_{puntverdubbeling} + 4 * t_{puntoptelling} \\ &= 7 * 4,804\ ms + 4 * 6,724\ ms \\ &= 60,524\ ms \end{aligned}$$

Op de curve kan dezelfde tijdsduur worden afgelezen.

Met behulp van Tabel 5.4 kan een aanvaller de afgelezen tijdsduur in Figuur 5.4 van ongeveer 60 ms linken aan een Hamming gewicht van vijf.



(a) Puntoptelling

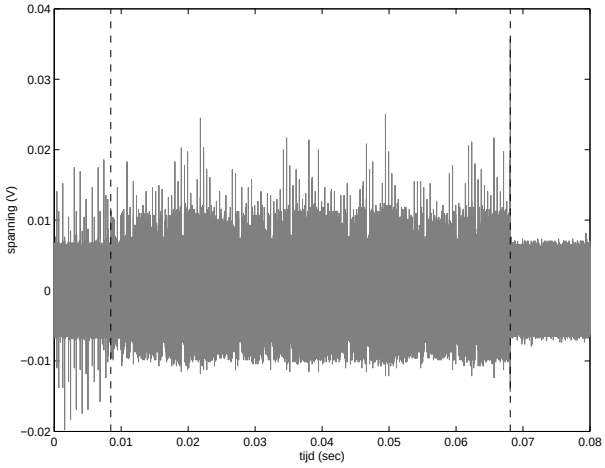


(b) Puntverdubbeling

Figuur 5.3: Vermogencurves voor 160 bit radix 4 puntoptelling en puntverdubbeling, klokfrequentie 250 kHz. De stippellijnen geven begin en einde van de bewerking aan.

5.4 Wegwerken van de zwakheden

In de vorige paragrafen werd duidelijk dat een aanvaller aan de hand van de vermogencurve van het cryptosysteem zeer makkelijk het Hamming gewicht van de sleutel kan bepalen. Om het systeem bestand te maken tegen dit soort tijdsaanvallen, introduceert deze paragraaf twee alternatieve implementaties van de puntvermenigvuldiging.



Figuur 5.4: Vermogencurve voor 160 bit radix 4 puntvermenigvuldiging met sleutel '10101101', klokfrequentie 250 kHz. De stippellijnen geven begin en einde van de bewerking aan.

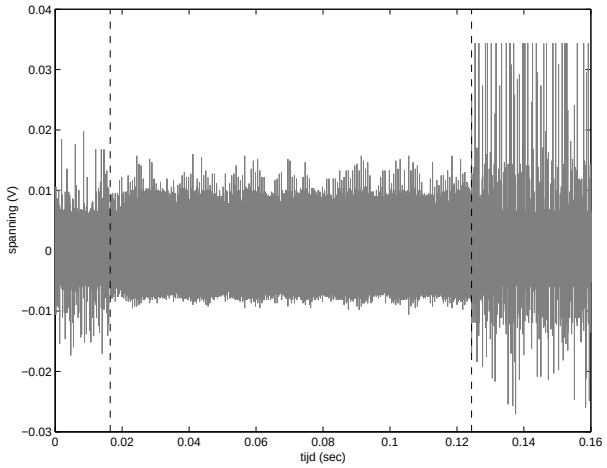
Tabel 5.4: Totale tijdsduur van een puntvermenigvuldiging met 8 bit sleutel in functie van het Hamming gewicht (HG)

HG	duur (ms)
1	33,628
2	40,352
3	47,076
4	53,800
5	60,524
6	67,248
7	73,972
8	80,696

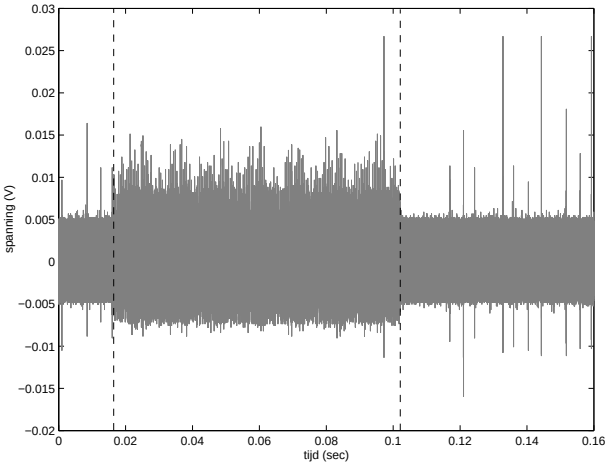
5.4.1 Eerste alternatief: altijd puntoptelling en puntverdubbeling in serie

Een eerste mogelijkheid om een aanvaller te beletten het Hamming gewicht van de sleutel te achterhalen, is gebaseerd op het klassieke algoritme voor de puntvermenigvuldiging (Algoritme 3.2.6.1). In een licht aangepaste vorm van dit algoritme, Algoritme 5.4.1.1, wordt de puntoptelling in elke lusherhaling uitgevoerd, ongeacht de waarde van de sleutelbit. Het al dan niet gebruiken van het resultaat van de puntoptelling in de volgende puntverdubbeling is echter wel afhankelijk van de sleutelbit. Figuur 5.5 toont de vermogencurves van de radix 1 en radix 4 puntvermenigvuldiging op basis van Algoritme 5.4.1.1.

Voor elke sleutelbit wordt er een puntoptelling en een puntverdubbeling in serie uitgevoerd. De totale tijdsduur van de radix 1 puntvermenigvuldiging is daarom gelijk



(a) Radix 1



(b) Radix 4

Figuur 5.5: Vermogencurves voor 160 bit radix 1 en radix 4 puntvermenigvuldiging met sleutel '10101101' volgens Algoritme 5.4.1.1, klokfrequentie 250 kHz. De stippellijnen geven begin en einde van de bewerking aan.

Algoritme 5.4.1.1 : Puntvermenigvuldiging (alternatief 1)

INPUT: Een punt P, geheel getal k, $0 < k < 2^n$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)$, $k_{n-1} = 1$
OUTPUT: $Q = [k]P$

1. $Q \leftarrow P$
2. **for** i from $n-2$ to 0 **do**:
3. $Q_1 \leftarrow 2Q$
4. $Q_2 \leftarrow Q_1 + P$
5. **if** $k_i = 1$ **then**
6. $Q \leftarrow Q_2$
7. **else**
8. $Q \leftarrow Q_1$
9. **end if**
10. **end for**
11. Return Q

---

aan

$$\begin{aligned} t_{puntvermenigvuldiging} &= 7 * t_{puntverdubbeling} + 7 * t_{puntoptelling} \\ &= 7 * 6,404\ ms + 7 * 8,964\ ms \\ &= 107,576\ ms. \end{aligned}$$

Voor de radix 4 puntvermenigvuldiging geldt

$$\begin{aligned} t_{puntvermenigvuldiging} &= 7 * t_{puntverdubbeling} + 7 * t_{puntoptelling} \\ &= 7 * 4,804\ ms + 7 * 6,724\ ms \\ &= 80,696\ ms. \end{aligned}$$

De totale tijdsduur is nu onafhankelijk van het Hamming gewicht van de sleutel. De mogelijkheid om het Hamming gewicht te bepalen op basis van een tijdsanalyse is dus uitgesloten.

Het uitvoeren van een puntoptelling en puntverdubbeling in serie vertraagt het systeem aanzienlijk. Een parallelle uitvoering van de twee bewerkingen zou daarom beter zijn. Algoritme 5.4.1.1 laat dit echter niet toe omdat de puntoptelling steeds het resultaat van de voorafgaande puntverdubbeling gebruikt. Daarom wordt een tweede alternatief voorgesteld dat gebruik maakt van een ander algoritme voor de puntvermenigvuldiging.

5.4.2 Tweede alternatief: altijd puntoptelling en puntverdubbeling in parallel

Het tweede alternatief voor de implementatie van de puntvermenigvuldiging is gebaseerd op Algoritme 5.4.2.1. In tegenstelling tot Algoritme 3.2.6.1 wordt hier de sleutel van LSB naar MSB geëvalueerd.

Om te beletten dat het Hamming gewicht van de sleutel kan achterhaald worden met behulp van een tijdsanalyse, moet de puntoptelling steeds uitgevoerd worden. Algoritme 5.4.2.2 realiseert dit. Wanneer de sleutelbit gelijk is aan 1 (0), wordt het resultaat

Algoritme 5.4.2.1 : Puntvermenigvuldiging (van LSB naar MSB)

INPUT: Een punt P, geheel getal k, $0 < k < 2^n$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)$
OUTPUT: $Q = [k]P$

1. $Q \leftarrow \mathcal{O}$, $S \leftarrow P$
2. **for** i from 0 to $n-1$ **do**:
3. **if** $k_i = 1$ **then**
4. $Q \leftarrow Q + S$
5. **end if**
6. $S \leftarrow 2S$
7. **end for**
8. Return Q

---

toegekend aan Q_2 (Q_3). Q_3 wordt verder niet gebruikt. In Algoritme 5.4.2.1 moet een optelling met het punt op oneindig $\mathcal{O}$ worden uitgevoerd bij de eerste sleutelbit gelijk aan 1. Algoritme 5.4.2.2 realiseert dit door de initiële waarde van Q_1 op $((0,\ldots,0),(0,\ldots,0),(0,\ldots,0))$ te zetten. Bij de eerste sleutelbit gelijk aan 1, heeft Q_1 nog geen nieuwe waarde gekregen en wordt S toegekend aan Q_1. Bij alle volgende lusherhalingen waarin $k_i = 1$ zal $Q_1 + S$ worden toegekend aan Q_1. De optelling $Q_1 + S$ wordt steeds uitgevoerd, ook al is de eerste sleutelbit gelijk aan 1 nog niet geëvalueerd. Dit belet een aanvaller om de eerste 1 in de sleutel te ontdekken met behulp van een tijdsanalyse.

Algoritme 5.4.2.2 : Puntvermenigvuldiging (alternatief 2)

INPUT: Een punt P, geheel getal k, $0 < k < 2^n$, $k = (k_{n-1}, k_{n-2}, \ldots, k_0)$
OUTPUT: $Q = [k]P$

1. $Q_1 \leftarrow ((0,\ldots,0),(0,\ldots,0),(0,\ldots,0))$, $S \leftarrow P$
2. **for** i from 0 to $n-1$ **do**:
3. **if** $k_i = 1$ **then**
4. $Q_2 \leftarrow Q_1 + S$
5. **if** $Q_1 = ((0,\ldots,0),(0,\ldots,0),(0,\ldots,0))$ **then**
6. $Q_1 \leftarrow S$
7. **else**
8. $Q_1 \leftarrow Q_2$
9. **end if**
10. **else**
11. $Q_3 \leftarrow Q_1 + S$
12. **end if**
13. $S \leftarrow 2S$
14. **end for**
15. Return Q_1

Eén van de ingangen van de puntoptelling in Algoritme 5.4.1.1 was steeds gelijk aan P met $Z_P = 1$. Het gebruikte algoritme voor de puntoptelling hield er rekening mee dat de Z-coördinaat van één van de ingangspunten steeds gelijk was aan 1. In Algoritme

5.4.2.2 zijn de Z-coördinaten van beide ingangen voor de puntoptelling verschillend van 1. Daarom wordt de puntoptelling uitgevoerd door Algoritme 5.4.2.3, voorgesteld in [19].

Algoritme 5.4.2.3 : Puntoptelling

INPUT: 2 punten op de curve: $P_1(X_1, Y_1, Z_1), P_2(X_2, Y_2, Z_2)$
coëfficiënt van de curve: a
OUTPUT: $P_3(X_3, Y_3, Z_3) = P_1 + P_2$

1. $T_1 \leftarrow Z_1 * Z_1$
2. $T_1 \leftarrow Y_2 * T_1$
3. $T_2 \leftarrow Z_2 * Z_2$
4. $T_2 \leftarrow Y_1 * T_2$
5. $T_2 \leftarrow Z_1 * X_2$ $T_3 \leftarrow T_1 + T_2$
6. $T_4 \leftarrow X_1 * Z_2$
7. $T_5 \leftarrow Z_1 * Z_2$ $T_4 \leftarrow T_2 + T_4$
8. $T_6 \leftarrow T_4 * T_5$
9. $Z_3 \leftarrow T_6 * T_6$
10. $T_7 \leftarrow T_5 * T_5$
11. $T_7 \leftarrow a * T_7$
12. $T_8 \leftarrow T_4 * T_4$ $T_7 \leftarrow T_6 + T_7$
13. $T_7 \leftarrow T_7 * T_8$
14. $T_6 \leftarrow T_3 * T_6$
15. $T_3 \leftarrow T_3 * T_3$ $T_7 \leftarrow T_6 + T_7$
16. $T_2 \leftarrow T_2 * T_5$ $X_3 \leftarrow T_3 + T_7$
17. $T_4 \leftarrow T_4 * T_4$
18. $T_2 \leftarrow T_2 * T_4$
19. $T_1 \leftarrow T_1 * T_4$ $T_2 \leftarrow T_2 + X_3$
20. $T_2 \leftarrow T_2 * T_6$ $T_1 \leftarrow T_1 + X_3$
21. $T_1 \leftarrow T_1 * Z_2$
22. $Y_3 \leftarrow T_1 + T_2$
23. Return X_3
24. Return Y_3
25. Return Z_3

Uit dit algoritme kan de tijdsduur van een radix 1 puntoptelling worden afgeleid:

$$\begin{aligned} t_{puntoptelling} &= 21 * t_{Montgomery\ vermenigvuldiging} + t_{modulaire\ optelling} \\ &= 21 * 640\ \mu s + 4\ \mu s \\ &= 13444\ \mu s \end{aligned}$$

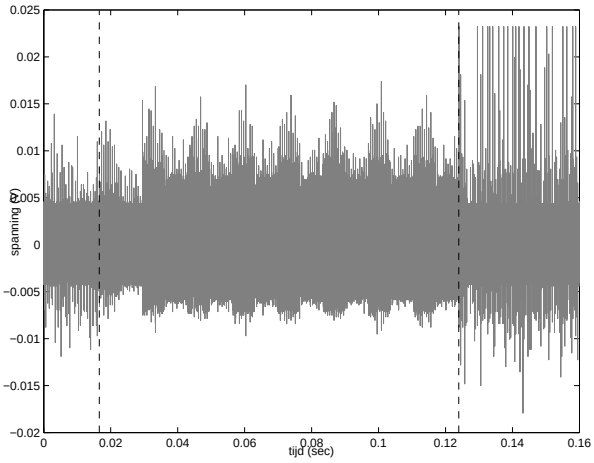
Voor de radix 4 puntoptelling geldt:

$$\begin{aligned} t_{puntoptelling} &= 21 * t_{Montgomery\ vermenigvuldiging} + t_{modulaire\ optelling} \\ &= 21 * 480\ \mu s + 4\ \mu s \\ &= 10084\ \mu s \end{aligned}$$

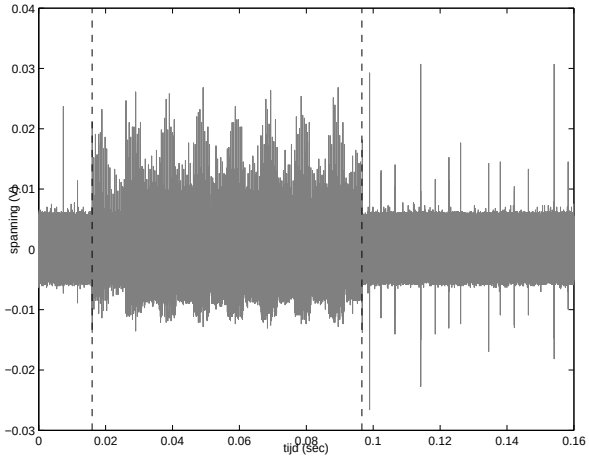
In tegenstelling tot Algoritme 5.4.1.1 is het in dit geval wel mogelijk om de puntverdubbeling in parallel met de puntoptelling uit te voeren, omdat de ene bewerking

niet moet wachten op het resultaat van de andere. Er zijn dan wel twee verschillende Montgomery vermenigvuldigers nodig. Er is hiervoor nog voldoende ruimte op de FPGA.

Figuur 5.6 toont de vermogencurves voor de radix 1 en radix 4 implementatie van het tweede alternatief voor de puntvermenigvuldiging.



(a) Radix 1



(b) Radix 4

Figuur 5.6: Vermogencurves voor 160 bit radix 1 en radix 4 puntvermenigvuldiging met sleutel '10101101' volgens Algoritme 5.4.2.2, klokfrequentie 250 kHz. De stippellijnen geven begin en einde van de bewerking aan.

Voor elke sleutelbit wordt er een puntoptelling en een puntverdubbeling in parallel uitgevoerd. De totale tijdsduur van de radix 1 puntvermenigvuldiging is daarom gelijk

aan

$$\begin{aligned} t_{puntvermenigvuldiging} &= 8 * t_{puntoptelling} \\ &= 8 * 13,444\ ms \\ &= 107,552\ ms. \end{aligned}$$

Voor de radix 4 puntvermenigvuldiging is dit

$$\begin{aligned} t_{puntvermenigvuldiging} &= 8 * t_{puntoptelling} \\ &= 8 * 10,084\ ms \\ &= 80,672\ ms. \end{aligned}$$

Net zoals bij het eerste alternatief is de totale tijdsduur onafhankelijk van het Hamming gewicht van de sleutel, waardoor een tijdsaanval onmogelijk wordt. De verhoopte snelheidswinst door het parallelliseren van de puntoptelling en -verdubbeling van het tweede alternatief is achterwege gebleven. De verklaring hiervoor is de aangepaste puntoptelling in Algoritme 5.4.2.3 die veel trager is dan de de puntoptelling in Algoritme 3.3.4.1.

5.5 Besluit

Dit hoofdstuk toont aan dat een cryptosysteem makkelijk resistent gemaakt kan worden tegen een tijdsaanval die het Hamming gewicht tracht te achterhalen. Hoofdstuk 6 zal daarom andere analyses uitvoeren op de vermogencurves. De eenvoudige vermogenanalyse zal uit de vorm van de vermogencurves de geheime sleutel proberen te ontdekken.

Hoofdstuk 6

Eenvoudige vermogenanalyse in $GF(p)$

6.1 Inleiding

Het vermijden van het ongewenst lekken van informatie in het vermogengebruik van een chip, is een moeilijke opgave voor een cryptograaf. Een chip bestaat immers uit een grote hoeveelheid transistoren (meestal CMOS), die samen logische poorten, registers,... vormen. Naast statisch vermogen (onafhankelijk van de activiteit van de chip) verbruiken deze ook veel dynamisch vermogen (bijkomend vermogen nodig voor schakelen van transistoren in werking). De grootte van het dynamisch vermogenverbruik is afhankelijk van het al dan niet omklappen van de logische waarden aan de uitgangen van logische poorten of registers. Wanneer een signaal van een logische 0 (lage spanning) naar een logische 1 (hoge spanning) wordt gebracht, kost dit namelijk vermogen. Er wordt stroom uit de voeding getrokken. Een overgang van een logische 1 naar een logische 0 gaat dan weer gepaard met een injectie van stroom in de grond. De vermogenconsumptie hangt bijgevolg zeer nauw samen met de data die de chip op dat ogenblik verwerkt.
Het verbruikte vermogen is dus duidelijk fysisch waarneembaar aan de buitenkant van de chip door te meten hoeveel stroom er uit de grond of voeding getrokken wordt. Vermits bij het schrijven van grote registers vele logische signalen tegelijk veranderen, kan dit schrijven eenvoudig opgemerkt worden in de vorm van een piek in het vermogenverbruik.

Een vermogengebaseerde aanval is, net zoals een tijdsgebaseerde aanval, voor de aanvaller zeer eenvoudig uit te voeren. Er is relatief weinig apparatuur voor nodig (ter waarde van enkele duizenden euro, zie Hoofdstuk 4) en de aanval is niet verwoestend. Hij kan vele keren uitgevoerd worden en is toch moeilijk detecteerbaar. Dit alles maakt vermogenanalyse tot een grote uitdaginge voor de cryptograaf.

Eenvoudige vermogenanalyse (Simple Power Analysis, SPA) werd ingevoerd door Kocher in 1998 [17] om het onderscheid te maken met differentiële vermogenanalyse. SPA bestudeert visueel de vermogenvariatie aan de hand van slechts één enkele uitvoering van het algoritme dat onder vuur ligt. Het doel van deze aanval is het raden/ontdekken op welk ogenblik welke instructie van het cryptografisch algoritme uitgevoerd wordt. Paragraaf 4.1 beschreef reeds hoe het mogelijk is te detecteren of in het algoritme een conditionele sprong genomen wordt of niet. Op deze manier lekt het circuit informatie

over de verwerkte data. Het elliptische kromme puntvermenigvuldigingsalgoritme (Algoritme 3.2.6.1) bevat zulk een sprongconditie. Het al dan niet nemen van de sprong hangt hier af van de waarde van de sleutelbits. Dit is een heel zwakke plek van het algoritme. Het is dan ook dit punt dat we, net als bij de tijdsanalyse, met de eenvoudige vermogenanalyse gaan aanvallen.

Voor SPA volstaat slechts één vermogenmeting op de werkende chip. De bekomen vermogenstroom (verbruikte vermogen ten opzichte van de tijd) wordt vervolgens visueel geanalyseerd. Om goed de verschillende zones in deze vermogencurve te kunnen onderscheiden, beginnen we met het meten van de basisbouwblokken, de modulaire opteller en de Montgomery vermenigvuldiger. Eens de vermogenpatronen van deze basisbewerkingen duidelijk zijn, komen de algoritmes van de puntoptelling en puntverdubbeling aan de beurt. In de laatste stap zijn in de vermogencurve van de puntvermenigvuldiging de patronen van deze puntoptelling en puntverdubbeling te herkennen. Er zal blijken dat de geheime scalar waarmee vermenigvuldigd wordt, zomaar voor het oprapen ligt. Deze metingen voeren we zowel uit op onze implementatie in radix 1 als in radix 4 (Paragraaf 6.2 en 6.3).
De grootste zwakheid van deze aanval is dat de aanvaller moet beschikken over de volledige kennis van het geïmplementeerde algoritme. Bovendien is het niet zo moeilijk een chip resistent te maken tegen deze vorm van aanvallen. Het enige dat moet gebeuren, is het weren van alle data-afhankelijke conditionele sprongen. Paragraaf 6.4 bespreekt enkele alternatieven om onze implementatie SPA-resistent te maken. Om het systeem nu nog te kunnen aanvallen is een meer verfijnde vorm van vermogenanalyse nodig. Hoofdstuk 7 gaat daarom dieper in op de differentiële vermogenanalyse.

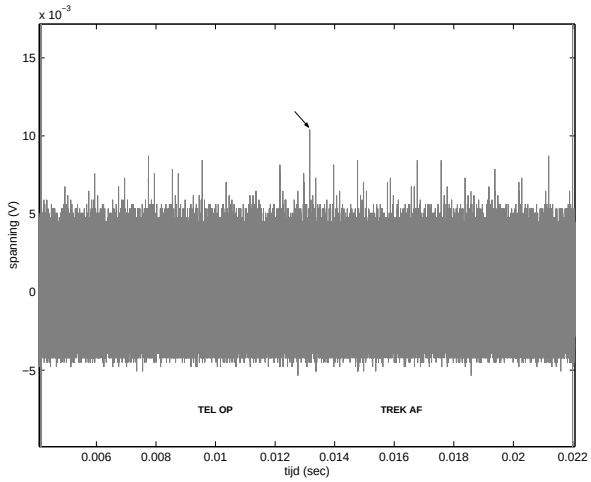
6.2 Metingen op de radix 1-implementatie

6.2.1 Modulaire optelling

Figuur 6.1 toont de vermogencurve van een modulaire optelling van 160 bits en een digit-lengte van 4 bits. De chip werd geklokt aan 5 kHz en de oscilloscoop nam 4M metingen aan 100M metingen per seconde. Voor de vermogenanalyse is enkel de vorm van de vermogencurves tijdens het uitvoeren van de bewerking van belang. De figuren van dit en volgend hoofdstuk geven daarom enkel dit deel van de vermogencurve weer. De modulaire optelling (zie Algoritme 3.2.2.1) bestaat steeds uit een gewone optelling, gevolgd door een gewone aftrekking. Tussen deze twee bewerkingen wordt het resultaat van de eerste operatie weggeschreven en worden de ingangen van de tweede operatie aangelegd. Dit vertaalt zich in een piek in het vermogenverbruik. Deze is op Figuur 6.1 aangeduid door een pijltje dat de overgang van de optelling naar de aftrekking aangeeft.
Binnen één optelling/aftrekking vinden er $(aantalbits)/(digit-lengte)$ kleine optellingen plaats. In dit geval 40 keer een 4 bit-optelling met behulp van een carry look ahead opteller. Deze zijn op de vermogencurve echter niet waarneembaar. Het schrijven van 4 bits is onvoldoende om altijd een duidelijke piek te genereren.

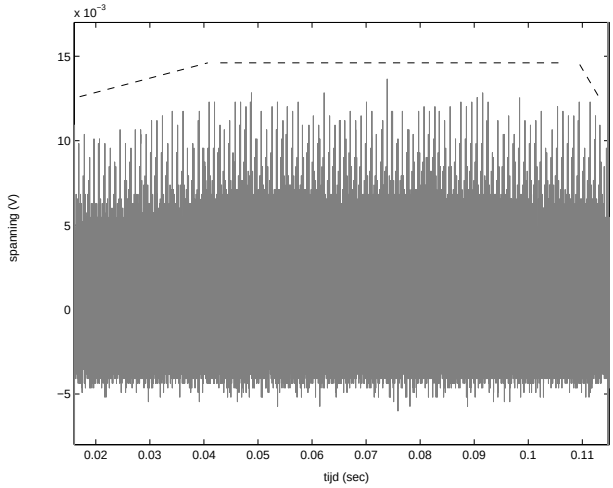
6.2.2 Montgomery vermenigvuldiging

Het tweede basisbouwblok is de Mongtomery vermenigvuldiger. Eerst komt de radix1-implementatie aan bod. Paragraaf 6.3 analyseert de vermenigvuldiger met radix 4.
De vermogencurve van een Montgomery radix 1 vermenigvuldiging voor 160 bits is



Figuur 6.1: Vermogencurve voor 160-bit modulaire optelling, klokfrequentie 5 kHz. Het pijltje duidt op het vermogenpiek bij het wegschrijven van het eerste tussenresultaat van de optelling.

te zien op Figuur 6.2. De klokfrequentie is 5 kHz en de oscilloscoop werkte aan 25M metingen per seconde (4M metingen).



Figuur 6.2: Vermogencurve voor 160-bit Montgomery vermenigvuldiging radix 1, klokfrequentie 5 kHz. De stippellijn verduidelijkt de trapezium-vorm.

Deze plot geeft weinig informatie vrij over de verschillende stappen waaruit het algoritme is opgebouwd. Het wegschrijven van het eindresultaat op het einde van de curve is wel duidelijk. Dit gebeurt na 96,8 msec, vermits één 160-bit Montgomery vermenigvuldiging 485 klokslagen (elk 0,2 msec met een klok van 5kHz) duurt (zie ook Hoofdstuk 3, in het bijzonder Tabel 3.4). Tussen het begin van de vermenigvuldiging en het wegschrijven is geen duidelijk patroon herkenbaar. Het register met het tussenresultaat T wordt elke klokslag ingelezen. Een vermogenpiek op elk van de 484

klokslagen zorgt ervoor dat de afzonderlijke pieken niet meer duidelijk waarneembaar zijn.
De systolische reeks van de Mongtomery vermenigvuldiger is opgebouwd uit 160 afzonderlijke blokjes. Aangezien deze één lange gepijplijnde ketting vormen (zie Paragraaf 3.2.3), zullen de blokjes verder in de rij slechts na een bepaalde tijd hun eerste ingangen krijgen en vermogen beginnen te verbruiken. De eerste blokjes zullen dan weer tegen het eind aan geen nieuwe ingangen meer krijgen en dus weer geen vermogen trekken. Dit heeft tot gevolg dat het vermogenverbuik in het begin lichtjes zal stijgen, tot het een vrij constante waarde bereikt wanneer alle blokjes aan het werk zijn. Tegen het einde zal het verbruik dan weer lichtjes beginnen dalen. Dit zien we ook (zij het zwakjes) op de plot. Bij de radix 4 implementatie zal dit veel duidelijker zijn. We zullen dit verder de 'trapezium'-vorm van de Mongtomery vermenigvuldiging noemen.

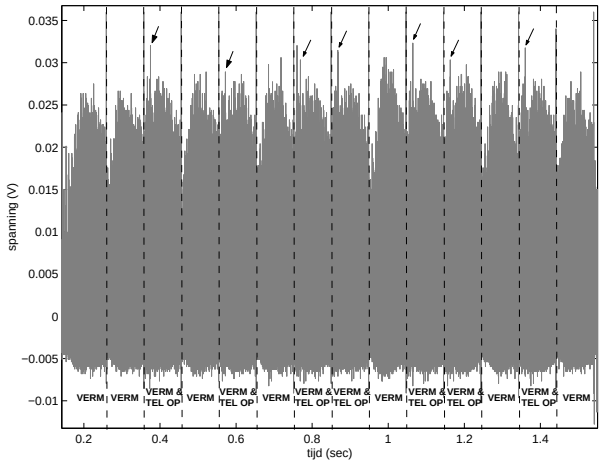
6.2.3 Puntoptelling en puntverdubbeling

Nu de vermogencurves van een modulaire optelling en een Montgomery vermenigvuldiging gekend zijn, komen de puntoptelling en puntverdubbeling aan de beurt. Tabel 6.1 geeft de opeenvolgende bewerkingen in een puntoptelling en puntverdubbeling (zie ook Algoritme 3.2.5.1).

Tabel 6.1: Volgorde van de bewerkingen in een puntoptelling en -verdubbeling ('TEL OP' staat voor een modulaire optelling, 'VERM' staat voor een Montgomery vermenigvuldiging)

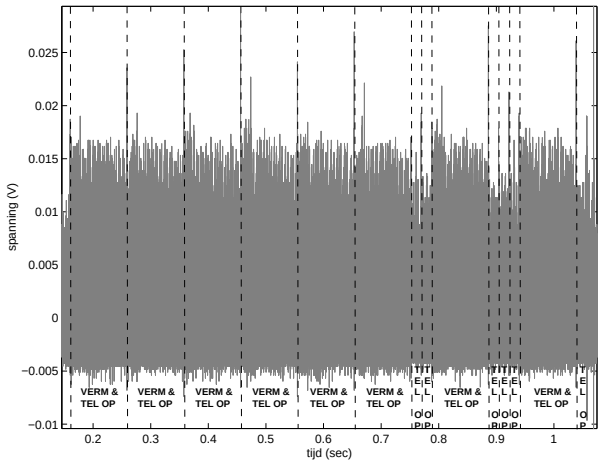
	puntoptelling	puntverdubbeling
1	VERM	VERM en TEL OP
2	VERM	VERM en TEL OP
3	VERM en TEL OP	VERM en TEL OP
4	VERM	VERM en TEL OP
5	VERM en TEL OP	VERM en TEL OP
6	VERM	VERM en TEL OP
7	VERM en TEL OP	TEL OP
8	VERM en TEL OP	TEL OP
9	VERM	VERM en TEL OP
10	VERM en TEL OP	TEL OP
11	VERM en TEL OP	TEL OP
12	VERM	TEL OP
13	VERM en TEL OP	VERM en TEL OP
14	VERM	TEL OP

Elk van de twee bewerkingen bestaat dus uit 14 stappen, die telkens bestaan uit één of twee basisoperaties. Deze stappen zijn zeer duidelijk terug te vinden in de vermogencurven van de beide bewerkingen. De overgang tussen twee stappen wordt namelijk gekenmerkt door een grote vermogenpiek, veroorzaakt door het wegschrijven van de resultaten van de voorgaande stap en het klaarzetten van de ingangen van de volgende. Figuur 6.3 toont het resultaat van een vermogenmeting tijdens een 160-bit puntoptelling radix 1 (klokfrequentie 5 kHz, oscilloscoop 4M metingen aan 2,5M metingen per seconde). De verschillende operaties zijn op de plot van elkaar gescheiden door stippellijnen.



Figuur 6.3: Vermogencurve voor 160-bit puntoptelling radix 1, klokfrequentie 5 kHz, 'Tel op' staat voor een modulaire optelling, 'Verm' voor een Montgomery vermenigvuldiging. De pijltjes duiden op de vermogenpieken bij het wegschrijven van de tussenresultaten. De stippellijnen scheiden de opeenvolgende bewerkingen.

Een 160 bit modulaire optelling vraagt maar 80 klokslagen of 16 msec. Een Montgomery vermenigvuldiging doet er daarentegen 485 klokslagen of bijna 100 msec over. Vermits bij de puntoptelling in elk van de 14 stappen een vermenigvuldiging zit, duurt elke stap even lang (bijna 100 msec). Het verschil tussen een periode met enkel een vermenigvuldiging of een met een optelling en een vermenigvuldiging, is een verhoogd vermogenverbruik aan het begin van de periodes met een optelling. De typische trapezium-vorm van de Montgomery vermenigvuldiging, die wel te zien is in de 'VERM'-stappen, is hier weg. Bovendien kan je hier na 17 msec een piek in het vermogenverbruik zien waar de resultaten van de optelling weggeschreven worden (pijltjes op de figuur).



Figuur 6.4: Vermogencurve voor 160-bit puntverdubbeling radix 1, klokfrequentie 5 kHz, 'Tel op' staat voor modulaire optelling, 'Verm' voor Montgomery vermenigvuldiging. De stippellijnen scheiden de opeenvolgende bewerkingen.

Figuur 6.4 toont het verbruik voor de puntverdubbeling (klokfrequentie 5kHz, oscilloscoop 4M metingen aan 2,5M metingen per seconde). Bij deze puntverdubbeling is het verschil tussen de verschillende stappen nog duidelijker dan bij de puntoptelling. Het gaat hier om ofwel een Montgomery vermenigvuldiging samen met een optelling, ofwel enkel een optelling. Vermits een optelling veel korter duurt dan een vermenigvuldiging en veel minder vermogen verbruikt (er worden elke klokslag veel minder registers geklokt), zijn deze gemakkelijk te onderscheiden op de plot. Merk ook weer de pieken op in het begin van elke 'Verm & Tel op' stap (vermenigvuldiging & optelling), die het wegschrijven van het resultaat van de optelling aangeven.
Een puntoptelling vraagt $14 * 485 = 6790$ klokslagen of ongeveer 1400 msec (14 160-bit radix1 Montgomery vermenigvuldigingen). Een puntverdubbeling doet er $8*485+6*80 = 4300$ klokslagen of een 900-tal msec over (8 160-bit radix1 Montgomery vermenigvuldigingen en 6 160 bit modulaire optellingen met digit-lengte 4). Tabel 6.2 geeft een overzicht.

Tabel 6.2: Overzicht van de nodige klokslagen voor de verschillende bewerkingen met radix 1

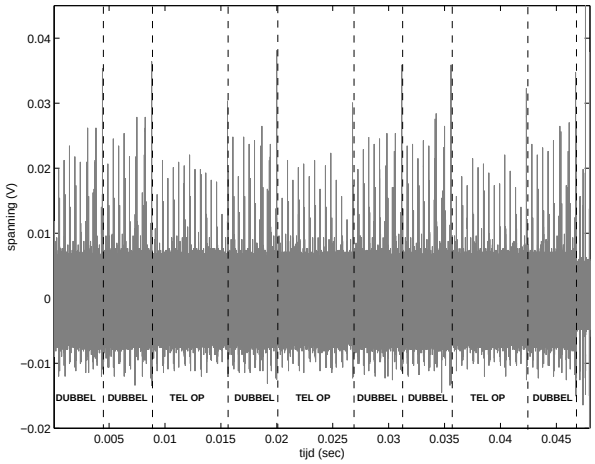
Modulaire optelling	80
Montgomery vermenigvuldiging	485
Puntverdubbeling	4300
Puntoptelling	6790

6.2.4 Puntvermenigvuldiging

Met de kennis van het vermogenverbruik van de puntoptelling en -verdubbeling is het volgende doel de puntvermenigvuldiging. Zoals in de Paragrafen 1.4.3 en 3.2.6 beschreven, wordt een vooraf afgesproken punt vermenigvuldigd met een geheime scalar. Een aanval op het systeem heeft als doel deze scalar, de geheime sleutel, te achterhalen. In elke iteratiestap i van de for-lus wordt een puntverdubbeling uitgevoerd. Indien de i-de bit van de scalar k gelijk is aan 1, wordt bij het tussenresultaat nog eens de ingang P opgeteld. Het geheim van het systeem, de scalar, kan dus achterhaald worden door na te gaan in welke iteratiestappen er wel en in welke er geen puntoptelling plaatsvindt.

Figuur 6.5 geeft het vermogenverbruik van één puntvermenigvuldiging. De scalar was $'0x5a'$, of binair $'1011010'$, de klokfrequentie 1MHz, de oscilloscoop nam 8M metingen aan 100M metingen per seconde. De grootste vermogenpieken geven het einde van een puntoptelling of een puntverdubbeling aan. Op die ogenblikken worden er immers steeds 4 keer 160 bits weggeschreven (de X-coördinaat, Y-coördinaat, Z-coördinaat en aZ^4-coördinaat van het tussenresulaat). Door het tijdsinterval tussen twee van deze pieken te observeren, is het eenvoudig vast te stellen welke operaties puntoptellingen en welke -verdubbelingen zijn. Deze laatste zijn namelijk veel korter dan puntoptellingen. Figuur 6.5 toont de opgemeten vermogencurve. Tabel 6.3 zet de sequentie van operaties die we op de plot terugvinden op een rijtje en leidt de scalar eruit af.

Het puntvermenigvuldigingsalgoritme 3.2.6.1 gaat ervan uit dat de eerste bit van de scalar 1 is en beschouwt deze verder niet. Vooraan aan de gevonden sequentie in Tabel 6.3 dient dus nog een 1 toegevoegd te worden. Dit geeft ons de bitstring $'1011010'$, wat inderdaad de geheime waarde van de scalar $'0x5a'$ is. Het algoritme, zoals het nu geïmplementeerd is, is dus onveilig voor de eenvoudige vermogenanalyse. Paragraaf 6.4 zal daarom enkele aanpassingen op het algoritme bespreken.



Figuur 6.5: Vermogencurve voor 160-bit puntvermenigvuldiging radix 1 met scalar $= '0x5a'$, klokfrequentie 1 MHz, 'Dubbel' staat voor puntverdubbeling, 'Tel op' voor puntoptelling. De stippellijnen scheiden de opeenvolgende bewerkingen.

Tabel 6.3: Sequentie van puntoptellingen en -verdubbelingen bij vermenigvuldiging met scalar $'0x5a'$

PUNTVERDUBBELING	0
PUNTVERDUBBELING PUNTOPTELLING	1
PUNTVERDUBBELING PUNTOPTELLING	1
PUNTVERDUBBELING	0
PUNTVERDUBBELING PUNTOPTELLING	1
PUNTVERDUBBELING	0

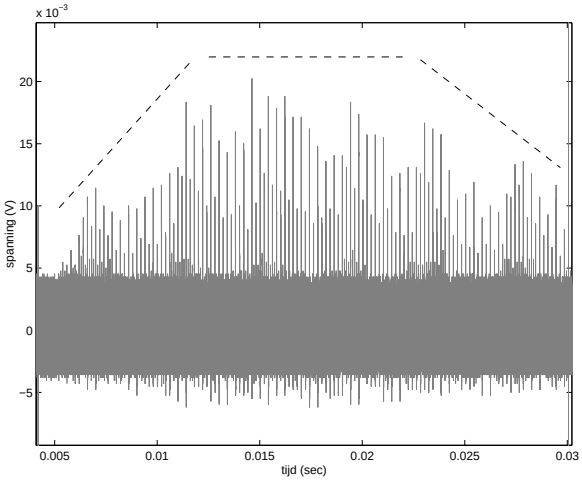
6.3 Metingen op de radix 4-implementatie

Hoofdstuk 3 behandelt twee implementaties van de Montgomery vermenigvuldiger. Het verschil tussen deze twee is de radix, ofwel de basis van de bewerkingen. Naast de klassieke radix 1 Montgomery vermenigvuldiging, is de radix 4 Montgomery vermenigvuldiging (zie Paragraaf 3.2.3) de tweede implementatie. De Paragrafen 6.3.1 tot 6.3.3 bestuderen de vermorgencurves van de radix 4 vermenigvuldiger zelf en de puntoptelling, -verdubbeling en -vermenigvuldiging die van het radix 4 Montgomery vermenigvuldigingsalgoritme gebruik maken.

6.3.1 Montgomery vermenigvuldiging

Tabel 3.4 geeft duidelijk de belangrijkste verschillen tussen de radix 1- en radix 4-implementatie. De vermenigvuldiging radix 4 heeft een kleinere vertraging (indien beide geklokt aan dezelfde frequentie), maar gebruikt een groter aantal transistoren en zal bijgevolg ook meer vermogen verbruiken.
Figuur 6.6 geeft de vermogenplot voor een 160-bit radix 4 Montgomery vermenigvul-

diging. De ingangen en klokfrequentie waren bij de meting ervan dezelfde als voor de meting van de radix 1 vermenigvuldiger (zie Figuur 6.2) (oscilloscoop 4M metingen aan 100M metingen per seconde).



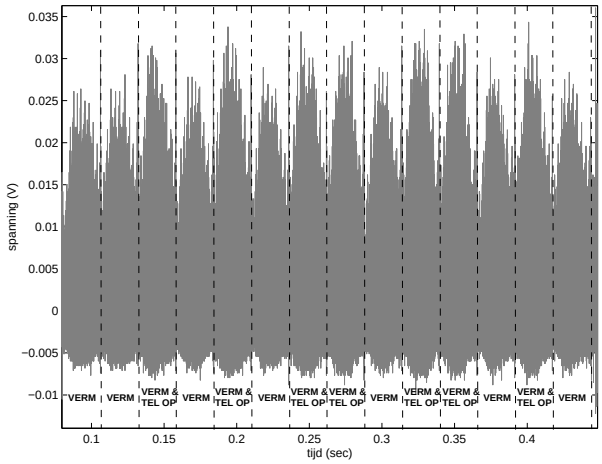
Figuur 6.6: Vermogencurve voor 160-bit Montgomery vermenigvuldiging radix 4, klokfrequentie 5 kHz. De stippellijn geeft de trapezium-vorm van het vermogenverbruik weer.

Deze radix 4 Montgomery vermenigvuldiger doet er 122 klokslagen (of 24,4 msec) over om tot een resultaat te komen. Op elk van deze klokslagen wordt het tussenresultaat (T) in een 160-bit register geklokt en weer als ingang aan de systolische reeks aangelegd. De plot toont duidelijk de 122 pieken die hiervan het gevolg zijn. Ook de typische 'trapezium'-vorm van het vermogenverbruik van een Montgomery vermenigvuldiger (zie Paragraaf 6.2.2) is te onderscheiden. Voor radix 4 is de vermogenplot veel beter interpreteerbaar dan de overeenkomstige plot voor radix 1. De verschillende pieken zijn duidelijker te onderscheiden (doordat we slechts 122 klokslagen plotten) en zijn bovendien hoger. Dit laatste wordt veroorzaakt door het bijna vier maal groter aantal transistoren bij een radix 4 implementatie (zie opnieuw Tabel 3.4).

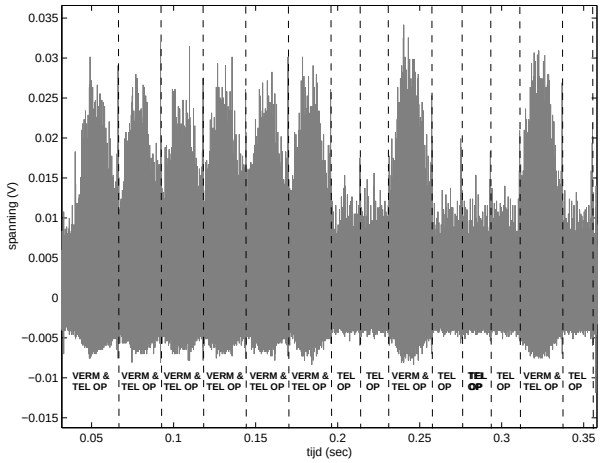
6.3.2 Puntoptelling en puntverdubbeling

Door het grotere vermogenverbruik van de Montgomery vermenigvuldiger zal het uitzicht van de vermogencurves van de puntoptelling en -verdubbeling wijzigen. Figuur 6.7 toont het resultaat van een meting op de radix 4 puntoptelling. Ook hier waren de ingangen en klokfrequentie dezelfde als bij de overeenkomstige Figuur 6.3 voor radix 1 (oscilloscoop 4M metingen aan 5M metingen per sec).

Anders dan bij de radix 1 implementatie, hebben de stappen met een vermenigvuldiging en een modulaire optelling hier wel nog de typische 'trapezium'-vorm. Het vermogenverbruik van één modulaire optelling is niet groot genoeg om de vermogencurve van de vermenigvuldiging zichtbaar te wijzigen. Het is toch nog mogelijk de 'Verm'-stappen (vermenigvuldiging) te onderscheiden van de 'Verm & Tel op'-stappen. Het vermogenverbruik in deze laatste ligt beduidend hoger. De duur van een modulaire optelling en een radix 4 Montgomery vermenigvuldiging liggen nu zeer dicht bij elkaar (80, respectievelijk 122 klokslagen), waardoor de optelling niet enkel bijdraagt tot het vermogen in het begin van de stap, maar in heel de stap (zie Figuur 6.7).



Figuur 6.7: Vermogencurve voor 160-bit puntoptelling radix 4, klokfrequentie 5 kHz, 'Tel op' staat voor modulaire optelling, 'Verm' voor Montgomery vermenigvuldiging. De stippellijnen scheiden de opeenvolgende bewerkingen.



Figuur 6.8: Vermogencurve voor 160-bit puntverdubbeling radix 4, klokfrequentie 5 kHz, 'Tel op' staat voor modulaire optelling, 'Verm' voor Montgomery vermenigvuldiging. De stippellijnen scheiden de opeenvolgende bewerkingen.

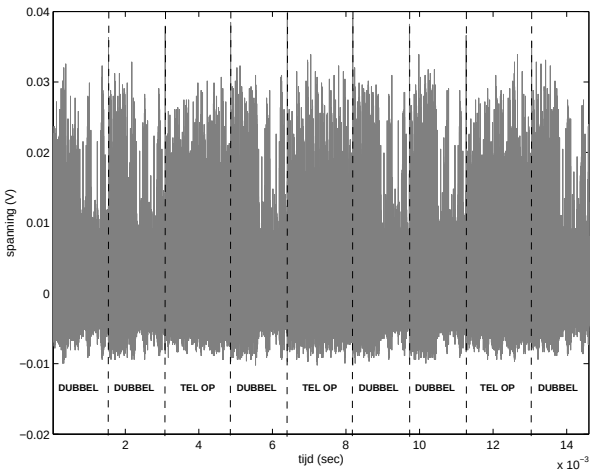
Het grote verschil in het verbruikte vermogen voor een modulaire opteller en een vermenigvuldiger komt nog sterker naar voren bij de puntverdubbeling. Hier hebben we immers stappen met enkel een optelling. Op Figuur 6.8 zijn dan ook duidelijk de 'Verm & Tel op'-stappen te onderscheiden van de 'Tel op'-stappen (160-bit puntverdubbeling, klok aan 5 kHz, oscilloscoop 4M metingen aan 10M metingen per seconde). Het verschil in duur tussen een puntoptelling en een -verdubbeling, dat bij radix 1 zo opvallend was, is hier veel kleiner. Reden is natuurlijk dat een Montgomery vermenigvuldiging radix 4 een vertraging heeft die in de buurt ligt van de vertraging van de modulaire optelling. De lengte van een puntoptelling en puntverdubbeling is nu ongeveer gelijk aan 1750 respectievelijk 1500 klokslagen. Tabel 6.4 geeft een overzicht.

Tabel 6.4: Overzicht van de nodige klokslagen voor de verschillende bewerkingen met radix 4

Modulaire optelling	80
Montgomery vermenigvuldiging	122
Puntverdubbeling	1500
Puntoptelling	1750

6.3.3 Puntvermenigvuldiging

Figuur 6.9 visualiseert een plot van het opgemeten vermogen tijdens een radix 4 160-bit puntvermenigvuldiging (klokfrequentie 100 kHz, oscilloscoop 8M metingen aan 100M metingen per sec). De duidelijke 'gaten' in het vermogenverbruik van de puntverdubbeling wanneer een optelling plaatsvindt, helpen om deze puntverdubbeling te onderscheiden van de puntoptelling. Het verschil in lengte tussen de twee is immers nog wel waarneembaar, maar veel minder uitgesproken dan bij radix 1.



Figuur 6.9: Vermogencurve voor 160-bit puntvermenigvuldiging radix 4 (sleutel = '0x5a'), klokfrequentie 100 kHz, 'Dubbel' staat voor puntverdubbeling, 'Tel op' voor puntoptelling. De stippellijnen scheiden de opeenvolgende bewerkingen.

De vermogenplot vertoont opnieuw dezelfde sequentie van operaties als in Figuur 6.5 (puntvermenigvuldiging radix 1). Ook hier kan de geheime scalar, "$0x5a$", eenvoudig afgelezen worden.

6.4 Wegwerken van de zwakheden

In de vorige paragrafen kwam aan het licht dat onze implementatie niet bestand is tegen een eenvoudige vermogenanalyse. Met één enkele vermogenplot is het al mogelijk de geheime scalar te bepalen. Dit is onaanvaardbaar. Deze paragraaf zal daarom een aantal wijzigingen op het naïeve puntvermenigvuldigingsalgoritme voorstellen, die

het SPA-resistent maken.[1] Enkele andere wijzigingen, die de veiligheid van dit cryptosysteem ten goede moeten komen, waren reeds in de implementatie van Hoofdstuk 3 ingebouwd: Paragraaf 3.2.2 en 3.2.3 bespraken hoe in de modulaire optelling en de Montgomery vermenigvuldiging een onveilige conditionele sprong weggewerkt wordt (Algoritme 3.2.2.1 respectievelijk Algoritme 3.2.3.1).

6.4.1 Eerste alternatief: puntoptelling en puntverdubbeling gelijk

In de vermogencurve van de puntvermenigvuldiging waren de puntoptellingen te onderscheiden van de -verdubbelingen, doordat ze

- een andere vorm hadden
- een verschillende lengte hadden.

Wegwerken van dit onderscheid, maakt het onmogelijk de opeenvolging van puntoptellingen en -verdubbelingen detecteren. Deze twee operaties krijgen hetzelfde uitzicht door in elk van hun 14 stappen telkens een modulaire optelling en een vermenigvuldiging uit te voeren. Indien één van de twee niet aanwezig is in het oorspronkelijke Algoritme 3.2.5.1, moet er dus een 'blinde' bewerking (Eng. dummy) ingevoerd worden, die zijn resultaat wegschrijft in een ongebruikt register. In het aangepaste Algoritme 6.4.1.1 zijn deze blinde bewerkingen vet gedrukt.

Er zijn dus 7 modulaire optellingen toegevoegd voor de puntoptelling en 6 Montgomery vermenigvuldigingen voor de puntverdubbeling. De ingangen van deze blinde bewerkingen zijn volledig willekeurig en er zijn natuurlijk nog tal van andere oplossingen mogelijk. De enige voorwaarden zijn dat in elke stap één optelling en één vermenigvuldiging plaatsvinden en dat de blinde bewerkingen de andere bewerkingen niet storen (geen gebruikte registers overschrijven,...).
Figuur 6.10 en Figuur 6.11 laten de vermogencurves zien van de implementatie van deze algoritmes met behulp van een radix 1 Mongtomery vermenigvuldiger (klokfrequentie 1 MHz, oscilloscoop 4M metingen aan 100M metingen per seconde).

Zoals te zien op deze figuren vallen de twee operaties met het blote oog niet meer van elkaar te onderscheiden. Ze tellen beide 14 stappen met een optelling en vermenigvuldiging.

Figuur 6.12 toont een vermogenplot van de puntvermenigvuldiging gebaseerd op deze aangepaste puntoptelling en -verdubbeling (radix 1, klokfrequentie 1 MHz, oscilloscoop 8M metingen aan 100M metingen per seconde, scalar $'0x5a'$).

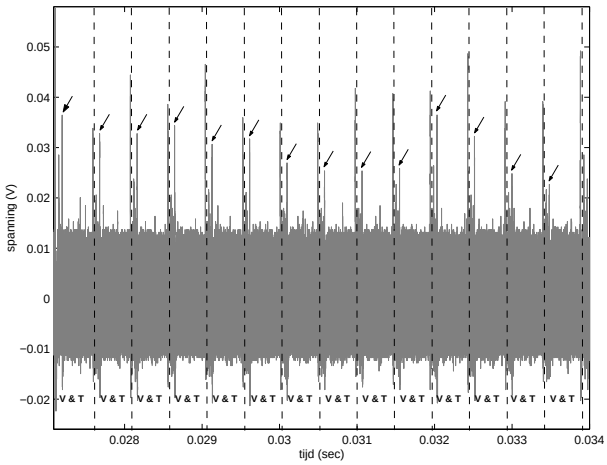
De grote pieken geven nog steeds de overgang van de ene naar de andere operatie (puntoptelling of -verdubbeling) aan, maar het valt niet meer uit te maken welke operaties het zijn.
Indien we de lengte van de scalar kennen, is het aantal $'1'$-en dat de geheime scalar bevat (Eng. Hamming weight) wel nog af te leiden uit de curve. Stel bijvoorbeeld dat geweten is dat de sleutel horende bij Figuur 6.12 zeven bits telt. Op de plot zijn negen verschillende bewerkingen te onderscheiden. Vermits een vermenigvuldiging met een scalar met lengte zeven zes verdubbelingen vraagt (de eerste bit wordt niet beschouwd in het algoritme, zie Algoritme 3.2.6.1), gebeuren er in totaal $9-6=3$ puntoptellingen.

[1] Elk van de twee hierna voorgestelde alternatieven is opgemeten op de radix 1-implementatie van Algoritme 3.2.3.1.

Algoritme 6.4.1.1 : Aangepaste puntoptelling en -verdubbeling

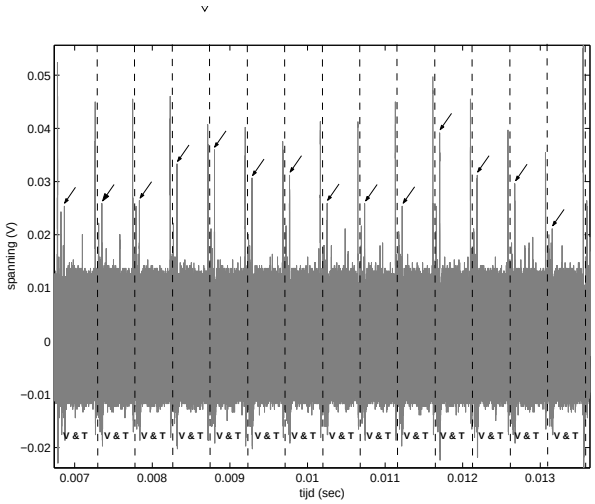
INPUT: $P_1 = (X_1, Y_1, 1, a)$	INPUT: $P_1 = (X_1, Y_1, Z_1, aZ_1^4)$
$P_2 = (X_2, Y_2, Z_2, aZ_2^4)$	
OUTPUT: $P_1 + P_2 = P3$	OUTPUT: $2P_1 = P3$
$= (X_3, Y_3, Z_3, aZ_3^4)$.	$= (X_3, Y_3, Z_3, aZ_3^4)$

1. $T_1 \leftarrow Z_2^2$	$\boldsymbol{T_2 \leftarrow 2Z_2}$	1. $T_1 \leftarrow Y_1^2$	$T_2 \leftarrow 2X_1$
2. $T_2 \leftarrow X_1T_1$	$\boldsymbol{T_3 \leftarrow 2T_1}$	2. $T_3 \leftarrow T_1^2$	$T_2 \leftarrow 2T_2$
3. $T_1 \leftarrow T_1Z_2$	$T_3 \leftarrow X_2 - T_2$	3. $T_1 \leftarrow T_2T_1$	$T_3 \leftarrow 2T_3$
4. $T_1 \leftarrow Y_1T_1$	$\boldsymbol{T_4 \leftarrow 2T_2}$	4. $T_2 \leftarrow X_1^2$	$T_3 \leftarrow 2T_3$
5. $T_4 \leftarrow T_3^2$	$T_5 \leftarrow Y_2 - T_1$	5. $T_4 \leftarrow Y_1Z_1$	$T_3 \leftarrow 2T_3$
6. $T_2 \leftarrow T_2T_4$	$\boldsymbol{T_6 \leftarrow 2T_2}$	6. $T_5 \leftarrow T_3(aZ_1^4)$	$T_6 \leftarrow 2T_2$
7. $T_4 \leftarrow T_4T_3$	$T_6 \leftarrow 2T_2$	7. $\boldsymbol{T_6 \leftarrow T_2^2}$	$T_2 \leftarrow T_6 + T_2$
8. $Z_3 \leftarrow Z_2T_3$	$T_6 \leftarrow T_4 + T_6$	8. $\boldsymbol{Z_3 \leftarrow T_3^2}$	$T_2 \leftarrow T_2 + (aZ_1^4)$
9. $T_3 \leftarrow T_5^2$	$\boldsymbol{X_3 \leftarrow 2T_5}$	9. $T_6 \leftarrow T_2^2$	$Z_3 \leftarrow 2T_4$
10. $T_1 \leftarrow T_1T_4$	$X_3 \leftarrow T_3 - T_6$	10. $\boldsymbol{X_3 \leftarrow T_1^2}$	$T_4 \leftarrow 2T_1$
11. $T_6 \leftarrow Z_3^2$	$T_2 \leftarrow T_2 - X_3$	11. $\boldsymbol{T_6 \leftarrow T_4^2}$	$X_3 \leftarrow T_6 - T_4$
12. $T_3 \leftarrow T_5T_2$	$\boldsymbol{T_4 \leftarrow 2T_6}$	12. $\boldsymbol{T_4 \leftarrow T_1^2}$	$T_1 \leftarrow T_1 - X_3$
13. $T_6 \leftarrow T_6^2$	$Y_3 \leftarrow T_3 - T_1$	13. $T_2 \leftarrow T_2T_1$	$aZ_3^4 \leftarrow 2T_5$
14. $aZ_3^4 \leftarrow aT_6$	$\boldsymbol{T_5 \leftarrow 2T_3}$	14. $\boldsymbol{T_5 \leftarrow T_3^2}$	$Y_3 \leftarrow T_2 - T_3$
15. Return X_3		15. Return X_3	
16. Return Y_3		16. Return Y_3	
17. Return Z_3		17. Return Z_3	

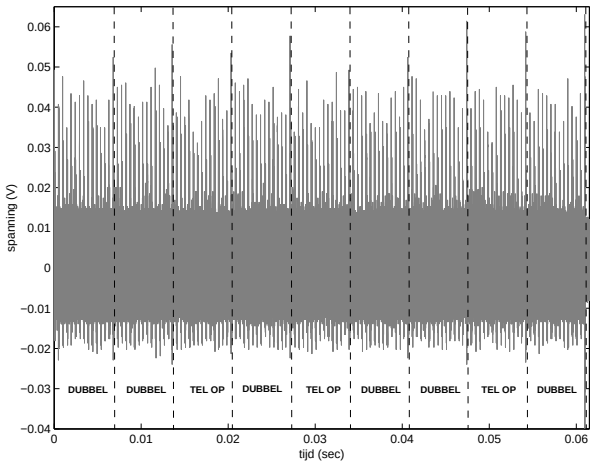


Figuur 6.10: Vermogencurve voor 160-bit aangepaste (alternatief 1) puntoptelling radix 1, klokfrequentie 1 MHz, 'T' staat voor modulaire optelling, 'V' voor Montgomery vermenigvuldiging. De pijltjes duiden op de vermogenpieken bij het wegschrijven van de tussenresultaten. De stippellijnen scheiden de opeenvolgende bewerkingen.

We weten dus dat de scalar 3+1 (de eerste bit) = 4 '1'-bits bevat. Deze implementatie lekt dus nog steeds informatie, zij het veel minder dan in haar oorspronkelijke vorm.



Figuur 6.11: Vermogencurve voor 160-bit aangepaste (alternatief 1) puntverdubbeling radix 1, klokfrequentie 1 MkHz, 'T' staat voor modulaire optelling, 'V' voor Montgomery vermenigvuldiging. De pijltjes duiden op de vermogenpieken bij het wegschrijven van de tussenresultaten. De stippellijnen scheiden de opeenvolgende bewerkingen.



Figuur 6.12: Vermogencurve voor 160-bit aangepaste (alternatief 1) puntvermenigvuldiging radix 1, klokfrequentie 1 MHz, 'Dubbel' staat voor puntverdubbeling, 'Tel op' voor puntoptelling. De stippellijnen scheiden de opeenvolgende bewerkingen.

Door de blinde bewerkingen verbruikt dit alternatief bovendien meer vermogen en heeft het een grotere vertraging dan het origineel.

6.4.2 Tweede alternatief: altijd puntoptelling en puntverdubbeling

In plaats van het algoritme voor de puntoptelling en -verdubbeling aan te pakken, is het ook mogelijk een aanpassing te doen aan het algoritme voor de puntvermenigvul-

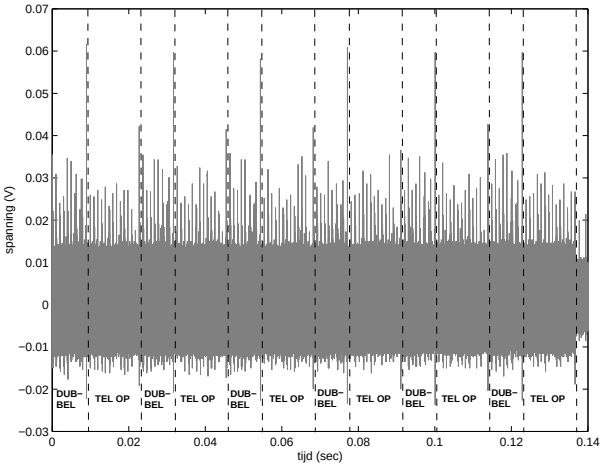
diging. Dit bevat immers de data-afhankelijke sprongconditie, die het aflezen van de scalar-bits uit de vermogencurve mogelijk maakt.
Zoals vermeld in Paragraaf 4.1, is een sprongconditie onveilig als in de ene ('if'-) tak een operatie wordt uitgevoerd met een verschillende duur of een verschillend vermogenverbruik dan in de andere ('else'-) tak en de sprongconditie bovendien data-afhankelijk is. Een data-afhankelijke sprongconditie is in dit geval onvermijdelijk voor de correcte werking van het algoritme (zie Algoritme 3.2.6.1). De enige oplossing om het puntvermenigvuldigingsalgoritme toch veilig te maken is dus zorgen dat de twee takken ('if' en 'else') een zelfde vermogenverbruik kennen. Algoritme 6.4.2.1 zorgt hiervoor.

Algoritme 6.4.2.1 : Aangepaste puntvermenigvuldiging

INPUT: Een punt P_1, geheel getal k, $0 < k < M$, $k = (k_{l-1}, k_{l-2}, \ldots, k_0)_2$,
$k_{l-1} = 1$ en M
OUTPUT: $Q = [k]P$
1. $Q \leftarrow P$
2. **for** i from $l-2$ to 0 **do**:
3. $Q_1 \leftarrow 2Q$
4. $Q_2 \leftarrow Q_1 + P$
5. **if** $k_i = 1$ **then**
6. $Q \leftarrow Q_2$
7. **else**
8. $Q \leftarrow Q_1$
9. **end if**
10. **end for**
11. Return Q

Zowel in de 'if'-tak als in de 'else'-tak worden data weggeschreven. Het enige verschil tussen deze twee is de variabele, die opgeslagen wordt. In de 'if'-tak is dit de variabele Q_2, de uitgang van de puntoptelling. In de andere tak (waar in het oorspronkelijke algoritme helemaal niets gebeurt) is dit een variabele Q_1, de uitgang van de puntverdubbeling. Met dit algoritme wordt er dus in elke iteratiestap van de forlus een puntverdubbeling én een puntoptelling gedaan. De vermogencurve geeft geen informatie aan de aanvaller over het wegschrijven van het resultaat vanuit Q_1 of Q_2. Met behulp van SPA kan een aanvaller niets meer te weten komen van de geheime scalar (tenzij de lengte ervan). Dit algoritme is volledig veilig voor de eenvoudige vermogenanalyse.

Figuur 6.13 visualiseert een vermogencurve van de meting van de implementatie van een 160-bit puntvermenigvuldiging, gebaseerd op Algoritme 6.4.2.1 (klokfrequentie 500 kHz, oscilloscoop 8M metingen aan 50M metingen per seconde, scalar $'0x5a'$).
Uit de figuur valt af te leiden dat er afwisselend een puntverdubbeling en een puntoptelling plaatsvindt en dit zes maal achter elkaar. De enige informatie die hieruit lekt, is dat de scalar zeven bits telt. Verder tasten we volledig in het duister.
Het nadeel van dit algoritme is dat het de puntvermenigvuldiging veel trager maakt. De puntoptelling, die al de langste van de twee operaties is, wordt nu veel vaker uitgevoerd. Dit zorgt natuurlijk ook voor een heleboel extra vermogenverbruik, een verlies aan performantie. Dit is de prijs die betaald moet worden voor de verhoogde veiligheid. Alternatief 1 (Paragraaf 6.4.1), wat minder veilig is, brengt immers een veel



Figuur 6.13: Vermogencurve voor 160-bit aangepaste (alternatief 2) puntvermenigvuldiging radix 1, klokfrequentie 500 kHz, 'Dubbel' staat voor puntverdubbeling, 'Tel op' voor puntoptelling. De stippellijnen scheiden de opeenvolgende bewerkingen.

kleinere extra vertraging met zich mee.

Tot slot geeft Tabel 6.5 een overzicht van de behaalde resultaten voor de verschillende bewerkingen. De Montgomery vermenigvuldiging gebeurde steeds met radix 1. Alle bewerkingen werden uitgevoerd met 160 bit woorden. Aangezien voor tijds- en vermogenanalyse de functionaliteit primeert boven snelheid, is bij alle implementaties enkel voor gezorgd dat het geheel op de gebruikte FPGA past. Van zodra de implementatie paste op de FPGA, werd geen aandacht besteed aan verdere optimalisatie van snelheid en oppervlaktegebruik.

Tabel 6.5: Overzicht van resultaten voor de implementaties van verschillende bewerkingen. Iedere bewerking gebeurt met 160 bits: gebruikte aantal bitslices (# b), het equivalente aantal poorten (# p), het percentage gebruikte chipoppervlakte (opp.) en de maximale klokfrequentie (max freq)

	Radix 1			
Implementatie	# b	# p	opp.	max freq
Montg. Vermenigvuldiging	1 684	37 216	17,8 %	51,5 MHz
Puntoptelling	5 820	128 900	61,8 %	20,65 MHz
Puntverdubbeling	5 490	121 329	58,3 %	
Puntvermenigvuldiging	6 912	152 755	73,4 %	
Alternatief 1	7 265	151 422	77 %	34,4 MHz
Alternatief 2	7 813	158 196	83 %	21 MHz

6.5 Besluit

De inleiding van dit hoofdstuk schetst uitgebreid het principe van de eenvoudige vermogenanalyse. Vervolgens gaven twee delen de resultaten van de vermogenaanvallen

toegepast op een radix 1- en een radix 4-implementatie. Voor elk deel zijn de aanvallen uitgevoerd op de modulaire optelling, de Montgomery vermenigvuldiging, de puntoptelling, -verdubbeling en -vermenigvuldiging. Beide implementaties bleken niet bestand tegen de eenvoudige vermogenaanval. De geheime sleutel kon van de vermogenplot afgeleid worden. Een laatste paragraaf illustreert tegenmaatregelen om het cryptosysteem resistent te maken tegen eenvoudige vermogenanalyse.

Er bestaat echter nog een krachtigere methode om een systeem aan te vallen met behulp van vermogenmetingen: differentiële vermogenanalyse (differential power analysis, DPA). Deze techniek is het onderwerp van Hoofdstuk 7.

Hoofdstuk 7

Differentiële vermogenanalyse in $GF(p)$

7.1 Inleiding

Het naïeve algoritme voor de puntvermenigvuldiging van Paragraaf 2.3.4 bleek in Hoofdstuk 6 niet veilig voor de eenvoudige vermogenanalyse. Twee verschillende implementaties van het algoritme in Paragrafen 6.4.1 en 6.4.2 vormden elk een veilig alternatief. Om de geheime sleutel nu nog te kunnen achterhalen, is dus een krachtigere aanval dan SPA nodig. De differentiële vermogenanalyse (Eng: differential power analysis, DPA) is hiervoor een goede kandidaat. Het doel van de aanval zal de tweede alternatieve implementatie (Algoritme 6.4.2.1) uit Hoofdstuk 6 zijn. De uitvoering van DPA op andere implementaties verloopt analoog.

Het gebruikte puntvermenigvuldigingsalgoritme zal onafhankelijk van de scalar k in iedere stap steeds een puntverdubbeling en een puntoptelling uitvoeren. De bits van k bepalen enkel of de variabele Q_1 of Q_2 de beginwaarde is voor de volgende iteratie van de *for*-lus. Op die manier blijft het tussenresultaat Q in elke iteratie van de *for*-lus toch afhankelijk van de bits van de scalar. Het zijn dan ook deze tussenresultaten die het onderwerp van de aanval vormen in de differentiële vermogenanalyse.

DPA maakt gebruik van zeer vele (enkele honderden tot duizenden) metingen. Dit in tegenstelling tot SPA, waar één enkele meting voldoende is. Terwijl een eenvoudige vermogenanalyse dus slechts enkele minuten duurt, vraagt een differentiële vermogenanalyse al gauw uren tot dagen werk.

Paragraaf 7.2 geeft de verschillende stappen voor DPA. Paragraaf 7.3 overloopt de benodigdheden voor een differentiële vermogenanalyse. Het in de praktijk uitvoeren van DPA zal immers meer bijkomende voorbereidingen vergen dan een eenvoudige vermogenanalyse of een tijdsanalyse. De resultaten van de vermogenaanval zijn het onderwerp van Paragraaf 7.4. Ook nu weer zijn er alternatieve implementaties mogelijk om de DPA-aanvallen op de chip moeilijker te maken. Ze worden besproken in Paragraaf 7.5. Het verschil met SPA is echter dat een chip (op dit ogenblik) niet volledig veilig gemaakt kan worden voor DPA. Men kan enkel proberen de aanval moeilijker te maken zodat het aantal benodigde metingen voor het achterhalen van de sleutel groter wordt.

J.S. Coron beschreef in [10] voor het eerst het principe van DPA op een elliptische kromme puntvermenigvuldigingsalgoritme. We willen hierbij toch opmerken dat Coron

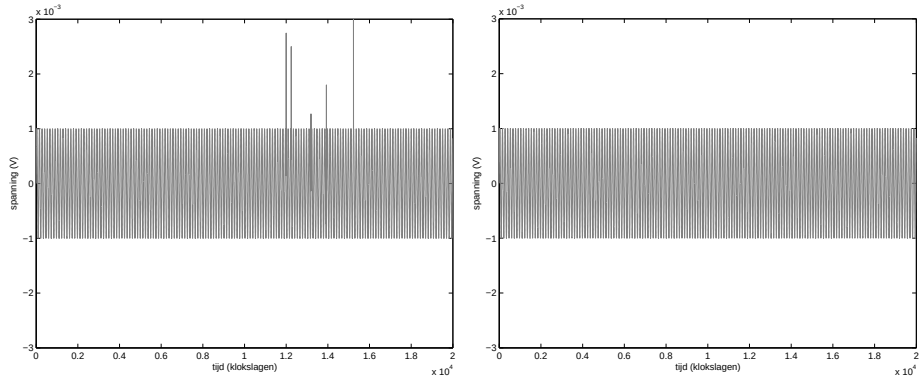
de aanval enkel in theorie beschreef en dat in de literatuur, voor zover bekend, nog nooit melding werd gemaakt van een praktische uitvoering van van differentiële vermogenanalyse op een elliptische kromme cryptosysteem. Dit hoofdstuk bewijst dat DPA op dit systeem ook in de realiteit toepasbaar is en het achterhalen van de geheime sleutel mogelijk maakt. Het uitvoeren van DPA op modulaire exponentiatie algoritmes (zie o.a. [23]) en DES (zie o.a. [18]) werd wel reeds onderzocht.

7.2 Methode

De differentiële vermogenanalyse maakt gebruik van het feit dat tijdens iteratie j van Algoritme 6.4.2.1 de variabele Q_1 enkel afhangt van de eerste $(k_{l-1}, \ldots, k_{l-j-1})$ bits van de scalar k. Aangezien de puntvermenigvuldiging uit opeenvolgende uitvoeringen van puntverdubbelingen en/of puntoptellingen bestaat, is ieder tussenresultaat van $[k]P$ bekomen met een bepaalde unieke volgorde van verdubbelingen en optellingen. Is bijvoorbeeld $k_{l-2} = 1$, dan zal in de tweede uitvoering van de *for*-lus het tussenresultaat Q_1 gelijk zijn aan $[6]P$. De volgorde van de uitgevoerde bewerkingen is hier (V, O, V), waarbij V staat voor puntverdubbeling en O voor puntoptelling. Is de scalarbit k_{l-2} echter 0, dan is dit tussenresultaat $[4]P$ met als volgorde van doorlopen bewerkingen (V, V). Vermits na elke iteratiestap het tussenresultaat in een register wordt weggeschreven, zal het vermogenverbruik van de chip ermee gecorreleerd zijn. Het verschil tussen het vermogenverbruik bij het schrijven van $[4]P$ of $[6]P$ is wegens ruis echter onmogelijk van elkaar te onderscheiden met slechts één meting.
Verschillende puntvermenigvuldigingen worden daarom opgemeten, waarbij telkens de scalar k en de gebruikte kromme dezelfde blijven, maar het ingangspunt P varieert van meting tot meting. Vervolgens wordt er een veronderstelling gemaakt ('orakel') voor de gezochte sleutelbit k_{l-2}. Op basis van deze veronderstelling wordt een selectiefunctie opgesteld, die het geheel van metingen onderverdeelt in twee groepen. Indien de veronderstelling voor k_{l-2} correct is, zou het vermogenverbruik van deze twee groepen significant moeten verschillen. Op deze manier is de veronderstelling geverifieerd en is er uitsluitsel over de gezochte bit van de sleutel.

Stel dat de veronderstelling luidt: $k_{l-2} = 0$. Onder deze veronderstelling klokken de registers van de chip op een bepaald moment het tussenresultaat $[4]P$ binnen. Een voorbeeld van een selectiefunctie is dan het onderverdelen van de metingen in twee groepen op basis van steeds dezelfde n-de bit van $[4]P_i$, (met $i = 0, \ldots,$ *aantal metingen*-1 en $n <$(*aantal bits van* P)-1). De ene groep bevat de metingen waarvoor de n-de bit van $[4]P_i$ een 0 is. De tweede groep bevat de punten waarvoor deze bit een 1 is. Enkel wanneer tijdens het uitvoeren van de puntvermenigvuldiging op de cryptochip het punt $[4]P_i$ ook daadwerkelijk berekend wordt (dus als de $(l-2)$-de scalarbit 0 is), zal het opgemeten vermogenverbruik voor de groep met de n-de bit 1 groter zijn op het moment dat Q_1 wordt berekend dan voor de groep met deze bit 0. Het wegschrijven van een 1 kost immers gemiddeld meer vermogen dan het wegschrijven van een 0. Indien niet $[4]P_i$ maar $[3]P_i$ (en dus ook $[6]P_i$) in de chip berekend wordt, zal het vermogenverbruik van de twee groepen ongeveer hetzelfde zijn. De voorspelde waarden van de n-de bit van $[4]P_i$, waarop de opsplitsing in deelgroepen is gebaseerd, worden namelijk nooit uitgerekend op de cryptochip.

Figuur 7.1(a) en (b) tonen hoe het verschil in vermogenverbruik (gemiddelde groep 1 - gemiddelde groep 0) er in het ideale geval (geen ruis, oneindig aantal metingen) uitziet voor (a) een correcte veronderstelling voor de aangevallen sleutelbit en (b) een foute veronderstelling voor de aangevallen sleutelbit. Het verschil in vermogenverbruik dient dus, indien de veronderstelling correct is (a), pieken te vertonen op de plaats waar de



(a) Correcte veronderstelling $(l-2)$-de sleutelbit

(b) Foutieve veronderstelling $(l-2)$-de sleutelbit

Figuur 7.1: Het verschil in vermogenverbruik voor een correcte en foutieve veronderstelling van de sleutelbit

voorspelde bits berekend worden. In dit geval is dit tussen klokslag 11000 en 15500, het gebied waar de tweede verdubbeling gebeurd ($[4]P$ of $[6]P$ berekend wordt) en er dus meer activiteit moet zijn voor de groep met 1 dan voor de groep met 0. Indien de veronderstelling foutief is (b), zou dit verschil in vermogenverbruik volledig willekeurig moeten zijn en geen pieken mogen vertonen. Dit is natuurlijk enkel zo in het ideale geval, zoals Paragraaf 7.4 duidelijk zal maken.

Het verschil in vermogenverbruik tussen de twee groepen onthult dus de $(l-2)$-de bit van de scalar. Met de kennis van deze bit, kan dan op dezelfde manier de volgende bit aangevallen worden, tot alle bits gekend zijn.

De concrete werkwijze voor het achterhalen van de scalarbit k_{l-2} met deze selectiefunctie is dus als volgt:

- Voer d metingen uit op de cryptochip, met telkens dezelfde scalar k en elliptische kromme, maar met d verschillende ingangspunten $P_1, \ldots, P_d$. $C_i(t)$ is het vermogenverbruik van de i-de meting.
- Voorspel zelf voor elk van deze punten P_i de n-de bit, s_i, van $[4]P_i$.
- Pas de functie $g(t)$ toe op deze groep van metingen, met

  $$g(t) = < C_i >_{i=1,2,\ldots,d|s_i=1} - < C_i >_{i=1,2,\ldots,d|s_i=0}$$

  met $< \ldots >$ de gemiddelde waarde.
- Plot deze functie $g(t)$. Een piek op het tijdstip t van de berekening van de bit s van $[4]P$, geeft aan dat de veronderstelling correct was en dus $k_{l-2} = 0$. Het uitblijven van een piek geeft aan dat $k_{l-2} = 1$.

De volgende bits van k zijn recursief met dezelfde methode te achterhalen.

De hierboven gebruikte selectiefunctie is slechts één van de vele mogelijke functies om de metingen op te delen in twee groepen. Paragraaf 7.4 zal nog enkele andere, betere selectiefuncties voorstellen.

7.3 Benodigdheden voor DPA

Bij de differentiële vermogenanalyse komt veel meer kijken dan bij de eenvoudige vermogenanalyse. De aanvaller moet kennis hebben van zowel statistiek, cryptografie, programmeren, elektronica en meetinstrumentatie. Dit vertaalt zich dan ook in een op elkaar afstemmen van verschillende apparaten en programma's nodig om deze aanval uit te voeren.
Naast het C-programma en de meetopstelling, beschreven in Hoofdstuk 4, wordt om punten op de kromme te generen Magma gebruikt en om de metingen te verwerken Java- en Matlab-code ingezet. Deze verschillende onderdelen worden hieronder verder toegelicht.

7.3.1 Magma

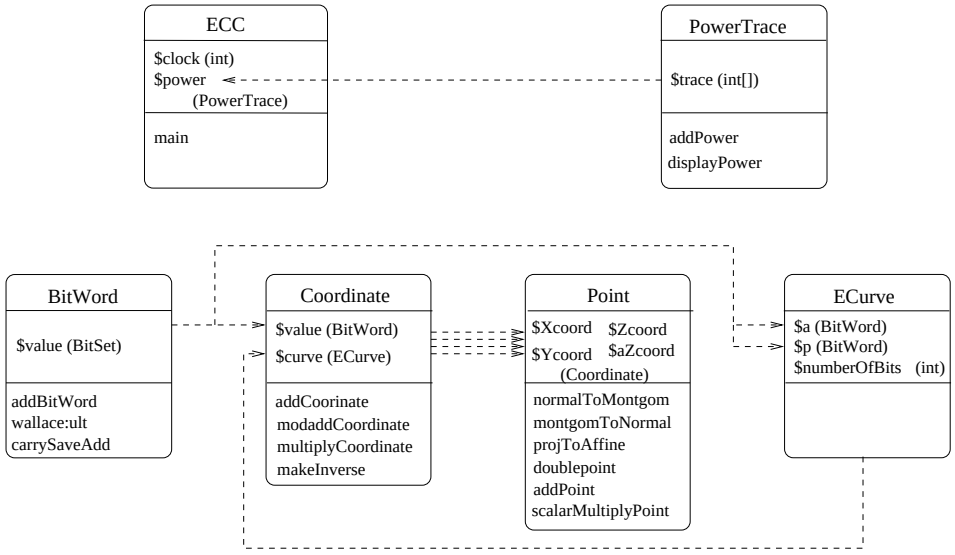
Magma is een 'Computer Algebra Systeem', ontworpen aan de universiteit van Sidney om problemen in algebra, getaltheorie, geometrie en combinatieleer op te lossen. Meer informatie is te vinden in [24]. Het programma kan een elliptische kromme aanmaken en punten op deze curve berekenen. Voor deze thesis werd een kromme (met 160 bit parameters) aangemaakt en werden duizend van haar punten samengebracht in een bestand 'PointsForDPA.txt'.

7.3.2 Visual C

Het C-programma, dat gebruikt werd bij SPA om de communicatie met de FPGA en de oscilloscoop te verzorgen, wordt voor DPA uitgebreid. Niet alleen is het aantal metingen verhoogd tot bijna duizend, maar bovendien vereist iedere meting ook het klaarzetten van de juiste ingangen. Alvorens een meting te starten, zal het C-programma een punt uit het 'PointsForDPA.txt'-bestand halen en dit converteren naar Montgomery formaat. Het schrijft dit resultaat weg in een bestand 'FileForPQi.txt' (waarbij i staat voor het nummer van de meting) en stuurt het tegelijk als ingang op naar de FPGA.
Tijdens de werking van de chip meet de oscilloscoop naast de trigger en de stroom uit de voeding nu ook de klok. Deze zal immers nodig zijn voor de synchronisatie van de verschillende metingen.
Het C-programma voert de hierboven beschreven sequentie van operaties d keer uit. Het resultaat hiervan zijn dus d verschillende 'FileForPQi.txt'-, 'data$i$.t'- en 'clock$i$.t'-bestanden. Deze laatste twee bevatten de opgemeten curve van het verbruikte vermogen en de klok voor meting i. Elk van deze twee bestanden vraagt 7,8 Mbyte aan opslagcapaciteit. Een vermogenaanval met enkele honderden tot duizenden metingen vergt bijgevolg een computer met voldoende geheugen om al de metingen op te slaan.

7.3.3 Java

Eens de verschillende metingen uitgevoerd zijn en de resultaten opgeslagen, dienen ze onderverdeeld te worden in twee groepen op basis van een bepaalde selectiefunctie, toegepast op de bits van een tussenresultaat, bv. $[4]P$. De voorspelling van deze $[4]P$ gebeurt met behulp van een Java-programma. Figuur 7.2 toont het klasse-diagramma van dit elliptische kromme programma.
De klasse *'Bitword'* vormt de basis van het hele systeem. Het overeenkomstige object stelt een woord van bits voor. Deze klasse implementeert fundamentele bewerkingen

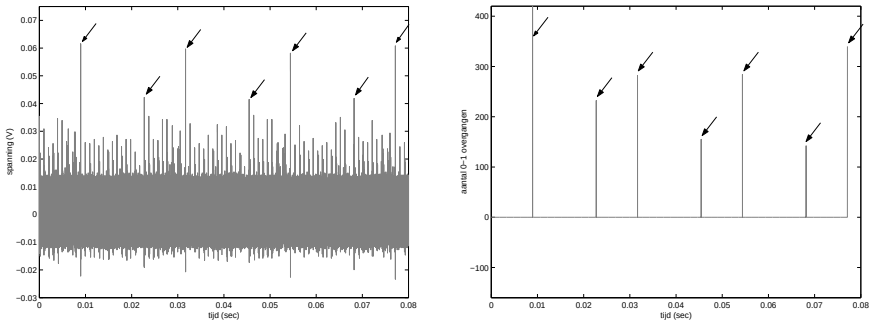


Figuur 7.2: Klasse-diagramma van het Java-programma

zoals vermenigvuldigingen en optellingen van binaire woorden. Van elk object van de klasse *'Coordinate'* wordt de waarde voorgesteld door een BitWord-object. Daarnaast bevat een Coordinate-object ook de kromme (ECurve) waarop het overeenkomstige punt ligt. Het programma kan coördinaten (modulair) optellen, vermenigvuldigen, inverteren,... Nog een trapje hoger op de ladder staan de objecten van de klasse *'Point'*. Zij stellen punten van de elliptische kromme voor, elk gekenmerkt door 4 (X-, Y-, Z- en aZ^4-) coördinaten. Naast de nodige conversies zijn op deze 'Point'-objecten ook de puntoptelling, puntverdubbeling en natuurlijk puntvermenigvuldiging gedefinieerd. Verder wordt het klasse-diagramma aangevuld met de klasse *'ECurve'* (elliptische kromme-objecten), *'PowerTrace'* (om het vermogen te voorspellen, zie verder) en *'ECC'*. Deze laatste is de top klasse, die de methode 'main' bevat. Deze methode geeft de sequentie van de uit te voeren operaties. De volledige Java-code kan gevonden worden in Bijlage D.

Bij het schrijven van het programma is er grote aandacht besteed aan het zo nauwgezet mogelijk volgen van de werking van de chip. Dit houdt in dat de software-implementatie dezelfde operaties, registers, tellers,... gebruikt als de uitvoering op chip. Dit is niet noodzakelijk om $[4]P$ correct te kunnen voorspellen, maar deze extra inspanning maakte wel het nagaan van de correcte functionele werking van de hardware mogelijk. Het stelde ons in staat om, alvorens te beginnen aan SPA, na te gaan of alle deelblokken van onze implementatie de juiste tussenresultaten en uitgangen gaven.
Bovendien was dit programma oorspronkelijk ook bedoeld om nog een derde soort vermogenaanval uit te voeren. Door de curve van het in de software voorspelde vermogen voor een bepaalde sleutel te correleren met de overeenkomstige gemeten curve, dachten we de correctheid van de voorspelde sleutel te kunnen detecteren. Dit bleek echter (met één meting) niet haalbaar. Waarschijnlijk strooien vooral ruis en invloed van de meetapparatuur roet in het eten. Figuur 7.3 vergelijkt een voorspelling van de pieken aan het eind van elke puntoptelling en -verdubbeling met een vermogenmeting voor dezelfde sleutel. De gelijkenis heeft echter geen bewijswaarde. Ook voor andere sleutels zou het resultaat gelijkaardige voorspellingen opleveren. Vermoedelijk zouden verschillende metingen, waarvan het gemiddelde genomen wordt, in plaats van

één enkele meting betere resultaten geven. Wegens tijdgebrek en andere prioriteiten hebben we dit echter niet meer kunnen onderzoeken.



(a) gemeten vermogencurve (b) voorspelde vermogencurve

Figuur 7.3: Vergelijking van de voorspelde en de gemeten vermogencurve voor sleutel $'0x5a'$. De pijltjes op de gemeten vermogencurve duiden op de vermogenpieken bij het wegschrijven van de tussenresultaten.

Tenslotte zou deze hardware-geïnspireerde software-implementatie ook zijn nut kunnen bewijzen bij het schatten van het aantal metingen nodig voor een differentiële vermogenaanval. Zo kan heel snel nagegaan worden of een andere implementatie of een overstap naar een andere soort logica (vb. differentiële logica) extra veiligheid tegen DPA met zich meebrengt. Dit wordt behandeld in [1], maar ligt buiten het doel van dit eindwerk.

De werking van het Java-programma beperkt zich dus tot het aanmaken van het selectie-bestand: Het opent één voor één de 'FileForPQi.txt'-bestanden, leest de ingangen voor de puntvermenigvuldiging in en berekent hiermee het aan te vallen tussenresultaat, bijvoorbeeld $[4]P_i$. Vervolgens bepaalt het programma de selectiebit. Dit is bijvoorbeeld de laatste bit van de X-coördinaat van de Montgomery vorm van $[4]P_i$. Het Java-programma schrijft deze bit voor elk van de ingangspunten P_i weg in een gemeenschappelijk bestand 'Prediction.txt'. Dit bestand bevat dus achteraf een reeks van d voorspelde selectiebits, één voor elke meting i, $i = 0, \ldots, d-1$.

7.3.4 Matlab

Een eigen ontwikkeld Matlab-programmaatje vormt de laatste stap in de differentiële vermogenanalyse. Dit programma zal de meetresultaten in groepen verdelen, ze verwerken en de resultaten visualiseren.
Het leest eerst het bestand 'Prediction.txt' in en zet de inhoud ervan in een matrix. Aan de hand hiervan worden de metingen, waarvoor het Java-programma een 0 voor de selectiebit berekende, één voor één ingeladen, geconverteerd naar Matlab-formaat en gesynchroniseerd. (De opgemeten punten die vallen voor de eerste stijgende klokflank, worden verwijderd.) Wanneer de volledige reeks doorlopen is, schrijft Matlab het gemiddelde van de bewerkte metingen weg in een variabele 'resultaat0'. De metingen, die overeenkomen met een voorspelde 1, worden op dezelfde manier verzameld in de variabele 'resultaat1'. Om het geheugen van Matlab niet te overbelasten, beperkt het programma het aantal variabelen die tegelijk actief zijn.

De twee variabelen 'resultaat0' en 'resultaat1' worden van elkaar afgetrokken om de functie $g(t)$ te bekomen. De plots zijn nog iets duidelijker wanneer alle punten van dit resultaat binnen één klokslag (de oscilloscoop nam zo een 100-tal punten per klokslag) worden samengeteld. Het is op deze plot dat de aan- of afwezigheid van een piek de sleutelbit verraadt. Paragraaf 7.4 licht de resultaten van deze methode toe.

De Matlab- en Java-programma's hierboven beschreven zijn gemaakt door de auteurs van deze thesis. Het C-programma is ontwikkeld in het kader van een vorige thesis ([14]) en is aangepast door Siddika Berna Örs. Magma is een programma van de universiteit van Sidney.

7.4 Metingen

Deze paragraaf behandelt de resultaten van de uitgevoerde metingen voor DPA. Doel van de aanval is steeds het Algoritme 6.4.2.1, met de scalar '0x5a'. Enkel de $(l-2)$-de bit (5de bit) van de scalar zal aangevallen worden. Eens deze gekend is, kunnen de overige bits op dezelfde manier bepaald worden. Voor elk van de hieronder beschreven aanvallen, werd telkens éénmaal deze bit 0 verondersteld (wat correct is) en éénmaal deze bit 1 verondersteld (wat fout is). Dit komt overeen met een voorspelling van $[4]P$ respectievelijk $[6]P$ voor het tussenresultaat Q_1.

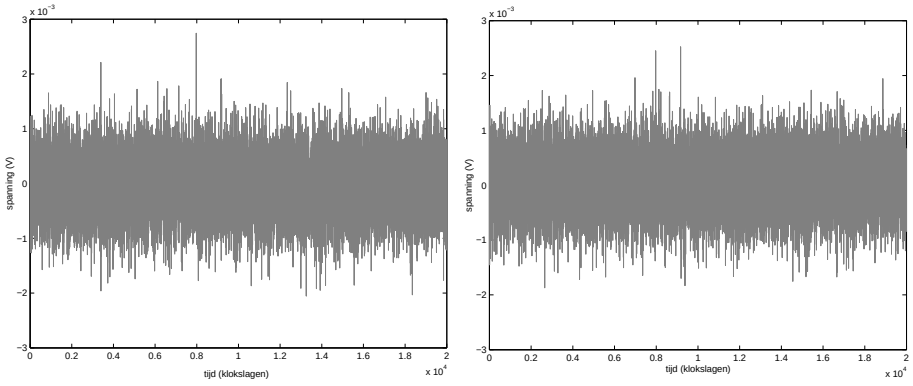
Bij het uitvoeren van een DPA zijn er twee parameters waarmee gevarieerd kan worden: het aantal metingen en de selectiefunctie. Beiden bepalen in grote mate het slagen van de aanval.

Het aantal gebruikte metingen bedraagt voor alle resultaten in deze paragraaf 500. Meer metingen zouden zonder enige twijfel leiden tot een beter (duidelijker) resultaat. Het kost echter zeer veel tijd deze te vergaren ($\pm$3 minuten per meting).

De selectiefunctie bepaalt voor elk van de metingen in welke van de twee verschillende groepen zij thuishoort. (Zie Paragraaf 7.2.) Het is ook mogelijk dat de selectiefunctie aangeeft dat de meting helemaal niet gebruikt zal worden in de verwerking. Het spreekt voor zich dat de keuze van deze selectiefunctie van cruciaal belang is. Deze functie moet er immers voor zorgen dat de correctheid van de voorspelde bit afgeleid kan worden uit het verschil in gemiddeld vermogenverbruik tussen de twee groepen van metingen. Deze paragraaf vergelijkt verschillende selectiefuncties.

Paragraaf 7.2 voerde reeds een eerste selectiefunctie in: De onderverdeling in twee groepen gebeurde op basis van 1 bit van het voorspelde tussenresultaat Q_1. Figuur 7.4 toont het verschil in vermogenverbruik van de twee groepen voor 500 metingen. De opdeling gebeurde voor Figuur 7.4(a) op basis van de laatste bit van de Montgomery voorstelling van $[4]P$ en voor (b) op basis van de laatste bit van de Montgomery voorstelling van $[6]P$.
Vermits de scalar '0x5a' gebruikt werd, verwachten we voor de plot met de voorspelling van $[4]P$ een verhoogde activiteit tussen de 11000ste en 15500ste klokslag ten opzichte van de tweede plot. De plots zijn echter veel minder duidelijk dan de plots voor het ideale geval (Figuur 7.1). Afleiden welke van de twee veronderstellingen correct is, is met deze vermogencurves onmogelijk. Vermoedelijk zouden (veel) meer metingen tot een beter resultaat leiden. De pieken die nog te zien zijn op beide grafieken, kunnen we niet verklaren. Hoogstwaarschijnlijk zijn ze het gevolg van te weinig metingen, waardoor er onvoldoende uitmiddeling gebeurt.

Een selectiefunctie, die slechts gebruikt maakt van 1 bit, is dus duidelijk niet krachtig



(a) Correcte veronderstelling 5de sleutelbit

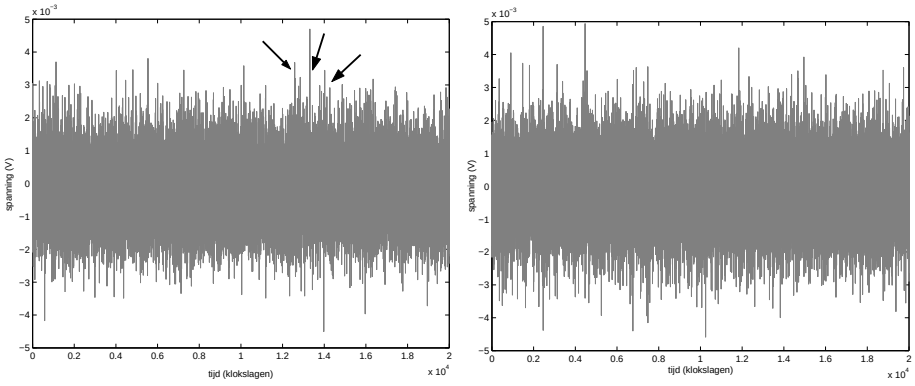
(b) Foutieve veronderstelling 5de sleutelbit

Figuur 7.4: Het resultaat na DPA met als selectiefunctie de laatste bit van de Montgomery voorstelling van de X-coördinaat van $[4]P$ (correcte veronderstelling) respectievelijk $[6]P$ (foute veronderstelling)

genoeg om met 500 metingen conclusies te kunnen trekken. Het is echter mogelijk niet één, maar meerdere bits tegelijk te beschouwen. De selectiefunctie kan bijvoorbeeld zijn: Neem de 4 laatste bits van de X-coördinaat van $[4]P$ en zet een meting in groep 0 indien het Hamming gewicht van deze bits 0 is, in groep 1 indien dit 4 is en gebruik de meting niet in alle andere gevallen. Een dergelijke selectiefunctie is veel krachtiger, vermits zij ervoor zorgt dat er een groter verschil is tussen de twee groepen in het aantal behandelde 1-en. Helaas bleek ook DPA met deze selectiefunctie en 500 metingen geen goede resultaten op te leveren.

De selectiefunctie dient dus te zorgen voor nog een groter verschil in het aantal behandelde 1-en. Dit kan door het Hamming gewicht van de X-, Y-, Z- en aZ^4-coördinaat tegelijk te beschouwen. Gemiddeld zal dit Hamming gewicht 320 (4*160/2) zijn. Laat groep 0 alle metingen met een Hamming gewicht < 305 bevatten en groep 1 die met een Hamming gewicht > 335 en gooi de rest van de metingen weg. Het verschil in vermogenverbruik tussen de twee groepen zal nu groter zijn. Er is immers een verschil van minstens 30 berekende 1-en tussen de twee groepen. Het nadeel van deze selectiefunctie is echter dat er zeer veel metingen weggegooid moeten worden. Van de 500 metingen, die gebruikt werden voor dit experiment, bleven er slechts een 70-tal over in elk van de twee groepen. Toch geeft een DPA met deze selectiefunctie een bevredigend resultaat. Figuur 7.5 toont een lichtjes verhoogde activiteit tussen de klokslagen 11000 en 15500 voor $[4]P$ (correcte voorspelling) ten opzichte van $[6]P$ (foutieve voorspelling). Op deze figuur zijn ook nog pieken op andere klokslagen te zien. Hier hebben we geen verklaring voor en we vermoeden dat deze het gevolg zijn van te weinig metingen.

In deze paragraaf werd aangetoond dat met voldoende metingen en een krachtige selectiefunctie het mogelijk is om uit deze implementatie met behulp van een DPA een sleutelbit te detecteren. Eens deze bit gekend is, is het mogelijk de volgende bit aan te vallen,... DPA is dus in staat de volledige sleutel te achterhalen. Dit lukte hier met slechts 500 metingen. Toch was het resultaat niet zo duidelijk en zijn meer metingen



(a) Correcte veronderstelling 5e sleutelbit

(b) Foutieve veronderstelling 5e sleutelbit

Figuur 7.5: Het resultaat na DPA met als selectiefunctie het Hamming gewicht (< 305 of > 335) van de Montgomery voorstelling van al de coördinaten van $[4]P$ (correcte veronderstelling) respectievelijk $[6]P$ (foute veronderstelling)

wenselijk. DPA, met in het bijzonder het benodigde aantal metingen en de keuze van de selectiefunctie, verdient zeker nog verder onderzoek.
Volgende paragraaf stelt enkele alternatieven voor om deze aanval te verhinderen. Het zal blijken dat het veel moeilijker is een chip bestand te maken tegen DPA dan tegen SPA.

7.5 Wegwerken van de zwakheden

Deze laatste paragraaf beschrijft drie mogelijke alternatieve implementaties, die beter bestand zijn tegen de hierboven toegepaste vorm van DPA. Elk van de drie alternatieven (beschreven in [10]) is gebaseerd op het introduceren van willekeurige getallen tijdens de berekening van $Q = [k]P$. Geen van de alternatieven heeft een significante impact op de efficiëntie van de puntvermenigvuldiging. Toch is er geen garantie dat deze alternatieven veiligheid bieden tegen andere aanvallen.
Het is zelfs niet altijd zo dat ze volledig resistent zijn tegen DPA. De veiligheid van de hieronder beschreven implementaties steunt immers steeds op het genereren van een random getal/bit. Gebeurt dit niet volledig random, dan is de implementatie niet meer veilig en kan ze met behulp van DPA en vele metingen gebroken worden. Goede random getal-generatoren op chip zijn momenteel nog niet beschikbaar.

7.5.1 Eerste alternatief: randomiseren van de geheime sleutel

Stel ϵ het aantal punten op de gebruikte kromme. De berekening van $Q = [k]P$ bestaat dan uit de volgende stappen:

1. Kies een willekeurig getal n met l bits.
2. Bereken $k' = k + n\epsilon$.

3. Bereken het punt $Q' = [k']P$. Dit is gelijk aan $Q = [k]P$, vermits $[\epsilon]P = \mathcal{O}$.

Dit alternatief maakt DPA, zoals beschreven in Paragraaf 7.2, onmogelijk. De geheime scalar verandert immers bij elke nieuwe uitvoering van het algoritme.

7.5.2 Tweede alternatief: afschermen van het punt P

Het te vermenigvuldigen punt P wordt afgeschermd door er een geheim willekeurig punt R bij op te tellen, waarvan we weten dat $S = [k]R$. Met de puntvermenigvuldiging wordt dus het punt $[k](R + P)$ berekend, waarna er $S = [k]R$ vanaf getrokken wordt. Zo bekomt men opnieuw het punt $Q = [k]P$.
De punten R en S zitten eventueel initieel opgeslagen in de cryptochip en worden bij elke uitvoering van het algoritme vervangen. Dit gebeurt als volgt:

$$R \leftarrow [2(-1)^b]R$$

$$S \leftarrow [2(-1)^b]S$$

met b een willekeurige bit, die bij elke uitvoering van het algoritme opnieuw gegenereerd wordt. Dit alternatief maakt van het te vermenigvuldigen punt een onbekende, waardoor DPA uit Paragraaf 7.2 onmogelijk wordt.

7.5.3 Derde alternatief: randomiseren van de projectieve coördinaten

Paragraaf 2.3.3 geeft een argumentatie voor het gebruik van projectieve coördinaten. Naast meer efficiëntere puntoptellingen en -verdubbelingen kunnen deze projectieve coördinaten nog een tweede voordeel geven: beveiliging tegen DPA.
De projectieve coördinaten van een punt (x, y) zijn immers niet uniek. Namelijk

$$(X, Y, Z) = (\lambda X, \lambda Y, \lambda Z)$$

voor elke $\lambda \neq 0$ in het eindig veld. Dit derde alternatief maakt hier handig gebruik van door de projectieve coördinaten van het te vermenigvuldigen punt P met een willekeurige scalar λ te vermenigvuldigen voor elke uitvoering van het algoritme. Dezelfde randomizatie is toepasbaar na elke puntoptelling en -verdubbeling.
Dit alternatief zorgt ervoor dat de aanvaller geen bit van een tussenresultaat Q kan voorspellen, zodat DPA van Paragraaf 7.2 onmogelijk is.

7.6 Besluit

Een uitgebreide beschrijving van het principe van DPA leidt dit hoofdstuk in. Een fundamenteel verschil met SPA bestaat erin dat er voor DPA niet één, maar heel veel metingen nodig zijn. Bijgevolg zijn voor deze aanvallen een meer uitgebreide meetopstelling en diverse dataverwerkingsprogramma's nodig. Deze worden bondig beschreven in Paragraaf 7.3. Paragraaf 7.4 bespreekt de resultaten van DPA met 500 metingen en verschillende selectiefuncties. Het blijkt dat de keuze van de selectiefunctie en het aantal metingen zeer belangrijk is voor het slagen van de aanval. Met behulp van een selectiefunctie op basis van het Hamming gewicht van de 4 coördinaten van het tussenresultaat wordt met succes een bit van de geheime sleutel ontmaskerd. Het aantal gebruikte metingen ligt echter aan de lage kant, waardoor het resultaat niet voldoende duidelijk is. Het hoofdstuk rondt af met een aantal mogelijke alternatieve implementaties, die beter bestand zijn tegen deze differentiële vermogenanalyse.

Hoofdstuk 8

Algemeen besluit en vervolgonderzoek

Dit eindwerk onderzocht voor het eerst de veiligheid van een FPGA-geïmplementeerd cryptosysteem met behulp van tijds- en vermogenanalyse. Het belaagde systeem steunt op een vorm van publieke sleutel cryptografie gebruik makend van elliptische krommen.
Een eigen ontwikkelde VHDL-implementatie op herprogrammeerbare FPGA vormde het doel van de aanvallen. Een tijdsanalyse op een eerste naïëve implementatie toonde aan dat het Hamming-gewicht van de geheime sleutel eenvoudig te achterhalen valt. Met behulp van eenvoudige vermogenanalyse bleek het zelfs mogelijk alle bits van de sleutel af te leiden.
Vervolgens zijn enkele mogelijke alternatieve implementaties gerealiseerd. Deze zijn, zoals de metingen bewijzen, wel resistent tegen de hoger genoemde aanvallen.
Met behulp van de differentiële vermogenanalyse werd echter ook uit deze implementaties met succes de sleutel geëxtraheerd. De tekst besluit met enkele verbeterde implementaties om deze differentiële vermogenaanvallen te omzeilen.

Dit werk bewijst dat de meetresultaten van een vermogenanalyse op FPGA en ASIC overeenkomen. Vermits dit impliceert dat de veiligheid van de twee realisaties dezelfde is, maakt dit de FPGA uiterst geschikt voor het bouwen van prototypes van cryptochips. Een FPGA laat immers toe zeer snel een systeem in ontwikkeling in te laden om na te gaan of het voldoende resistent is tegen aanvallen. Een eventuele aanpassing van de implementatie is eenvoudig aan te brengen. Eens de resultaten tevreden stellen, kan de dure en trage productie van de beoogde ASIC gestart worden.
Dit zelfde proces is ook in dit eindwerk doorlopen. Op korte tijd werd een volledig cryptosysteem geïmplementeerd en getest op FPGA. Verschillende malen werden er wijzigingen aan aangebracht, die de veiligheid verhoogden.

Dit eindwerk werkte een implementatie uit van het bestookte cryptosysteem gebaseerd op velden met even en oneven karakteristiek. De vermenigvuldigingen in velden met oneven karakteristiek kregen een implementatie met radix 1 en radix 4. Die in velden met even karakteristiek kreeg bovendien een bijkomende radix 8 implementatie. Vermoedelijk bieden implementaties met hogere radices snelheidsvoordelen. Het afwegen van deze snelheidswinst tegenover de nadelen van mogelijk meer en dus duurdere hardware is iets wat zeker nog verder onderzocht dient te worden. Ook op het vlak van de puntoptelling en puntverdubbeling is een optimalisatie van het aantal benodigde klokslagen mogelijk.

Vooral differentiële vermogenanalyse biedt nog veel ruimte voor vervolgonderzoek. Zo kan nagegaan worden of de voorgestelde alternatieve implementaties voldoende veiligheid bieden. Ook het verschil in veiligheid tussen een radix 1, 4 of 8 implementatie leidt waarschijnlijk tot interessante vaststellingen. Verder schuilen in het zoeken naar betere selectiefuncties voor krachtigere aanvallen nog vele mogelijkheden. De Java- en VHDL-code in Bijlage D vormen een goed vertrekpunt voor deze onderzoeken.

We kunnen dit eindwerk besluiten met de overtuiging dat het laatste woord nog niet gezegd is in de zoektocht naar betere bescherming tegen de verschillende aanvallen. Het eeuwige duel tussen de cryptograaf, die betere beveiliging bedenkt, en de cryptanalyst, die hierop reageert met een krachtigere aanval, is weer enkele ronden verder. Het einde van het duel is nog niet in zicht. Op deze manier blijft het onderzoek in dit domein levendig en vormen de cryptografen en cryptanalisten elkaars stimulans.

Bijlage A

Groepen

Definitie A.0.0.1 *Een* groep *is een verzameling van elementen (G) die, samen met een bewerking (+), moet voldoen aan de vier fundamentele eigenschappen: inwendigheid, associativiteit, eenheidselement en invers element. Dus* $< G, + >$ *is een* groep *als en slechts als*

- *Inwendigheid:*
$$\forall x, y \in G : x + y \in G \tag{A.1}$$
- *Associativiteit:*
$$\forall x, y, z \in G : x + (y + z) = (x + y) + z \tag{A.2}$$
- *Eenheidselement:*
$$\exists e \in G : \forall x \in G : x + e = e + x = x \tag{A.3}$$
- *Invers element:*
$$\forall x \in G : \exists\text{-}x \in G : x + (-x) = (-x) + x = e \tag{A.4}$$

Definitie A.0.0.2 *Een* Abelse groep *is een groep met de extra eigenschap van commutativiteit:*

- *Commutativiteit:*
$$\forall x, y \in G : x + y = y + x \tag{A.5}$$

Voorbeeld A.0.0.1 $< \mathbb{R}, + >$ *en* $< \mathbb{C}, + >$ *zijn Abelse groepen, terwijl de groep van niet-singuliere matrices niet Abels is ten opzichte van de vermenigvuldiging.*

Groepen kunnen onderverdeeld worden in eindige en oneindige groepen, respectievelijk met een eindig en een oneindig aantal elementen.

Voorbeeld A.0.0.2 $< \mathbb{Z}_5, +_5 >$*, met* $+_5$ *de optelling modulo 5, is een eindige groep.* $\mathbb{Z}_5$ *bevat alle natuurlijk getallen modulo 5: 0, 1, 2, 3 en 4.*

Definitie A.0.0.3 *Stel G is een groep en* $x \in G$*. De* orde *van x is de kleinste waarde voor n waarvoor* $\underbrace{x + x + \ldots + x}_{n-1\,keer} = e$ *(het eenheidselement).*

Voorbeeld A.0.0.3 *In* $< \mathbb{Z}_5, +_5 >$ *is de orde van 2 gelijk aan 5 omdat* $(2+2+2+2+2) \bmod 5 = 0$.

Definitie A.0.0.4 *De maximale orde van de elementen van een groep, noemt men de* exponent *van de groep.*

Eigenschap A.0.0.1 *De orde van elk element van de groep deelt de exponent van de groep en de exponent deelt de kardinaliteit van de groep.*

Het bewijs van deze eigenschap bevindt zich in [21].

Definitie A.0.0.5 *Een* cyclische groep *is een groep waarvan alle elementen gegenereerd worden door een element van de groep telkens bij zichzelf op te tellen. Dit element wordt de* generator *genoemd.*

Voorbeeld A.0.0.4 *1 en 5 zijn generatoren voor* $< \mathbb{Z}_6, + >$ *terwijl 0, 2, 3 en 4 dat niet zijn.*

Eén van de moeilijke problemen waarop cryptografie is gebaseerd, is het berekenen van de inverse van een element in een cyclische Abelse groep (zie [21]).

Definitie A.0.0.6 *Twee groepen* $< G_1, + >$ *en* $< G_2, * >$ *zijn* isomorf *als er een bijectie* $f : G_1 \to G_2$ *bestaat die voldoet aan* $f(x+y) = f(x) * f(y)$. *Dit wordt genoteerd met* $G_1 \cong G_2$.

Bijlage B

Velden

Sommige wiskundige toepassingen hebben meer dan één bewerking nodig. Daarom zijn groepen niet altijd voldoende. Velden komen tegemoet aan dit tekort.

Definitie B.0.0.7 *Een* veld *is een verzameling van elementen (F) die, samen met twee bewerkingen (+,*), moeten voldoen aan de* veldaxioma's. $< F, +, * >$ *is een veld, als en slechts als:*

- *Inwendigheid:*

$$\forall x, y \in F : x + y \in F \tag{B.1}$$

$$\forall x, y \in F : x * y \in F \tag{B.2}$$

- *Commutativiteit:*

$$\forall x, y \in F : x + y = y + x \tag{B.3}$$

$$\forall x, y \in F : x * y = y * x \tag{B.4}$$

- *Associativiteit:*

$$\forall x, y, z \in F : x + (y + z) = (x + y) + z \tag{B.5}$$

$$\forall x, y, z \in F : x * (y * z) = (x * y) * z \tag{B.6}$$

- *Distributiviteit:*

$$\forall x, y, z \in F : x * (y + z) = x * y + x * z \tag{B.7}$$

- *Eenheidselement:*

$$\exists 0 \in F : \forall x \in F : x + 0 = 0 + x = x \tag{B.8}$$

$$\exists 1 \in F : \forall x \in F : x * 1 = 1 * x = x \tag{B.9}$$

- *Invers element:*

$$\forall x \in F : \exists \text{-}x \in F : x + (-x) = (-x) + x = 0 \tag{B.10}$$

$$\forall x \in F, x \neq 0 : \exists x^{-1} \in F : x * x^{-1} = x^{-1} * x = 1 \tag{B.11}$$

Eigenschap B.0.0.2 *Stel dat* $< F, +, * >$ *een veld is, dan is* $< F, + >$ *een additieve Abelse groep en* $< F \backslash \{0\}, * >$ *een multiplicatieve Abelse groep.*

Deze eigenschap is een gevolg van de veldaxioma's.

Definitie B.0.0.8 *Een veld met een eindig aantal elementen, noemt men een* eindig veld *of* Galois veld *(GF).*

Eigenschap B.0.0.3 *Elk element van een veld heeft dezelfde orde voor de optelling. Dit is de orde of exponent van de additieve groep.*

Het bewijs van deze eigenschap is te vinden in [5], p 416.

Definitie B.0.0.9 *De* karakteristiek *van een veld is de exponent van zijn additieve groep.*

Eigenschap B.0.0.4 *De karakteristiek is altijd een priemgetal.*

Het bewijs van deze eigenschap bevindt zich in [21].

Eindige velden zijn van groot belang in de cryptografie. Men kan aantonen dat het aantal elementen van een eindig veld steeds gelijk is aan een priemgetal p tot een gehele macht n. Eindige velden met p^n elementen worden voorgesteld door $GF(p^n)$. GF staat voor Galois veld (Eng: 'Galois Field'). Velden kunnen gecategoriseerd worden op basis van de karakteristiek p.

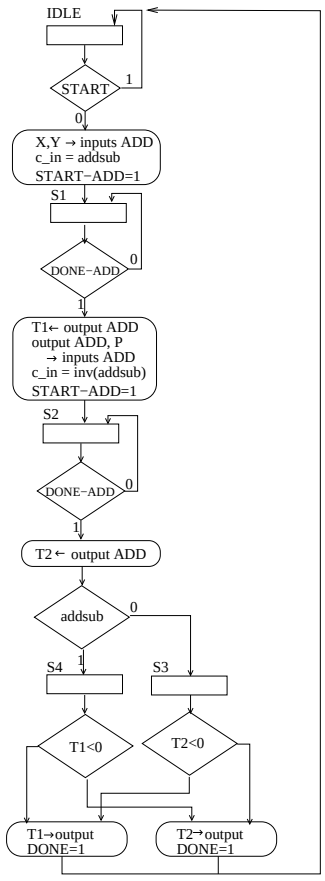
In velden met *oneven karakteristiek* is p een priemgetal verschillend van 2. Velden $GF(p^n)$ van oneven karakteristiek met $n > 1$, zijn zeer moeilijk te implementeren in hardware. Daarom is n voor cryptografische toepassingen bijna altijd gelijk aan 1. $GF(p)$ is het veld van de gehele getallen modulo p. Optelling, aftrekking en vermenigvuldiging van twee elementen komt neer op het uitvoeren van de overeenkomstige traditionele bewerking op de gehele getallen en het reduceren van het resultaat modulo p.

Velden met $p = 2$ hebben een *even karakteristiek*. In dit geval kan n ook verschillend zijn van 1, omdat de elementen van het veld makkelijk kunnen voorgesteld worden door binaire vectoren van dimensie n. Deze eigenschap is zeer nuttig voor hardware implementatie.

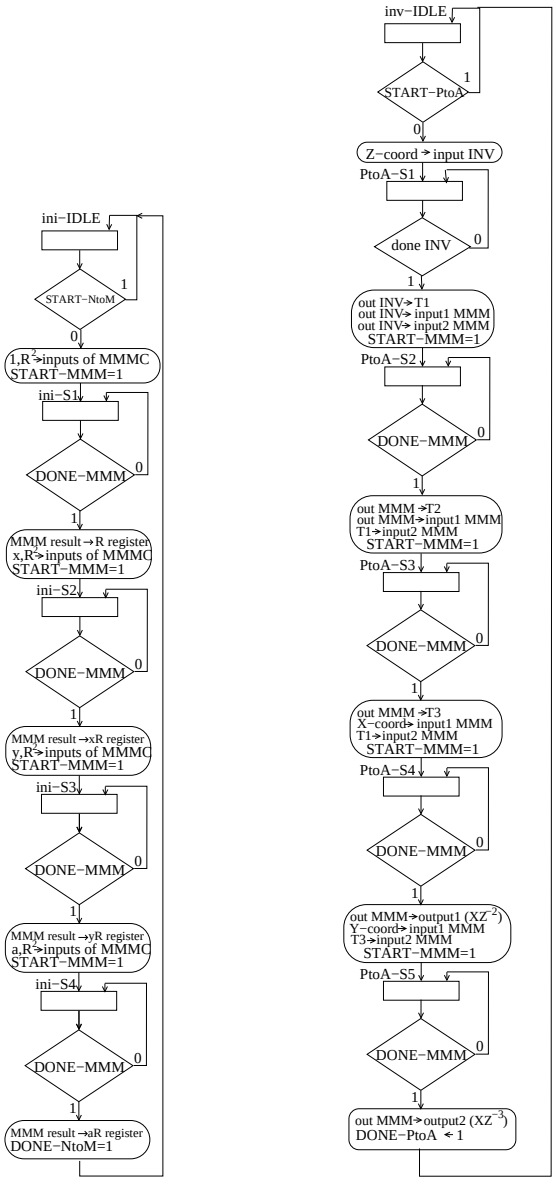
$GF(p^n)$ kan geïnterpreteerd worden als een uitbreiding van het veld $GF(p)$. Men kan ieder veld uitbreiden tot het algebraïsch gesloten is. Een veld in K is algebraïsch gesloten indien iedere veelterm met al zijn coëfficiënten in K een wortel heeft in K (zie ook [2], p. 527).

Bijlage C

FSM



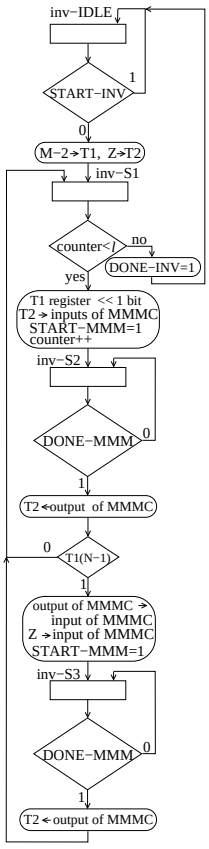
Figuur C.1: State-machine van de modulaire opteller/aftrekker in $GF(p)$



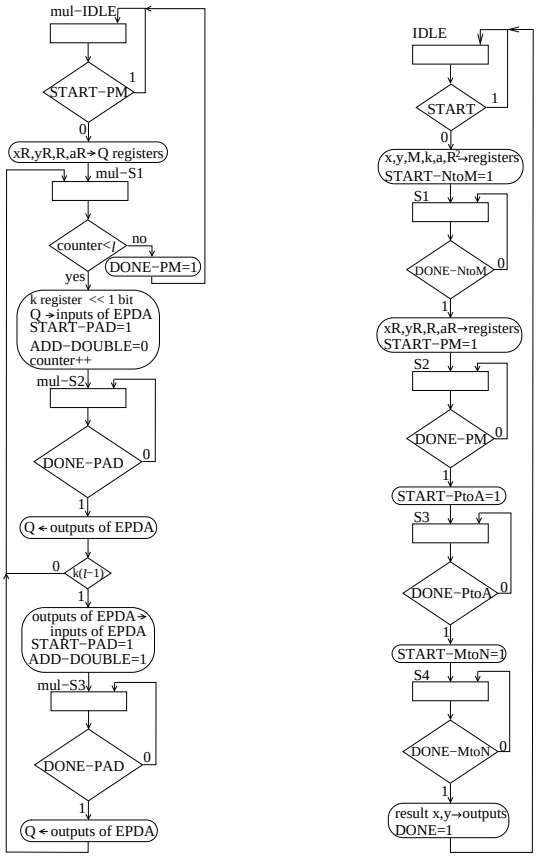
(a) Normaal naar Montgomery

(b) Projectief naar affien

Figuur C.2: State-machines van de puntomzettingen 'normaal naar Montgomery' en 'projectief naar affien' in $GF(p)$



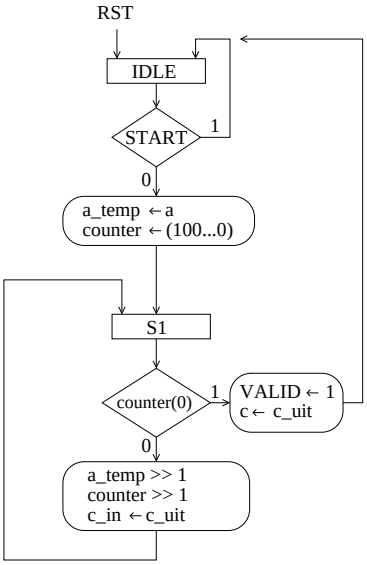
Figuur C.3: State-machine van de inversie in $GF(p)$



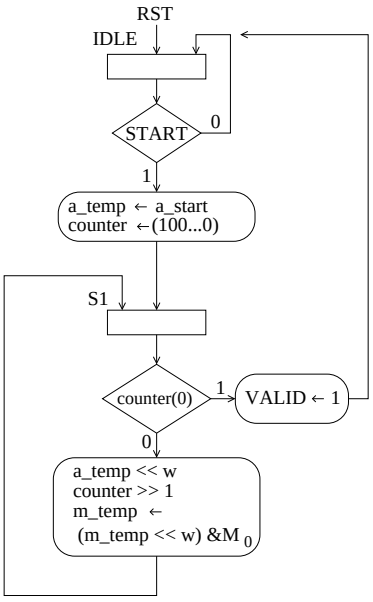
(a) Puntvermenigvuldiging

(b) Hoofd controller

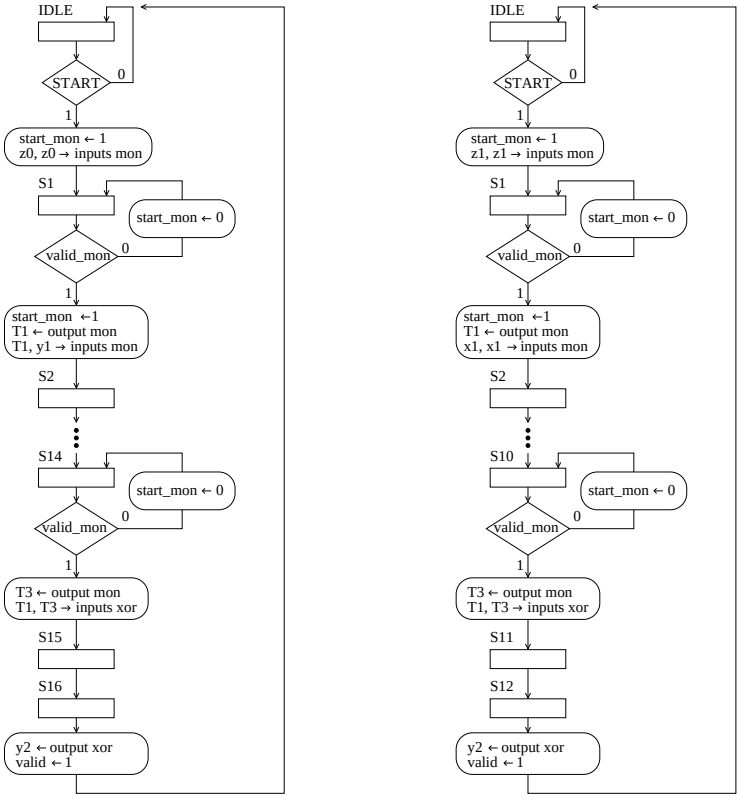
Figuur C.4: State-machines van de puntvermenigvuldiging en hoofd controller in $GF(p)$



Figuur C.5: State-machine van het bit-level algoritme $GF(2^n)$



Figuur C.6: State-machine van het word-level algoritme $GF(2^n)$



(a) Puntoptelling (b) Puntverdubbeling

Figuur C.7: State-machines van de puntoptelling en de puntverdubbeling in $GF(2^n)$

Bijlage D

CD-Rom

De VHDL-, Java- en Matlab-code geschreven in het kader van dit eindwerk bevinden zich in de bijgevoede CD-Rom.

Bibliografie

[1] M. Aigner and E. Oswald. Power analysis tutorial. Technical report, 1998.

[2] M. Artin. *Algebra*. Prentice Hall, 1991.

[3] L. Batina. Power analysis attacks on cryptographic algorithms. Technical report, Stan Ackermans Intstituut, Eindhoven, Nederland, 2001.

[4] E. Biham and A. Shamir. Power analysis of the key scheduling of the AES candidates. In *Proceedings of the second AES Candidate Conference*, pages 115–122, 1999.

[5] G. Birkhoff and S. Mac Lane. *A Survey of Modern Algebra*. Macmilian Publishing Co., 1977.

[6] I.F. Blake, G. Seroussi, and N.P. Smart. *Elliptic Curves in Cryptography*. London Mathematical Society Lecture Note Series. Cambridge University Press Syndicate, 1999.

[7] Ç.K. Koç and T. Acar. Montgomery multiplication in GF(2^k). *Designs, Codes and Cryptography*, 14:57–69, 1998.

[8] S. Chari, C. Jutla, J.R. Rao, and P. Rohatgi. A cautionary note regarding evaluation of AES candidates on smart-cards. In *Proceedings of the second AES Candidate Conference*, pages 133–147, 1999.

[9] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In K. Ohta and D. Pei, editors, *Proceedings of ASIACRYPT 1998*, number 1514 in Lecture Notes in Computer Science, pages 51–65. Springer-Verlag, 1998.

[10] J.S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Ç. K. Koç and C. Paar, editors, *Proceedings of Cryptographic Hardware and Embedded Systems (CHES'99)*, number 1717 in Lecture Notes in Computer Science, pages 292–302. Springer-Verlag, 1999.

[11] J. Daemen and V. Rijmen. Resistance against implementation attacks: A comparative study of the AES proposals. In *Proceedings of the second AES Candidate Conference*, pages 122–132, 1999.

[12] J.F. Dhem, F. Koeune, P.A. Leroux, P. Mestré, J.J. Quisquater, and J.L. Willems. A practical implementation of the timing attack. Technical report, 1998.

[13] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, Nov 1976.

[14] F. Engels en K. Vandekerkhove. *Vermogenanalyse op smartcards.* Eindwerk K.U.Leuven, 2000.

[15] D. Johnson and A. Menezes. The elliptic curve digital signature algorithm (ECDSA). Technical Report CORR 99-34, Department of Combinatorics & Optimization, University of Waterloo, Canada, February 24 2000. `http://www.cacr.math.uwaterloo.ca`.

[16] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. In N. Koblitz, editor, *Advances in Cryptology: Proceedings of CRYPTO'96*, number 1109 in Lecture Notes in Computer Science, pages 104–113. Springer-Verlag, 1996.

[17] P. Kocher, J. Jaffe, and B. Jun. Introduction to differential power analysis and related attacks. `http://www.cryptography.com/dpa/technical`, 1998.

[18] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology: Proceedings of CRYPTO'99*, number 1666 in Lecture Notes in Computer Science, pages 388–397. Springer-Verlag, 1999.

[19] J. López and R. Dahab. Improved algorithms for elliptic curve arithmetic in GF(2^n). Technical report.

[20] M.M. Mano and C.R. Kime. *Logic and Computer Design Fundamentals.* Prentice Hall, Upper Saddle River, New Jersey 07458, second edition, 2001.

[21] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[22] A.J. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.

[23] T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Power analysis attacks of modular exponentiation in smartcards. Number 1717 in Lecture Notes in Computer Science, pages 144–157. Springer-Verlag, August 1999.

[24] School of Mathematics and Statistics of the University of Sydney. Magma computational algebra system. `http://magma.maths.usyd.edu.au/magma/`.

[25] S. B. Örs, E. Oswald, and B. Preneel. Power-analysis attacks on an fpga – first experimental results. In C. Walter, Ç. K. Koç, and C. Paar, editors, *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2003)*, Lecture Notes in Computer Science, 2003.

[26] H. Orup. Simplifying quotient determination in high-radix modular multiplication. In *Proceedings of the 12th Symposium on Computer Arithmetic*, pages 193–199. IEEE, 1995.

[27] B. Parhami. *Computer Aritmetic.* Oxford University Press, 2000.

[28] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[29] L. Batina S.B. Örs, B. Preneel, and J. Vandewalle. Hardware implementation of an elliptic curve processor over GF(p). In *IEEE 14th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, The Hague, The Netherlands, June 24-26 2003.

[30] Xilinx. Xilinx data sheets. `http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp`.