

KATHOLIEKE UNIVERSITEIT LEUVEN  
FACULTEIT TOEGEPASTE WETENSCHAPPEN  
DEPARTEMENT ELEKTROTECHNIEK

KASTEELPARK ARENBERG 10, B-3001 LEUVEN (HEVERLEE), BELGIË  
TEL.: +32 16 321130 - FAX: +32 16 321986



## Compacte implementaties van pairings

Promotoren:  
Prof. dr. ir. Bart Preneel  
Prof. dr. ir. Ingrid Verbauwhede

Dagelijkse begeleiders:  
dr. ir. Lejla Batina  
ir. Miroslav Knezevic

Eindwerk voorgedragen tot het behalen  
van het diploma van Burgerlijk ingeni-  
eur, richting elektrotechniek, optie mul-  
timedia en signaalverwerking

door:

Anthony Van Herrewege

22 mei 2009

Copyright K.U. Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden.

Voor aanvragen tot, of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend U tot de K.U. Leuven, Departement Elektrotechniek - ESAT, Kasteelpark Arenberg 10, BE-3001 Heverlee (België). Telefoon +32 16 32 11 30 & Fax. +32 16 32 19 86

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in dit afstudeerwerk beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Copyright by K.U. Leuven

Without written permission of the promoters and the authors it is forbidden to reproduce or adapt in any form or by any means any part of this publication.

Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to K.U. Leuven, Departement Elektrotechniek - ESAT, Kasteelpark Arenberg 10, BE-3001 Heverlee (Belgium). Tel. +32 16 32 11 30 & Fax. +32 16 32 19 86.

A written permission of the promotor is also required to use the methods, products, schematics and programs described in this work for industrial or commercial use, and for submitting this publication to scientific contests.

# Samenvatting

In deze thesis wordt de berekening van de Tate pairing onder de loep genomen. Een pairing is een wiskundige constructie waarvan in 2001 ontdekt werd dat ze gebruikt kan worden voor het implementeren van identiteitsgebaseerde cryptografie.

Er wordt een compact hardware ontwerp voorgesteld dat de Tate pairing over een supersinguliere curve in  $\mathbb{F}_{2^{163}}$  op een zeer zuinige manier kan berekenen. Het geheugenverbruik van gebruikte algoritmes wordt geminimaliseerd en een ontwerp voor een efficiënt geheugenblok passeert de revue. Ook wordt het effect van verscheidene vermogensoptimalisaties onderzocht.

De uiteindelijke schakeling neemt minimum ongeveer 30k gates in beslag, meer dan drie keer kleiner dan het kleinste ontwerp uit de literatuur. Een verbruik zo laag als 206 nanowatt kan bereikt worden, hoewel in dat geval de rekentijd 51.5 seconden bedraagt. Indien gewenst kan de berekening versneld worden door uitbreiding van de rekenkern. De energie-efficiëntie is, afhankelijk van gekozen parameters, tussen twee en twintig maal beter dan die van de enige andere implementatie uit de literatuur waarvoor dit berekend kon worden.

Ten tijde van dit schrijven zijn nog geen andere compacte, energie-efficiënte ontwerpen gepubliceerd. Het voorgestelde ontwerp is in dat opzicht dus uniek.

# Inhoudsopgave

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Inleiding</b>                               | <b>1</b>  |
| 1.1      | Basisachtergrond cryptografie . . . . .        | 1         |
| 1.2      | Identiteitsgebaseerde cryptografie . . . . .   | 3         |
| 1.3      | Pairings . . . . .                             | 4         |
| 1.3.1    | Microchip implementaties . . . . .             | 5         |
| 1.3.2    | FPGA implementaties . . . . .                  | 5         |
| <b>2</b> | <b>Pairings</b>                                | <b>6</b>  |
| 2.1      | Inleidende wiskunde . . . . .                  | 6         |
| 2.1.1    | Eindige velden . . . . .                       | 6         |
| 2.1.2    | Elliptische krommen . . . . .                  | 7         |
| 2.2      | Definitie van pairings . . . . .               | 8         |
| 2.3      | Berekening van de Tate pairing . . . . .       | 9         |
| 2.4      | Veiligheid van pairings . . . . .              | 10        |
| 2.5      | Toepassingen van pairings . . . . .            | 12        |
| 2.5.1    | Encryptie . . . . .                            | 12        |
| 2.5.2    | Digitale handtekeningen . . . . .              | 14        |
| 2.5.3    | Sleutelafspreek . . . . .                      | 14        |
| <b>3</b> | <b>Implementatie</b>                           | <b>16</b> |
| 3.1      | Parameters . . . . .                           | 16        |
| 3.2      | Beperkingen . . . . .                          | 18        |
| 3.3      | Modulaire Aritmetische Logische Unit . . . . . | 19        |
| 3.4      | $\mathbb{F}_{2^m}$ kern . . . . .              | 21        |
| 3.4.1    | Basisontwerp . . . . .                         | 21        |
| 3.4.2    | Versnelling van de vermenigvuldiging . . . . . | 23        |
| 3.5      | Controller voor het Miller algoritme . . . . . | 25        |
| 3.5.1    | Algemeen ontwerp . . . . .                     | 25        |
| 3.5.2    | For-lus . . . . .                              | 26        |
| 3.5.3    | Finale machtsverheffing . . . . .              | 32        |
| 3.5.4    | Geheugen . . . . .                             | 39        |
| 3.5.5    | FSM . . . . .                                  | 43        |
| 3.6      | Optimalisaties . . . . .                       | 44        |

|          |   |           |
|----------|---|-----------|
| 3.6.1    | Registers zonder reset . . . . .                    | 45        |
| 3.6.2    | Clock gating . . . . .                              | 45        |
| <b>4</b> | <b>Resultaten</b>                                   | <b>48</b> |
| 4.1      | Gebruikte software . . . . .                        | 48        |
| 4.2      | Gerapporteerd vermogen . . . . .                    | 48        |
| 4.3      | Benodigde rektijd . . . . .                         | 49        |
| 4.4      | Basisimplementatie & optimalisaties . . . . .       | 50        |
| 4.5      | Meerdere MALU's . . . . .                           | 52        |
| 4.6      | Hogere kloksnelheid vs. meerdere MALU's . . . . .   | 52        |
| 4.7      | Vergelijking met bestaande implementaties . . . . . | 53        |
| <b>5</b> | <b>Algemeen besluit</b>                             | <b>57</b> |
| 5.1      | Besluit . . . . .                                   | 57        |
| 5.2      | Toekomstig onderzoek . . . . .                      | 58        |
|          | <b>Bibliografie</b>                                 | <b>59</b> |

# Lijst van figuren

|      |   |    |
|------|---|----|
| 1.1  | Algemene werking van symmetrische encryptie . . . . .   | 2  |
| 1.2  | Algemene werking van asymmetrische encryptie . . . . .  | 2  |
| 2.1  | “tangent-and-add” methode op een elliptische kromme . . . .   | 8  |
| 3.1  | MALU - Basis ontwerp met shift . . . . .  | 20 |
| 3.2  | MALU - Geoptimaliseerd ontwerp met shift . . . . .  | 21 |
| 3.3  | Schakeling voor $\mathbb{F}_{2^m}$ kern . . . . .   | 22 |
| 3.4  | Logica voor besturing van de $\mathbb{F}_{2^m}$ kern . . . . .  | 23 |
| 3.5  | Aaneenschakeling van $d$ MALU's ter versnelling van de vermenigvuldiging . . . . .                                    | 24 |
| 3.6  | Schakeling voor de uitvoering van het Miller algoritme . . . .  | 26 |
| 3.7  | Circulair registerblokontwerp en mogelijke bewerkingen . . .  | 40 |
| 3.8  | Circulair registerblok geoptimaliseerd voor laag energieverbruik  | 42 |
| 3.9  | Ontwerp van de schakeling rond de eerste twee registers in het registerblok . . . . .                                 | 43 |
| 3.10 | FSM ontwerp van de controller voor het Miller algoritme . . .   | 44 |
| 3.11 | Schakeling voor clock gating - Basis ontwerp . . . . .  | 46 |
| 3.12 | Schakeling voor clock gating - Verbeterd ontwerp . . . . .  | 46 |
| 3.13 | Vermogenverbruikende onderdelen van een D-type master-slave flip-flop bij constante waarde van de klok ingang . . . . | 46 |
| 3.14 | Schakeling voor clock gating - Laag vermogen ontwerp . . . .  | 47 |
| 4.1  | Aantal klokcycli $c$ nodig voor één pairing i.f.v. het aantal MALU's $d$ voor veldgrootte $m = 163$ . . . . .         | 50 |
| 4.2  | Syntheseresultaten voor implementaties met één MALU en $f = 10$ kHz . . . . .   | 51 |
| 4.3  | Syntheseresultaten voor implementaties met meerdere MALU's en $f = 10$ kHz . . . . .                                  | 54 |

# Lijst van tabellen

|     |  |    |
|-----|--|----|
| 1.1 | Resultaten uit de literatuur voor implementaties ontwikkeld op een MICA node . . . . .   | 5  |
| 1.2 | Resultaten uit de literatuur voor implementaties ontwikkeld voor FPGA's . . . . .  | 5  |
| 3.1 | Oppervlakte van elementen in een ASIC schakeling . . . . .   | 19 |
| 3.2 | Nummer $m \bmod d$ van de MALU waar het resultaat van een vermenigvuldiging zich bevindt voor $d$ MALU's indien $m = 163$      | 25 |
| 4.1 | Aantal klokcycli $c$ nodig voor één pairing i.f.v. het aantal MALU's $d$ voor veldgrootte $m = 163$ . . . . .                  | 50 |
| 4.2 | Syntheseresultaten voor implementaties met één MALU . . .  | 51 |
| 4.3 | Oppervlakte van de individuele onderdelen in de implementatie met één MALU . . . . .   | 52 |
| 4.4 | Syntheseresultaten voor implementaties met $d$ MALU's . . .  | 53 |
| 4.5 | Vergelijking van syntheseresultaten voor twee verschillende implementaties met dezelfde uitvoeringstijd voor één pairing .     | 55 |
| 4.6 | Vergelijking van de implementatie voorgesteld in deze thesis met ASIC implementaties uit de literatuur . . . . .               | 55 |
| 4.7 | Vergelijking van energie-efficiëntie van de implementatie voorgesteld in deze thesis met ASIC implementaties uit de literatuur | 56 |

# Lijst van algoritmes

|      |   |    |
|------|---|----|
| 2.1  | Millers algoritme voor de Tate pairing . . . . .  | 10 |
| 2.2  | Millers algoritme voor de Tate pairing - Efficiëntere versie . . .                                      | 11 |
| 3.3  | Modulo optelling in $\mathbb{F}_{2^m}$ . . . . .  | 20 |
| 3.4  | “Shift and add” vermenigvuldiging in $\mathbb{F}_{2^m}$ . . . . .                                       | 22 |
| 3.5  | Miller algoritme voor de Tate pairing met parameters ingevuld   | 25 |
| 3.6  | Uitwerking van de verdubbelstap voor supersinguliere krommen in het Miller algoritme . . . . .          | 28 |
| 3.7  | Uitwerking van de optelstap voor supersinguliere krommen in het Miller algoritme . . . . .              | 28 |
| 3.8  | Inversie in $\mathbb{F}_{2^{163}}$ . . . . .  | 30 |
| 3.9  | Uitwerking van van $F^2 \in \mathbb{F}_{2^{4m}}$ . . . . .  | 31 |
| 3.10 | Uitwerking van de vermenigvuldiging $F \cdot G$ in het Miller algoritme                                 | 32 |
| 3.11 | Uitwerking van berekening van noemers voor de finale machtsverheffing in het Miller algoritme . . . . . | 35 |
| 3.12 | Uitwerking van $A^{-1} \in \mathbb{F}_{2^{2m}}$ . . . . .   | 35 |
| 3.13 | Uitwerking van $A \cdot B \in \mathbb{F}_{2^{2m}}$ . . . . .  | 36 |
| 3.14 | Uitwerking van $I^{2^m+1} \in \mathbb{F}_{2^{4m}}$ . . . . .  | 38 |
| 3.15 | Uitwerking van $I \cdot J \in \mathbb{F}_{2^{4m}}$ . . . . .  | 39 |



# Lijst van afkortingen

|      |   |
|------|---|
| ASIC | Application-Specific Integrated Circuit                           |
| ECC  | Elliptic Curve Cryptography                                       |
| FPGA | Field-Programmable Gate Array                                     |
| MALU | Modulaire Arithmetische Logische Unit                             |
| MUX  | Multiplexer   |
| RAM  | Random-Access Memory  |
| VHDL | Very high speed integrated circuits Hardware Description Language |

# Lijst van symbolen

## Algemeen

|                          |  |
|--------------------------|--|
| $\mathbb{G}$             | Groep  |
| $\mathbb{F}$             | Galoisveld   |
| $\mathbb{Z}$             | Groep van de gehele getallen   |
| $\mathbb{F}/m\mathbb{F}$ | $\mathbb{F}$ modulo $m$  |
| $m$                      | Graad van het Galoisveld $\mathbb{F}_{2^m}$                                  |
| $E$                      | Elliptische kromme   |
| $\#E$                    | Aantal punten op de elliptische kromme $E$                                   |
| $x_A$                    | x-coördinaat van het punt $A$  |
| $y_A$                    | y-coördinaat van het punt $A$  |
| $e(P, Q)$                | Tate pairing van $P$ en $Q$  |
| $A_i$                    | Coëfficiënt van graad $i$ van veelterm $A$ (tenzij expliciet anders vermeld) |
| $A_{i,j}$                | Coëfficiënten van graad $i$ t.e.m. $j$ van veelterm $A$                      |
| $\text{degree}(A)$       | Graad van de veelterm $A$  |
| $\text{Hamm}(A)$         | Hamming gewicht van binaire representatie van $A$                            |
| $\min(a, b)$             | Minimum uit de lijst van opgegeven argumenten ( $a$ en $b$ )                 |
| $\text{rand}(E)$         | Willekeurig punt op de kromme $E$  |

## Operators

|            |   |
|------------|---|
| $\oplus$   | exclusieve OR   |
| $\#$       | concatenatie  |
| $A \ll a$  | logische verschuiving van $A$ over $a$ posities naar links  |
| $A \gg a$  | logische verschuiving van $A$ over $a$ posities naar rechts |
| $a \mid b$ | $a$ is een deler van $b$                                    |

# Hoofdstuk 1

## Inleiding

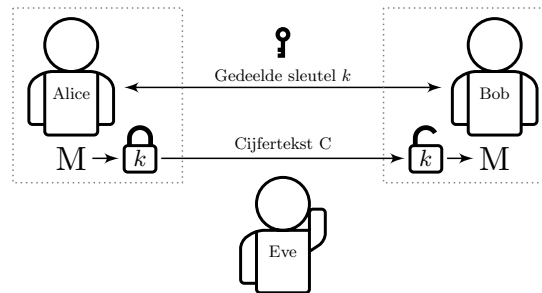
In dit inleidende hoofdstuk zal enige achtergrondinformatie verschaft worden omtrent cryptografie. Verder wordt het concept identiteitsgebaseerde cryptografie duidelijk gemaakt. Er zal uitgelegd worden waarom de recente ontdekking van pairings hier zo belangrijk voor is. Ten slotte geeft een kort overzicht aan wat in de literatuur terug te vinden is qua implementaties van pairings. In het volgende hoofdstuk wordt de werking van pairings dan wiskundig uitgespit.

### 1.1 Basisachtergrond cryptografie

Sinds het begin der tijden is er een nood geweest aan manieren om berichten versleuteld te verzenden tussen twee partijen. Voorbeelden van enkele klassieke encryptiemethoden zijn het Atbashcijfer [1] (Babylonië, 600 v. Chr.), het Caesarcijfer [2] (Rome, 56 n. Chr.) en het dubbele transpositiecijfer [3] (o.a. gebruikt door weerstandsgroepen in WO II). Eén eigenschap die al deze methodes met elkaar gemeen hebben, is het gebruik van dezelfde sleutel voor versleutelen en ontcijferen. Ook vele moderne encryptiemethodes, zoals bijvoorbeeld 3DES [4] en AES [5], gebruiken dit principe, dat men symmetrische versleuteling noemt.

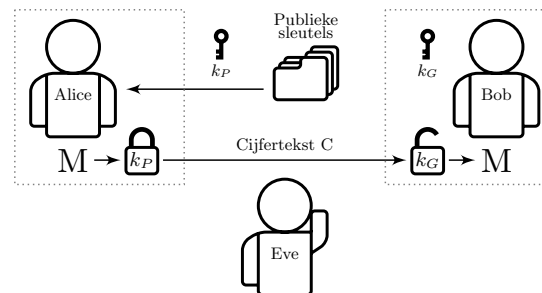
In Figuur 1.1 wordt de algemene werking van een symmetrische encryptie getoond. Alice zendt een bericht  $M$  naar Bob, gecijferd met een door hen beiden gekende sleutel  $k$ . Bob op zijn beurt ontcijfert met diezelfde sleutel de cijfertekst  $C$ . Indien Eve de vooraf afgesproken sleutel kent, kan zij alle communicatie tussen Alice en Bob ontcijferen. Er is dus nood aan een manier om veilig een sleutel  $k$  af te spreken tussen twee partijen.

Een oplossing voor het veilig afspreken van een gedeelde sleutel was tot 1976 niet gekend. Toen stelden Diffie en Hellman hun algoritme voor sleuteluitwisseling over een onbeveiligd kanaal voor [6]. Deze ontdekking plaveide de weg voor asymmetrische cryptografie (ook wel publieke sleutel cryptografie genoemd). Met behulp van dit type cryptografie kunnen eveneens



Figuur 1.1: Algemene werking van symmetrische encryptie

boodschappen versleuteld worden. Dit wordt geïllustreerd in Figuur 1.2. Wanneer Alice een bericht  $M$  naar Bob wil versturen, zoekt ze eerst zijn publieke sleutel  $k_P$  op in een databank. Vervolgens versleutelt ze haar bericht met die publieke sleutel. Enkel Bob kan met behulp van zijn geheime sleutel  $k_G$  dan het bericht ontcijferen. Een systeem als dit biedt het grote voordeel dat er geen nood is om de gebruikte (publieke) sleutel geheim te houden. Het is namelijk onmogelijk om met de publieke sleutel de cijfertekst te ontcijferen. Eve heeft er in dit geval dus geen baat bij de gebruikte publieke sleutel te onderscheppen.



Figuur 1.2: Algemene werking van asymmetrische encryptie

Een andere toepassing van asymmetrische cryptografie is het plaatsen en verifiëren van digitale handtekeningen. Digitale handtekeningen zijn vergelijkbaar met klassieke handtekeningen. Ze kunnen, indien juist geïmplementeerd, gebruikt worden om te verifiëren dat een bepaald bericht effectief door de persoon verstuurd is die de verzender beweert te zijn. Ook kan men aan de hand van een digitale handtekening nagaan of de inhoud van een bericht niet gewijzigd werd door een derde persoon tijdens de verzending. ECDSA [7] en RSA [8] zijn enkele van de vele cryptografische algoritmes die toelaten digitale handtekeningen te genereren.

Om te verzekeren dat de publieke sleutels van elke mogelijke ontvanger voorradig zijn, dient een soort een centrale databank voorzien te worden.

Indien iemand het voor anderen mogelijk wil maken hem versleutelde berichten te versturen, genereert die persoon eerst een publieke en een private sleutel. De publieke sleutel wordt vervolgens naar de database gestuurd, waar iedereen hem kan ophalen.

## 1.2 Identiteitsgebaseerde cryptografie

Een nadeel van de publieke sleutel cryptografie zoals voorgesteld in de vorige paragraaf zit hem in het sleutelbeheer. Er is geen manier om zeker te zijn dat, wanneer de publieke sleutel van Bob opgevraagd wordt, de verkregen sleutel effectief die van Bob is. Indien Eve bijvoorbeeld haar publieke sleutel in de database laat opslaan onder Bobs naam, zal Alice berichten voor Bob versleutelen met Eves publieke sleutel. Een mogelijke oplossing hiervoor is bijvoorbeeld het “web of trust”, zoals geïmplementeerd door de software PGP [9]. Daarbij kunnen mensen aangeven of ze een bepaalde publieke sleutel betrouwbaar vinden of niet. Een sleutel die vergezeld wordt van meerdere getuigenissen van betrouwbaarheid zal dat dan waarschijnlijk ook zijn. Verder bestaat er een zogenaamde “revocation list”, die aangeeft welke sleutels niet meer geldig zijn.

Uiteraard is ook zo een systeem niet volledig waterdicht. Iemand kan bijvoorbeeld onder verschillende identiteiten sleutels insturen en vervolgens, met al die verschillende identiteiten, zijn sleutels een certificaat van vertrouwen geven. Indien iemands publieke sleutel zou afgeleid kunnen worden van bekende gegevens omtrent zijn identiteit, dan zouden deze problemen niet bestaan.

In 1984 stelde Shamir een methode voor waarbij dit mogelijk zou zijn [10]. Het basisidee is als volgt: in plaats van een centrale databank voor publieke sleutels is er een centrale server die private sleutels voor elke gebruiker genereert aan de hand van geheime parameters. Gebruikers kunnen hun private sleutel dus niet zelf berekenen. De centrale server publiceert ook informatie omtrent hoe iemands identiteitsgegevens kunnen worden omgezet naar een publieke sleutel. Wanneer een gebruiker wil deelnemen aan beveiligde communicatie, meldt hij zich aan bij de centrale server en verkrijgt hij zijn private sleutel alsook de parameters om publieke sleutels te berekenen. Voor zowel de private sleutel als de parameters wordt ervan uitgegaan dat deze levenslang gelden. Een gebruiker dient zich dus slechts eenmalig aan te melden.

Uiteraard is ook dit concept niet zonder problemen. Indien bijvoorbeeld de geheime parameters van de centrale server achterhaald worden, is het onmogelijk gebruikers daarvan op de hoogte te brengen. Een mogelijke oplossing is elke gebruiker om de zoveel tijd te voorzien van een nieuwe geheime sleutel. In dat geval stelt zich echter een nieuw probleem, want dan moet een veilige methode bedacht worden om de nieuwe sleutels naar

de gebruikers te transporteren.

Door deze problemen is de toepassing van identiteitsgebaseerde cryptografie eerder geschikt voor kleine groepen mensen, bijvoorbeeld intern in een bedrijf. In dat geval kost het weinig moeite iedereen op regelmatige tijdstippen van nieuwe sleutels te voorzien.

Een andere ideale toepassing is het gebruik van dit type cryptografie in netwerken van sensoren. Zo'n netwerk kan bestaan uit honderden nodes met een beperkte reken- en vermogenscapaciteit. Indien publieke sleutel cryptografie als vanouds zou worden gebruikt, zou dit leiden tot veel extra communicatie tussen de nodes en een centrale server. Telkens de nodes meetgegevens naar de server willen sturen, zouden ze diezelfde server eerst moeten contacteren om zijn publieke sleutel te weten te komen. In het omgekeerde geval zou de server telkens hij een node wil contacteren hetzelfde moeten doen. Als echter identiteitsgebaseerde cryptografie wordt toegepast, is al deze extra communicatie niet meer nodig, aangezien men de benodigde publieke sleutels kan berekenen met behulp van een unieke ID.

Een uitvoerig overzicht van identiteitsgebaseerde cryptografie en de bijhorende mogelijkheden en problemen valt buiten het bestek van thesis. In het volgende hoofdstuk zullen wel enkele mogelijkheden naderbij bestudeerd worden. Een uitstekend startpunt voor meer informatie is Maas [11].

### 1.3 Pairings

Hoewel het idee reeds in 1984 gepubliceerd werd, zou het echter tot 2001 duren eer Boneh en Franklin een efficiënt algoritme voor identiteitsgebaseerde cryptografie voorstelden [12]. Zij stelden een schema op dat toeliet de ideeën van Shamir ook effectief te implementeren. In hun voorstel maakten ze gebruik van de Weil pairing [13]. Al gauw verschenen er variaties op het oorspronkelijke schema. Daarin werd het gebruik van andere pairings voorgesteld, zoals bijvoorbeeld de Tate [14] of de  $\eta_T$  [15] pairing. Wat pairings juist zijn en hoe ze gebruikt kunnen worden, zal in het volgende hoofdstuk uitvoerig aan bod komen.

Alvorens de wiskunde achter pairings in te duiken, wordt eerst nog een overzicht gegeven van de huidige “state of the art” van implementaties van pairings. De mogelijkheden van implementaties op een microchip en een FPGA passeren de revue. Implementaties van pairings op computers zijn per definitie niet in een omgeving met beperkte resources toepasbaar. Ze hebben dus weinig te maken met deze thesis, die als uitgangspunt compacte implementaties heeft. Vandaar dat dit type implementaties dan ook niet bestudeerd zal worden.

### 1.3.1 Microchip implementaties

Implementaties van pairings voor gebruik op een MICA node [16], specifiek ontwikkeld voor gebruik in netwerken van sensoren, worden voorgesteld in [17], [18] en [19]. De processor op deze node is een ATmega128L microchip [20]. Een overzicht van de resultaten is gegeven in Tabel 1.1. Rekening houdend met het stroomverbruik en de batterijspanning gegeven in [19], is het vermogenverbruik waarschijnlijk ongeveer 23.60 *mW*.

Tabel 1.1: Resultaten uit de literatuur voor implementaties ontwikkeld op een MICA node [16]

|               | TinyTate [17]          | TinyPBC [18]           | NanoECC [19]           |                        |
|---------------|------------------------|------------------------|------------------------|------------------------|
|               |                        |                        | Binair                 | Priem                  |
| Veld          | $\mathbb{F}_p$ 256 bit | $\mathbb{F}_{2^{271}}$ | $\mathbb{F}_{2^{163}}$ | $\mathbb{F}_p$ 160 bit |
| Pairing       | Tate                   | $\eta_T$               | Tate                   | Tate                   |
| Rekentijd (s) | 30.21                  | 5.45                   | 10.96                  | 17.93                  |

### 1.3.2 FPGA implementaties

In de literatuur zijn vrij veel ontwerpen voor FPGA's terug te vinden. Het probleem is echter dat men zich bij het ontwerp hiervan steeds toelegt op het behalen van een zo hoog mogelijke snelheid, wat resulteert in een grote oppervlakte. Dit type implementaties is dus minder geschikt voor toepassingen met beperkte resources.

Toch wordt in Tabel 1.2 een summier overzicht gegeven van een zeer beperkt aantal ontwerpen. Bij de selectie hiervan werd vooral gekozen voor ontwerpen waarin in een vrij klein veld gerekend werd. Er dient in acht te worden genomen dat bij al deze implementaties snelheid, en niet een compacte, zuinige implementatie, het voornaamste doel is.

Tabel 1.2: Resultaten uit de literatuur voor implementaties ontwikkeld voor FPGA's

|                            | Veld                   | Pairing  | Opp.<br>[slices] | $f$<br>[MHz] | Reken-<br>tijd [ $\mu s$ ] |
|----------------------------|------------------------|----------|------------------|--------------|----------------------------|
| Ronan <i>et al.</i> [21]   | $\mathbb{F}_{2^{103}}$ | Tate     | 21021            | 51           | 206                        |
| Shu <i>et al.</i> [22]     | $\mathbb{F}_{2^{239}}$ | Tate     | 25287            | 84           | 41                         |
| Keller <i>et al.</i> [23]  | $\mathbb{F}_{2^{251}}$ | Tate     | 27725            | 40           | 2370                       |
| Grabher en Page [24]       | $\mathbb{F}_{3^{97}}$  | Tate     | 4481             | 150          | 432                        |
| Beuchat <i>et al.</i> [25] | $\mathbb{F}_{3^{97}}$  | $\eta_T$ | 1833             | 145          | 192                        |

## Hoofdstuk 2

# Pairings

In dit hoofdstuk zal de werking van pairings wiskundig uit de doeken gedaan worden. Meer specifiek zal de Tate pairing bestudeerd worden. Er zal duidelijk gemaakt worden hoe de pairing berekend kan worden. Vervolgens worden enkele schema's voorgesteld die gebruikt kunnen worden voor versleuteling van gegevens en de aanmaak van digitale handtekeningen. In het volgende hoofdstuk wordt dan een schakeling ontworpen waarmee de Tate pairing berekend kan worden.

Enkel de hoogstnodige theorie zal hier behandeld worden. Voor een meer diepgaande uiteenzetting wordt als startpunt opnieuw verwezen naar Maas [11]. Het is uit dit werk dat de informatie in de volgende paragrafen afkomstig is, tenzij anders vermeld.

### 2.1 Inleidende wiskunde

Alvorens de wiskundige theorie van pairings nader bekeken wordt, dient die van elliptische krommen duidelijk gemaakt te worden. Het is met behulp van deze constructies dat pairings berekend kunnen worden. Elliptische krommen worden doorgaans gedefinieerd over een eindig veld. Vandaar dat de noodzakelijke theorie van beiden hier even heel kort herhaald wordt.

#### 2.1.1 Eindige velden

Een eindig veld  $\mathbb{F}_q$  wordt gedefinieerd door zijn karakteristiek  $q$ . Die karakteristiek is bij cryptografische toepassingen doorgaans een groot priemgetal  $p$  of 2, hoewel tegenwoordig ook onderzoek gedaan wordt naar toepassingen met een karakteristiek 3. Een veld zonder zijn nul element wordt aangeduid als

$$\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}.$$

Volgens de kleine stelling van Fermat geldt in elk eindig veld  $\mathbb{F}_q$  dat  $a^q = a$ . Van deze gelijkheid zal in het volgende hoofdstuk handig gebruikt



gemaakt worden wanneer de inverse van een element berekend moet worden.

Verder kunnen extensies van graad  $m$  van een veld gedefinieerd worden. In het geval  $q = 2$  bekomt men dan een nieuw veld  $\mathbb{F}_{2^m}$ . Er dient dan ook een reductieveelterm  $R$  opgegeven te worden. Een extensieveld wordt gedefinieerd als:

$$\mathbb{F}_{q^m} \cong \mathbb{F}_q[z]/(R).$$

**Voorbeeld** Om de constructie van een extensieveld enigszins te verduidelijken wordt een klein voorbeeld gegeven. Er wordt gewerkt in karakteristiek  $q = 2$ . Stel  $R = z^2 + z + 1$ . Het extensieveld is dus gedefinieerd als:

$$\mathbb{F}_{2^2} \cong \mathbb{F}_2[z]/(z^2 + z + 1).$$

Verder  $A = z$  en  $B = z + 1$ . De resultaten van de optelling en vermenigvuldiging van  $A$  en  $B$  zijn dan respectievelijk:

$$\begin{aligned} A + B &= 2z + 1 & A \cdot B &= z^2 + z \\ &= 1 & &= 2z + 1 \end{aligned}$$

### 2.1.2 Elliptische krommen

Een elliptische kromme  $E$  wordt gevormd door de verzameling van punten die voldoen aan de vergelijking:

$$E : Y^2Z + a_1XYZ + a_2YZ^2 = X^3 + a_3X^2Z + a_4XZ^2 + a_5Z^3.$$

Het enige punt waarvoor  $Z = 0$  en waarvoor de vergelijking geldt ( $X = 0, Y = 1, Z = 0$ ), wordt het punt op oneindig  $\mathcal{O}$  genoemd. Indien wordt gesteld dat  $x = \frac{X}{Z}$  en  $y = \frac{Y}{Z}$ , bekomt men de affiene Weierstrass vergelijking:

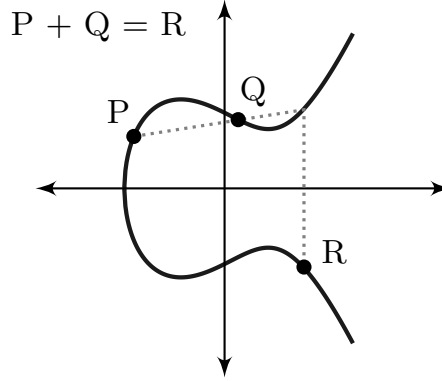
$$E : y^2 + a_1xy + a_2y = x^3 + a_3x^2 + a_4x + a_5.$$

Merk op dat in deze vorm het punt  $\mathcal{O}$  niet meer voldoet aan de vergelijking, ook al behoort het nog steeds tot de kromme. De kromme dient zo gedefinieerd te zijn dat  $\forall A \in E$  de partiële afgeleiden  $\frac{\partial A}{\partial X}$ ,  $\frac{\partial A}{\partial Y}$  en  $\frac{\partial A}{\partial Z}$  nooit allen tegelijkertijd gelijk zijn aan nul.

De natuurlijke bewerking op een kromme is de “tangent-and-chord” methode, die wordt weergegeven in Figuur 2.1. De bewerking wordt additief geschreven en heeft als neutraal element het punt op oneindig  $\mathcal{O}$ . De formules om het resultaat van de “tangent-and-chord” methode te berekenen zijn afhankelijk van het type elliptische kromme.

Aan de hand van de vorige bewerking kan een scalaire vermenigvuldiging vastgelegd worden, met  $a \in \mathbb{Z}$ :

$$\begin{aligned} a \cdot A &= A + \dots + A \\ 0 \cdot A &= \mathcal{O} \\ -a \cdot A &= a \cdot -A \end{aligned}$$



Figuur 2.1: “tangent-and-add” methode op een elliptische kromme

De orde  $n$  van een punt  $A$  op de kromme is gelijk aan de minimum waarde waarvoor  $n \cdot A = \mathcal{O}$ . Het is mogelijk dat  $n = \infty$ . Van alle punten waarvoor  $n$  een deler is van  $l$  wordt gezegd dat ze in de  $l$ -torsiesubgroep van  $E$  zitten. Zo een subgroep wordt genoteerd als:

$$E[l] = \{A \in E : l \cdot A = \mathcal{O}\}.$$

Het aantal punten  $\#E$  op  $E$  wordt de orde van de kromme genoemd. Voor een kromme over een veld  $\mathbb{F}_q$  is  $\#E = q + 1 - t$ , met  $t$  de “trace” van de kromme. Indien  $q \mid t$  wordt de kromme supersingulier genoemd. Voor bepaalde types krommen bestaat er een gesloten formule voor  $\#E$ .

Ten slotte wordt nog de Tate-inbeddingsgraad  $k$  gedefinieerd. Stel  $l \in \mathbb{Z}$  copriem aan  $q$  zodat  $E(\mathbb{F}_q)$  een punt van orde  $l$  bevat. Dan is  $k$  van  $E$  tegenover  $l$  gedefinieerd als het kleinste gehele getal waarvoor  $l \mid q^k - 1$ .

## 2.2 Definitie van pairings

Ten tijde van dit schrijven waren onder meer volgende pairings bekend: Weil, Tate,  $\eta_T$  [15] en Ate [26]. In deze thesis zal gewerkt worden met de Tate pairing, waarvan bewezen is dat ze efficiënter te berekenen is dan de Weil pairing.

De vermelde  $\eta_T$  en Ate pairing zijn variaties op de Tate pairing die gebruikt kunnen worden indien voor specifieke elliptische krommen gekozen wordt. Indien aan de juiste voorwaarden voldaan wordt, zal de schakeling die in het volgende hoofdstuk wordt voorgesteld dus ook gebruikt kunnen worden om deze pairings te berekenen.

De Tate pairing is een functie  $e(P, Q)$  met als argumenten twee punten uit een additieve groep en als resultaat een punt in een multiplicatieve groep:

$$e(P, Q) : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T.$$

Een pairing moet tevens volgende eigenschappen bezitten: goed gedefinieerd, non-degenererend en bilineariteit. Het is de combinatie van deze drie eigenschappen die het zo moeilijk maakt de pairing te definiëren. De betekenis van deze drie eigenschappen is:

1. Goed gedefinieerd:  $e(\mathcal{O}, Q) = 1 \forall Q \in \mathbb{G}_2$ ;  $e(P, \mathcal{O}) = 1 \forall P \in \mathbb{G}_1$ .
2. Non-degenererend:  $\forall P \in \mathbb{G}_1, \exists Q \in \mathbb{G}_2$  waarvoor  $e(P, Q) \neq 1$ .
3. Bilineariteit:  $\forall P_1, P_2, P \in \mathbb{G}_1$  en  $\forall Q_1, Q_2, Q \in \mathbb{G}_2$  geldt:

$$\begin{aligned} e(P_1 + P_2, Q) &\equiv e(P_1, Q) \cdot e(P_2, Q) & \text{en} \\ e(P, Q_1 + Q_2) &\equiv e(P, Q_1) \cdot e(P, Q_2). \end{aligned}$$

De Tate pairing over elliptische krommen wordt als volgt geschreven:

$$e(P, Q) : E(\mathbb{F}_{q^k})[l] \times E(\mathbb{F}_{q^k})/lE(\mathbb{F}_{q^k}) \mapsto \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^l.$$

Het resultaat van de Tate pairing is niet uniek, maar een element van een equivalentieklasse  $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^l$ . Voor twee resultaten  $a, b \in \mathbb{F}_{q^k}^*$  geldt  $a \equiv b$  als en slechts als er een element  $c \in \mathbb{F}_{q^k}^*$  bestaat waarvoor  $a = bc^l$ . Aangezien voor cryptografische toepassingen een uniek resultaat gewenst zal zijn, dient die  $n$ -de macht weggewerkt te worden. Dit kan door het resultaat te verheffen tot de macht  $\frac{q^k-1}{l}$ . Aangezien  $a^{q^k-1} = 1$  voor elke  $a \in \mathbb{F}_{q^k}^*$  (kleine stelling van Fermat), verkrijgt men aldus een  $l$ -eenheidswortel. Soms spreekt men over de gereduceerde Tate pairing wanneer men expliciet wil aangeven dat deze machtsverheffing wordt uitgevoerd.

Het formeel bewijzen van de Tate pairing valt buiten het bestek van deze thesis. Voor meer informatie zie onder andere [11, 27, 28].

## 2.3 Berekening van de Tate pairing

Reeds in 1986 stelde Miller een methode voor om de Tate pairing te berekenen [29, 30]. Algoritme 2.1 geeft weer hoe dat in zijn werk gaat.

Daarbij staat  $G_{A,B}(C)$  voor de evaluatie van het punt  $C$  in de vergelijking van de rechte door de punten  $A$  en  $B$ . Indien  $G_{2,A}$  vermeld staat, wil dit de vergelijking door de raaklijn aan  $2 \cdot A$  zeggen.

In [30] wordt aangetoond dat indien gewerkt wordt in het veld  $\mathbb{F}_{2^m}$  het algoritme sterk vereenvoudigd kan worden. Er moet dan gebruik gemaakt worden van de supersinguliere kromme

$$E(\mathbb{F}_{2^m}) : y^3 + y = x^3 + x + b$$

met  $b \in \{0, 1\}$ . Verder dient ook een distortie map  $\phi$  toegepast te worden op het punt  $Q$ . Voor een veld  $\mathbb{F}_{2^m}$  dient de mapping te voldoen aan volgende voorwaarden:

$$\phi : (x_Q, y_Q) \mapsto (x_Q + s^2, y_Q + x_Q s + t^6)$$

**Algoritme 2.1:** Millers algoritme voor de Tate pairing

---

**Input:**  $l \in \mathbb{Z}; P \in E(\mathbb{F}_{q^k})[l]; Q \in E(\mathbb{F}_{q^k})/lE(\mathbb{F}_{q^k})$   
**Output:**  $e(P, Q) \in \mathbb{F}_{q^k}^*$   
**Data:**  $i, t \in \mathbb{Z}; Q' \in E(\mathbb{F}_{q^k}); V \in E(\mathbb{F}_{q^k})[l]; F \in \mathbb{F}_{q^k}$

```

1  $t \leftarrow \lfloor \log_2(l) \rfloor$ 
2  $Q' \leftarrow \text{rand}(E)$ 
3  $S \leftarrow Q + Q'$ 
4  $F \leftarrow 1; V \leftarrow P$ 
5 for  $i = t - 1$  tot 0 do
6    $F \leftarrow F^2 \cdot \frac{G_{V,V}(S) \cdot G_{2 \cdot V}(Q')}{G_{2 \cdot V}(S) \cdot G_{V,V}(Q')}$ 
7    $V \leftarrow 2 \cdot V$ 
8   if  $l_i = 1$  then
9      $F \leftarrow F \cdot \frac{G_{V,P}(S) \cdot G_{V+P}(Q')}{G_{V+P}(S) \cdot G_{V,P}(Q')}$ 
10     $V \leftarrow V + P$ 
11 return  $F^{\frac{q^k-1}{l}}$ 

```

---

met  $s, t \in \mathbb{F}_{2^{km}}$ . Daarbij moeten  $s$  en  $t$  voldoen aan volgende vergelijkingen:

$$\begin{cases} s^4 + s = 0 \\ t^2 + t + s^6 + s^2 = 0 \end{cases}$$

Een bijkomend voordeel van het gebruik van de vermelde kromme is haar lage inbeddingsgraad  $k = 4$ . Hoe groter  $k$ , hoe meer bewerkingen het immers kost om de pairing te berekenen. Tevens zorgt de distortie map er voor dat  $e(P, P) \neq 1$ , wat wordt omgeschreven als sterk niet-degenererend.

Ten slotte dienen lichtelijk andere argument opgegeven worden, deze aangepaste Tate pairing is namelijk gedefinieerd als:

$$\hat{e}(P, Q) : E(\mathbb{F}_q)[l] \times E(\mathbb{F}_q)[l] \mapsto \mathbb{F}_{q^k}^* / (\mathbb{F}_{q^k}^*)^l.$$

De resulterende efficiëntere versie van Millers algoritme is gegeven in Algoritme 2.2.

## 2.4 Veiligheid van pairings

Nu de wiskundige achtergrond van de Tate pairing enigszins verduidelijkt is, zal getoond worden waarom pairings veilig genoeg zijn om gebruikt te kunnen worden in cryptografische toepassingen.

Eerst dienen echter de begrippen een moeilijk en een gemakkelijk probleem gedefinieerd te worden. Een moeilijk probleem is een probleem waarvoor geen algoritme bekend is dat een oplossing kan vinden in polynomiale

---

**Algoritme 2.2:** Millers algoritme voor de Tate pairing - Efficiëntere versie

---

**Input:**  $l \in \mathbb{Z}; P, Q \in E(\mathbb{F}_{2^m})[l]$

**Output:**  $\hat{e}(P, Q) \in \mathbb{F}_{2^{km}}^*$

**Data:**  $i, t \in \mathbb{Z}; F, V \in \mathbb{F}_{2^{km}}$

```

1  $t \leftarrow \lfloor \log_2(l) \rfloor$ 
2  $F \leftarrow 1; V \leftarrow P$ 
3 for  $i = t - 1$  tot 0 do
4    $F \leftarrow F^2 \cdot G_{V,V}(\phi(Q))$ 
5    $V \leftarrow 2 \cdot V$ 
6   if  $l_i = 1$  en  $i \neq 0$  then
7      $F \leftarrow F \cdot G_{V,P}(\phi(Q))$ 
8      $V \leftarrow V + P$ 
9 return  $F^{\frac{2^{km}-1}{l}}$ 
```

---

tijd. Voor een gemakkelijk probleem bestaan er wel algoritmen die in polynomiale tijd een oplossing leveren.

De veiligheid van pairings vindt zijn oorsprong in het discrete logaritme (DL) probleem. Dit wordt gedefinieerd als volgt. Definieer de multiplicatieve groep  $\mathbb{G}$  van orde  $n$ , met  $n$  priem en  $g$  een generator voor  $\mathbb{G}$ . Stel nu  $h \in \mathbb{G}$  en dat  $n$  priem is. In theorie is het niet nodig dat  $n$  priem is. Indien dit niet het geval is, kan het probleem echter gereduceerd worden tot een probleem met  $n$  priem via de Chinese reststelling. Stel nu  $x \in \mathbb{Z}_n^*$ , zodat  $h = g^x$ . Het discrete logaritme probleem bestaat er dan in  $x$  te vinden, gegeven  $g$  en  $h$ . Dit wordt geschreven als  $\text{DL}(h) = x$ .

Het DL probleem wordt algemeen aanzien als moeilijk. De snelste algoritmes die dit probleem kunnen oplossen voor een willekeurige groep  $\mathbb{G}$  (d.w.z. die geen specifieke eigenschappen van het veld uitbuiten) hebben een complexiteit  $O(\sqrt{n})$ , met  $n$  de orde van de groep. Aangezien  $n$  exponentieel groeit met de grootte van de groep, geldt hetzelfde voor de uitvoeringstijd van de algoritmes. Enkele voorbeelden van zulke algoritmes zijn Baby-step giant-step [31] en Pollards rho algoritme [32]. Voor bepaalde groepen zijn algoritmes bekend met subexponentiële tijdscomplexiteit. In het geval de groep gedefinieerd is als de punten op een supersinguliere elliptische krommen, kan men bijvoorbeeld MOV- [33] of FR-reductie [27] gebruiken. In dat geval is het noodzakelijk de grootte van de groep waarover de kromme gedefinieerd is groot genoeg te nemen, zodat het computationeel onhaalbaar blijft het DL probleem op te lossen.

Een probleem verwant aan het DL probleem is het computationeel Diffie-Hellman (CDH) probleem. Stel  $a, b \in \mathbb{Z}_n^*$ . In dit geval worden  $g, g^a, g^b \in \mathbb{G}$  gegeven. Het probleem is nu een element  $h \in \mathbb{G}$  te vinden waarvoor  $h = g^{ab}$ .

Het probleem wordt uitgedrukt als  $\text{CDH}(g^a, g^b) = h$ .

Aangezien het CDH probleem kan opgelost worden indien het DL probleem kan opgelost worden, is het dus zeker niet moeilijker. Indien het CDH probleem moeilijk is, is het DL probleem dat dus ook. Of het omgekeerde geldt, is niet geweten. Tot nu toe zijn echter geen methodes gekend die het CDH probleem sneller oplossen dan de methodes die het DL probleem oplossen. De twee worden dan ook als equivalent beschouwd.

Ten derde is er dan het beslissings Diffie-Hellman (DDH) probleem. Stel  $a, b, c \in \mathbb{Z}_n^*$ . Nu dient, gegeven  $g, g^a, g^b, g^c \in \mathbb{G}$ , beslist te worden of  $g^c = g^{ab} \pmod{n}$ . Dit wordt uitgedrukt als  $\text{DDH}(g^a, g^b, g^c) = 1$  indien waar en  $\text{DDH}(g^a, g^b, g^c) = 0$  in het andere geval.

Het DDH probleem kan opgelost worden aan de hand van het CDH probleem door  $h = \text{CDH}(g^a, g^b)$  te berekenen en na te gaan of  $h = g^c$ . Het is dus duidelijk dat het DDH probleem zeker niet moeilijker is dan het CDH probleem. Er zijn echter bepaalde groepen bekend waarvan geweten is dat het DDH probleem makkelijk is en het CDH probleem moeilijk. Deze groepen worden “gap Diffie-Hellman” groepen genoemd. Pairings genereren zo een groep, het CDH probleem over elliptische krommen wordt immers als moeilijk beschouwd en ze laten toe  $\text{DDH}(aP, bP, cP)$  makkelijk te beantwoorden:

$$\begin{aligned} e(aP, bP) &= e(P, bP)^a \\ &= e(P, abP) \\ &= e(P, cP) \end{aligned}$$

als en slechts als  $ab = c$ .

## 2.5 Toepassingen van pairings

Hoewel de theorie van pairings enigszins is uitgelegd, blijft nog steeds de vraag hoe van hun eigenschappen gebruik gemaakt kan worden in cryptografische algoritmen. In deze paragraaf zullen enkele mogelijkheden voorgesteld worden. Opnieuw wordt er op gewezen dat dit zeker geen volledige lijst van mogelijkheden is en meer informatie te vinden is in [11].

### 2.5.1 Encryptie

Verschillende encryptieschema's met pairings zijn bekend. Hier wordt BasicIdent voorgesteld, het eerste bekende schema voor identiteits-gebaseerde encryptie, in 2001 gepubliceerd door Boneh en Franklin [12]. Het schema bestaat uit vier algoritmen. Het eerste, ‘setup’, initialiseert de systeemparameters, waaronder de geheime hoofdsleutel voor de centrale server. Ten tweede is er het ‘extract’ algoritme dat gebruikt wordt om gebruikers te voorzien van een geheime sleutel. De laatste twee, ‘encrypt’ en ‘decrypt’,

laten toe, zoals de naam het zegt, berichten te versleutelen en ontcijferen. De vier algoritmen worden hieronder verder uitgelegd.

**Setup** Eerst wordt een  $k$ -bit lang priemgetal  $q$  gekozen. Er wordt een additieve en een multiplicatieve groep van orde  $q$  gedefinieerd, respectievelijk  $\mathbb{G}_1$  en  $\mathbb{G}_2$ . Ten slotte is er nog een generator  $P$  van  $\mathbb{G}_1$  nodig.

Vervolgens wordt de geheime hoofdsleutel  $s \in \mathbb{Z}_q^*$  van de server willekeurig gekozen en stelt men  $P_{\text{pub}} = sP$ . Tevens is er nood aan twee cryptografische hashfuncties:  $H_1 : \{0, 1\}^* \mapsto \mathbb{G}_1^*$  en  $H_2 : \mathbb{G}_2 \mapsto \{0, 1\}^n$ . De waarde van  $n$  kan willekeurig gekozen worden en zal de lengte van het te versleutelen bericht bepalen. Deze waarde heeft dus geen invloed op de veiligheid van het systeem.

Een bericht  $M = \{0, 1\}^n$  en de sleuteltekst  $C \in \mathbb{G}_1^* \times \{0, 1\}^n$ .

De publieke systeemparemeters zijn:  $q, \mathbb{G}_1, \mathbb{G}_2, n, P, P_{\text{pub}}, H_1$  en  $H_2$ . Elke gebruiker heeft toegang tot deze parameters.

**Extract** De server berekent een gebruiker zijn geheime sleutel als:

$$d_{\text{ID}} = sQ_{\text{ID}},$$

met  $Q_{\text{ID}} = H_1(\text{ID})$ . ID is een bitstring die uniek de identiteit van de gebruiker bepaalt en  $s$  de geheime hoofdsleutel. Enkel de server kan dus geheime sleutels berekenen.

**Encrypt** Kies een willekeurige  $r \in \mathbb{Z}_q^*$  en stel  $Q_{\text{ID}} = H_1(\text{ID})$ . De sleuteltekst  $C$  van bericht  $M$  is dan:

$$C = \langle rP, M \oplus H_2(g_{\text{ID}}^r) \rangle,$$

met  $g_{\text{ID}}^r = e(Q_{\text{ID}}, P_{\text{pub}})$ .

**Decrypt** Een sleuteltekst  $C = \langle U, V \rangle$  wordt ontcijfert als volgt:

$$M = V \oplus H_2(e(d_{\text{ID}}, U)).$$

Dat het resultaat wel degelijk klopt is als volgt te bewijzen:

$$\begin{aligned} e(d_{\text{ID}}, U) &= e(sQ_{\text{ID}}, rP) \\ &= e(Q_{\text{ID}}, P)^{sr} \\ &= e(Q_{\text{ID}}, P_{\text{pub}})^r \\ &= g_{\text{ID}}^r. \end{aligned}$$

Het bericht  $M$  ondergaat dus twee maal een exclusieve OR met  $g_{\text{ID}}^r$ .

Het is mogelijk uitbreidingen aan dit systeem toe te voegen. Zo kan bijvoorbeeld bij de berekening van iemands publieke sleutel een datum bij de

ID gevoegd worden. De ontvanger dient de centrale server dan om een specifieke sleutel te vragen om het ontvangen berichten te ontcijferen. Afhankelijk van de datum van aanvraag kan de server deze aanvraag dan inwilligen of weigeren. Merk op dat in dit geval de ontvanger wel steeds de server moet contacteren alvorens ontcijfering mogelijk is.

### 2.5.2 Digitale handtekeningen

Net zoals er verschillende schema's bestaan voor identiteits-gebaseerde encryptie, bestaan er ook verscheidene voor digitale handtekeningen. Hier wordt dat van Cha en Cheon [34] gepresenteerd. Opnieuw bestaat het schema uit vier algoritmes, 'setup', 'extract', 'sign' en 'verify'. Daarbij zijn de 'setup' en 'extract' algoritmes identiek aan die van het BasicIdent schema, met als extra publieke parameter een hashfunctie  $H_3 : \{0, 1\}^* \times \mathbb{G}_1 \mapsto \mathbb{Z}_q^*$ . De 'sign' en 'verify' algoritmes werken als volgt:

**Sign** Kies een willekeurige  $r \in \mathbb{Z}_q^*$  en bereken  $U = rQ_{\text{ID}}$ . Stel vervolgens  $h = H_3(M, U)$  en  $V = (r + h) \cdot d_{\text{ID}}$ . De handtekening is dan  $\sigma = \langle U, V \rangle$ .

**Verify** Een handtekening bij een bericht  $M$  is geldig indien

$$e(P, V) = e(P_{\text{pub}}, U + hQ_{\text{ID}}),$$

met  $h$  zoals gedefinieerd in het 'sign' algoritme. Dat dit klopt kan geverifieerd worden als volgt:

$$\begin{aligned} e(P, V) &= e(P, (r + h) \cdot d_{\text{ID}}) \\ &= e(P, (r + h) \cdot sQ_{\text{ID}}) \\ &= e(P, (r + h) \cdot Q_{\text{ID}})^s \\ &= e(P_{\text{pub}}, U + hQ_{\text{ID}}) \end{aligned}$$

### 2.5.3 Sleutelafpraak

Naast sleutelafpraak zoals die bestaat in publieke sleutelcryptografie laten pairings enkele opmerkelijke schema's toe om sleutels af te spreken. Twee daarvan worden hier gepresenteerd.

#### Niet-interactieve sleutelafpraak

Een niet-interactieve sleutelafpraak is een sleutelafpraak waarbij de betrokken partijen geen contact met elkaar dienen op te nemen om de sleutel af te spreken. Dit heeft als voordeel dat een man-in-the-middle aanval onmogelijk is. Volgende techniek werd voorgesteld in [35]. Twee partijen A en



B spreken een gezamenlijke sleutel  $k$  af door middel van volgende formules:

$$\begin{aligned} k &= e(d_A, Q_B) \\ &= e(sQ_A, Q_B) \\ &= e(Q_A, d_B). \end{aligned}$$

Aangezien beiden hun persoonlijke geheime sleutel bezitten en de publieke sleutel van de andere persoon kunnen berekenen, is er dus geen communicatie noodzakelijk. Een nadeel van dit schema is dat de afgesproken sleutel steeds dezelfde zal zijn. In [36] wordt een schema voorgesteld om de sleutel dynamisch te maken, daarbij is echter wel communicatie vereist.

### Driepartijdige sleutelafpraak

Met behulp van pairings is het mogelijk tussen drie partijen een sleutel af te spreken in slechts één communicatieronde [37], daar waar er normaal minstens twee rondes nodig zijn. Dit gaat als volgt in zijn werk: de drie partijen publiceren respectievelijk  $aP$ ,  $bP$  en  $cP$ , met willekeurige  $a, b, c \in \mathbb{Z}_q^*$ . De gedeelde geheime sleutel  $k$  is dan:

$$\begin{aligned} k &= e(bP, cP)^a \\ &= e(aP, cP)^b \\ &= e(aP, bP)^c \\ &= e(P, P)^{abc} \end{aligned}$$

Merk op dat dit schema geen gebruik maakt van identiteiten en dus niet tot identiteits-gebaseerde cryptografie behoort. Tevens is het niet bestand tegen man-in-the-middle aanvallen. In [38] wordt met behulp van publieke sleutel cryptografie een resistent schema uitgewerkt.

## Hoofdstuk 3

# Implementatie

In dit hoofdstuk wordt de implementatie van een schakeling voor de berekening van de Tate pairing uit de doeken gedaan. Het uiteindelijke doel is een zo compact mogelijke ASIC implementatie te verkrijgen.

Er zal onderzocht worden welke basisbewerkingen nodig zijn en hoe deze verwezenlijkt kunnen worden in hardware. Vervolgens wordt een schakeling ontworpen die aan de hand hiervan alle nodige berekeningen kan uitvoeren in het gekozen veld. Ten slotte is er dan nog de schakeling die alle berekeningen voor het Miller algoritme (zie Paragraaf 2.3) in goede banen leidt. In het volgende hoofdstuk zal de resulterende schakeling dan beoordeeld worden.

Allereerst worden echter noodzakelijke parameters vastgelegd, zodat alle bewerkingen exact gedefiniëerd kunnen worden. Vervolgens wordt bekeken welke beperkingen aan de schakeling opgelegd moeten worden.

### 3.1 Parameters

Alvorens kan begonnen worden met het ontwerpen van de implementatie, moeten alle bewerkingen in het Miller algoritme volledig gedefiniëerd zijn. Daartoe dienen bepaalde parameters vastgelegd worden.

Een belangrijke parameter is het veld waarover gewerkt zal worden. Omdat de berekeningen in een veld  $\mathbb{F}_2$  (en extensies ervan) veel eenvoudiger zijn dan die in een veld modulo een priemgetal, wordt ervoor gekozen in zulk een veld te werken.

Ook moet een kromme vastgelegd worden. Er wordt gekozen voor de reeds eerder vermelde supersinguliere elliptische kromme

$$E(\mathbb{F}_{2^m}) : y^3 + y = x^3 + x + b,$$

waarbij  $b \in \{0, 1\}$ . Dat type krommen heeft een kleine inbeddingsgraad  $k = 4$  en geeft aanleiding tot simpele rekenregels voor de verdubbeling en optelling van punten [30, 39]. Verder worden de volgende hulp variabelen

bepaald:

$$\delta = \begin{cases} b & m \equiv 1, 7 \pmod{8} \\ 1 - b & m \equiv 3, 5 \pmod{8} \end{cases}$$

$$\nu = (-1)^\delta$$

De orde van de kromme is gelijk aan [30]:

$$\#E(\mathbb{F}_{2^m}) = 2^m + \nu\sqrt{2^{m+1}} + 1$$

Om de Tate pairing efficiënt te kunnen berekenen, moet de variabele  $b$  zo gekozen worden dat  $\#E$  enkel uit optellingen bestaat. Met andere woorden:  $b$  moet zo gekozen worden dat  $\nu = 1$ .

Alvorens dit kan gebeuren, moet de grootte  $m$  van het veld  $\mathbb{F}_{2^m}$  vastgelegd worden. Dit bepaalt de cryptografische sterkte alsook de grootte van het eindresultaat. Des te groter  $m$ , des te groter uiteraard de benodigde registers om alle data in op te slaan. Uitgaande van de data in [40] wordt gekozen voor  $m = 163$ . Dit geeft een cryptografische sterkte vergelijkbaar met 86 bits symmetrische cryptografie. Dit zou neerkomen op een gegarandeerde veiligheid tot 2021. De algoritmen en schakelingen die volgen kunnen allen perfect toegepast worden op grotere velden, dus indien gewenst kan  $m$  verhoogd worden. Dit zal echter navenante effecten hebben op de grootte en het vermogenverbruik van de uiteindelijke implementatie.

Nu  $m$  vastgelegd is, kan hetzelfde gedaan worden voor  $b$ . Om een aftrekking in de formule voor  $\#E$  te voorkomen, dient  $b = 1$  te zijn in dit geval. Meteen kan dan ook een  $l$ -torsiesubgroep van  $E$  vastgelegd worden. Daarbij moet er op gelet worden dat  $l \mid \#E$ . De eenvoudigste keuze is:

$$\begin{aligned} l = \#E &= 2^{163} + \sqrt{2^{163+1}} + 1 \\ &= 2^{163} + 2^{82} + 1. \end{aligned}$$

Nu het veld waarover gewerkt wordt, bepaald is, kan een reductieveelterm  $R$  gekozen worden. Bij de keuze daarvan werd uitgegaan van de aangegeven parameters in [41]. De veelterm is:

$$R = z^{163} + z^7 + z^6 + z^3 + 1.$$

Ook moet het extensieveld  $\mathbb{F}_{2^{km}}$  gedefinieerd worden waarover de Tate pairing berekend zal worden. Aangezien  $k = 4$ , kan het veld opgebouwd worden in twee stappen. Eerst wordt  $\mathbb{F}_{2^{2m}}$  bepaald als volgt:

$$\mathbb{F}_{2^{2m}} \cong \mathbb{F}_{2^m}[x]/(x^2 + x + 1)$$

en hierop wordt het volgende extensieveld gebouwd:

$$\mathbb{F}_{2^{4m}} \cong \mathbb{F}_{2^{2m}}[y]/(y^2 + (x + 1)y + 1).$$

Een element van  $\mathbb{F}_{2^{4m}}$  kan dus voorgesteld worden als vier elementen van  $\mathbb{F}_{2^m}$ .

Ten slotte moet een distortiemap  $\phi$  bepaald worden. Zoals reeds vermeld in Hoofdstuk 2 dient die volgens [30] als volgt te zijn. Een punt  $A \in \mathbb{F}_{2^m}$  ondergaat onder de distortie de transformatie:

$$\phi : (x_Q, y_Q) \mapsto (x_Q + s^2, y_Q + x_Q s + t^6)$$

met  $s, t \in \mathbb{F}_{2^{4m}}$ . Daarbij moeten  $s$  en  $t$  voldoen aan volgende vergelijkingen:

$$\begin{cases} s^4 + s = 0 \\ t^2 + t + s^6 + s^2 = 0 \end{cases}$$

Een mogelijke oplossing hiervoor is:

$$\begin{aligned} s &= x + 1, \\ t &= xy \end{aligned}$$

Een distortie van  $A$  kan dus ook geschreven worden als:

$$\begin{aligned} x_\phi &= x_A + x, \\ y_\phi &= (x_A + y_A) + x_A \cdot x + xy. \end{aligned}$$

Merk op dat de constante term van  $x_\phi$  en de coëfficiënt van  $x$  van  $y_\phi$  beiden gelijk zijn aan  $x_A$ . Door deze handige vorm zullen in de implementatie slechts twee registers nodig zijn om de distortie van het punt  $Q$  bij te houden.

## 3.2 Beperkingen

Het doel is de uiteindelijke schakeling zo klein mogelijk te maken, zodat ze gebruikt kan worden in bv. netwerken van sensoren of een smartcard. Beperking van de oppervlakte is dus de belangrijkste factor. Een tweede belangrijke factor is vermogenverbruik, om dit te beperken bestaan specifieke technieken. Het verbruik hangt ook samen met de oppervlakte, dus het beperken daarvan zal het verbruik ten goede komen. In eerste instantie zal dus getracht worden de implementatie zo compact mogelijk te maken. Het verbruik kan verder ook verlaagd worden door een lagere kloksnelheid voor de schakeling te gebruiken, wat uiteraard de rekensnelheid niet bevordert. De rekensnelheid is echter geen prioriteit en dus zal dit aspect bij het ontwerp van de schakelingen genegeerd worden.

Algemeen kan dus gesteld worden dat hoe kleiner het uiteindelijke resultaat is, hoe beter. Het is dus cruciaal de elementen te identificeren die het meeste plaats innemen in een ASIC schakeling. In Tabel 3.1 is de grootte van de belangrijkste elementen te vinden. Deze cijfers gelden enkel bij gebruik van  $0,13 \mu\text{m}$  low leakage technologie van Faraday Technology Corporation. De ordening van de elementen zal echter grotendeels behouden worden voor andere technologieën. Uit de tabel blijkt dat het gebruik van flip-flops (registers), adders en multiplexers zoveel mogelijk beperkt moet worden.

Tabel 3.1: Grootte van elementen in een ASIC schakeling (0,13  $\mu m$  low leakage technologie van Faraday Corporation [42])

| Element                  | Opp. $\left[\frac{\text{gate}}{\text{bit}}\right]$ |
|--------------------------|--|
| D flip-flop met reset    | 6  |
| D flip-flop zonder reset | 5,5  |
| full adder               | 5,5  |
| D latch                  | 4,25   |
| 3 ingang MUX             | 4  |
| 2 ingang XNOR            | 3,75   |
| 2 ingang XOR             | 3,75   |
| 2 ingang MUX             | 2,25   |
| 2 ingang OR              | 1,25   |
| 2 ingang AND             | 1,25   |
| 2 ingang NOR             | 1  |
| 2 ingang NAND            | 1  |
| NOT                      | 0,75   |

### 3.3 Modulaire Aritmetische Logische Unit

De kern van de hardware implementatie wordt gevormd door de Modulaire Aritmetische Logische Unit (MALU) [43, 44]. Dit circuit laat toe basis bewerkingen uit te voeren op getallen. Gezien de beperking die is opgelegd aan de oppervlakte van de schakeling, wordt enkel de optelling geïmplementeerd. Later wordt met behulp daarvan elke andere nodige berekening verwezenlijkt.

Aangezien er in het veld  $\mathbb{F}_{2^m}$  gewerkt wordt, is een optelling equivalent aan een XOR bewerking. De bewerking die moet uitgevoerd kunnen worden is:

$$\begin{aligned} T + B &= T \oplus B \\ &= R \pmod{P} \end{aligned}$$

Merk op dat bij een optelling de graad van  $R$  enkel kleiner of gelijk kan zijn aan die van  $T$  en  $B$ . Indien  $B$  van graad  $< m$  is (dus  $\in \mathbb{F}_{2^m}$ ) en  $T$  van graad  $\leq m$  ( $\in \mathbb{F}_{2^{m+1}}$ ), is de optelling te implementeren als in Algoritme 3.3.

In Paragraaf 3.5.2 zal blijken dat het vaak nodig zal zijn om het resultaat  $R$  te vermenigvuldigen met  $z$ , m.a.w. alle bits 1 plaats naar links te verschuiven. Een voor de hand liggende schakeling die dit alles implementeert, is te zien in Figuur 3.1. Ingang  $P_{\text{in}}$  dient afhankelijk van de graad van  $T$  ingesteld te worden op 0 of  $P$ . De ingangen  $T$  en  $P_{\text{in}}$  zijn  $m$  bits aangezien het resultaat voor de vermenigvuldiging met  $z$  steeds van graad  $< m$  is en bit  $m + 1$  dus toch steeds 0 zou zijn. De hoogste graad term na

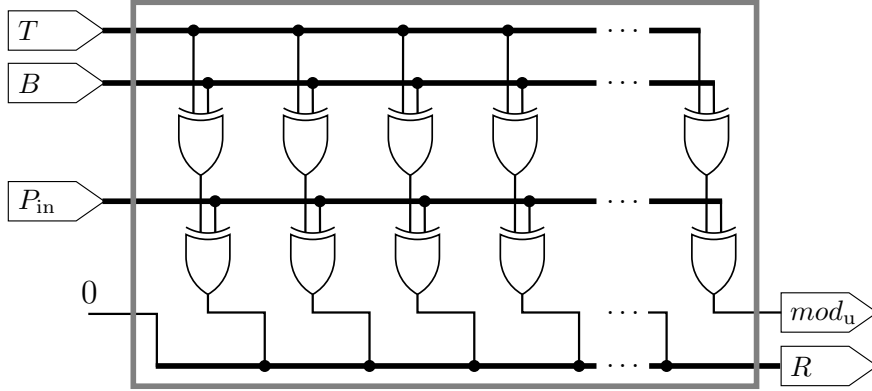
**Algoritme 3.3:** Modulo optelling in  $\mathbb{F}_{2^m}$ **Input:**  $B \in \mathbb{F}_{2^m}$ ,  $T \in \mathbb{F}_{2^{m+1}}$ **Output:**  $R \bmod P \in \mathbb{F}_{2^m}$ 

```

1  $R \leftarrow T \oplus B$ 
2 if  $\text{degree}(R) = m$  then
3    $R \leftarrow R \oplus P$ 
4 return  $R$ 

```

de shift wordt naar buiten gebracht als  $\text{mod}_u$ . De implementatie bestaat uit  $2m$  XOR poorten.



Figuur 3.1: MALU - Basis ontwerp met shift

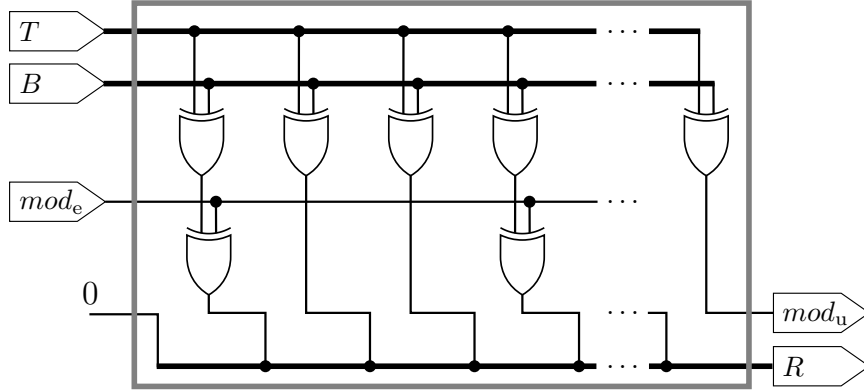
Aangezien voor het ontwerp het veld en de modulo veelterm op voorhand bepaald zijn, is het mogelijk een zeer groot aantal XOR poorten uit het ontwerp te verwijderen. De ingang  $P$  en de bijhorende  $m$  XOR poorten kunnen vervangen worden door een 1 bit ‘modulo enable’ ingang  $\text{mod}_e$  en er worden enkel XOR poorten geplaatst voor de bits  $i$  waarvoor  $P_i = 1$ . Hierdoor wordt het aantal ingangen drastisch verkleind en worden

$$\Delta = m - (\text{Hamm}(P) - 1)$$

XOR poorten uitgespaard.

In dit geval is  $m = 163$  en  $P = z^{163} + z^7 + z^6 + z^3 + 1$ . Er zijn dus  $\text{Hamm}(P) - 1 = 4$  XOR poorten nodig, wat een besparing van  $163 - 4 = 159$  XOR poorten oplevert (51% minder dan het oorspronkelijk aantal).

De resulterende schakeling is te zien in Figuur 3.2.



Figuur 3.2: MALU - Geoptimaliseerd ontwerp met shift

### 3.4 $\mathbb{F}_{2^m}$ kern

#### 3.4.1 Basisontwerp

De eerder ontworpen MALU schakeling laat toe optellingen te doen, maar het Miller algoritme vereist dat er ook vermenigvuldigingen worden uitgerekend. Delingen en machtsverheffingen kunnen met behulp van vermenigvuldiging berekend worden en dienen dus niet rechtstreeks geïmplementeerd te worden. Indien dus zowel optellingen als vermenigvuldigingen berekend kunnen worden, is alles voorhanden om de Tate pairing te berekenen.

Door toepassing van een “shift and add” algoritme, kan de waarde van  $A \cdot B = R$  berekend worden met behulp van de MALU schakeling. In Algoritme 3.4 is te zien hoe dit juist in z’n werk gaat. Door de modulobewerking telkens op het tussenresultaat uit te voeren, is het steeds van graad  $\leq m$  en kan het opgeslagen worden in  $T$ . Op het einde moet het resultaat door  $z$  gedeeld worden, wat neerkomt op een verschuiving van alle bits met 1 plaats naar rechts.

Wanneer de optelling en vermenigvuldiging nu in een schakeling gegoten worden, dient deze te weten welke van de twee bewerkingen moet uitgevoerd worden. Verder moet het mogelijk zijn de uitkomst  $R$  in het register  $T$  op te slaan. Op die manier is het mogelijk de uitgang van de schakeling gelijk te stellen aan  $R$  zolang geen nieuwe berekening gestart wordt.

Verderop zal gezien worden dat in het Miller algoritme verscheidene keren de som  $R + 1$  moet berekend worden. Daarom wordt aan de schakeling een ingang *plus\_one* toegevoegd die hierin helpt voorzien. De uiteindelijke schakeling is te zien in Figuur 3.3. Er wordt zo veel mogelijk bespaard op registers. Het register *cycle* (equivalent aan  $i$  in Algoritme 3.4) is  $\lceil \log_2(m) \rceil$  bits lang en register  $T$   $m$  bits. De waarde van  $T_m$  wordt opgeslagen in register *mod*. Alle overige registers zijn 1 bit groot. Merk dus op dat de variabele

**Algoritme 3.4:** “Shift and add” vermenigvuldiging in  $\mathbb{F}_{2^m}$ **Input:**  $A, B \in \mathbb{F}_{2^m}$ **Output:**  $R = A \cdot B \in \mathbb{F}_{2^m}$ **Data:**  $T \in \mathbb{F}_{2^{m+1}}$ 

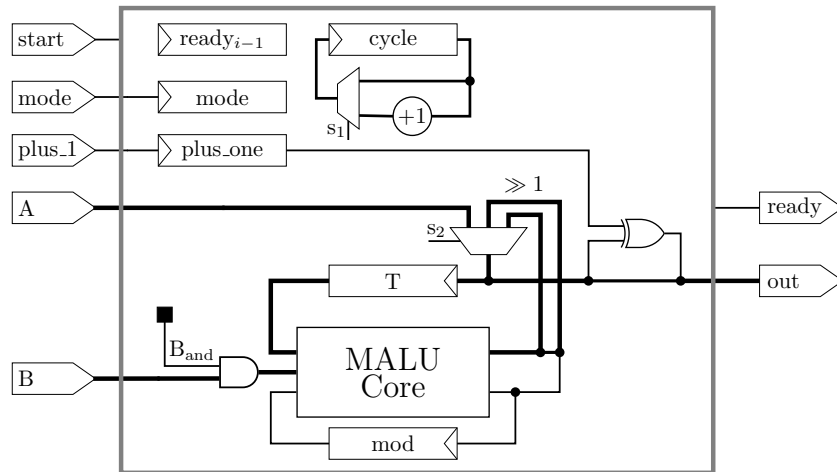
```

1  $T \leftarrow 0$ 
2 for  $i \leftarrow m - 1$  to 0 do
3   if  $A_i = 1$  then
4      $b \leftarrow B$ 
5   else
6      $b \leftarrow 0$ 
7    $T \leftarrow T \oplus b$ 
8   if  $\text{degree}(T) = m$  then
9      $T \leftarrow T \oplus P$ 
10   $T \leftarrow T \ll 1$ 
11  $R \leftarrow T \gg 1$ 

```

$T$  uit Algoritme 3.4 hier gevormd wordt door de combinatie van de registers  $T$  en  $\text{mod}$ .

Voor  $A$  wordt geen apart register voorzien en in plaats van  $A_i$  wordt steeds bit  $A_m$  ingelezen voor de vermenigvuldiging. De schakeling die gekoppeld is aan de ingang van deze schakeling dient dus te voorzien in een methode om elke klokslag de juiste  $A_i$  aan te bieden op  $A_m$ . Dit kan simpelweg gebeuren door elke klokslag na de start van de berekening het register dat  $A$  bevat één positie naar links door te schuiven.

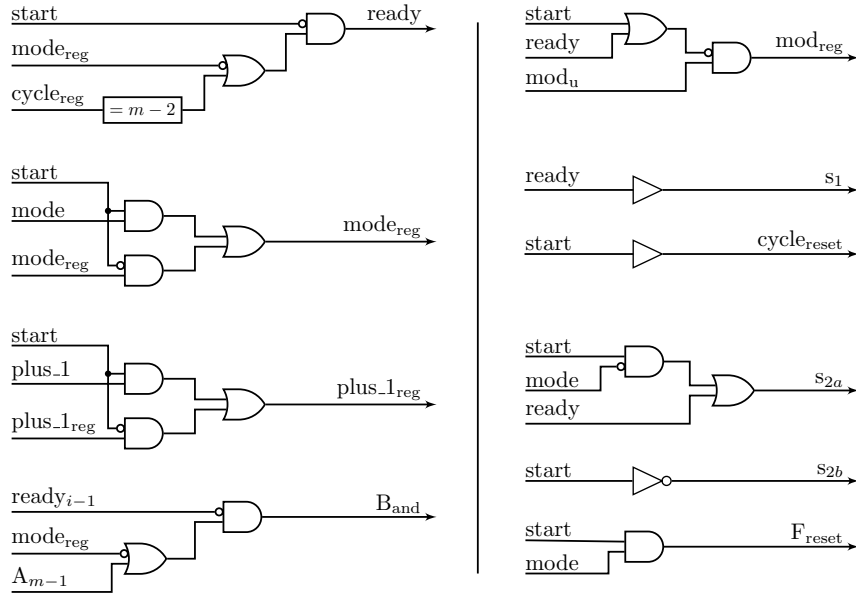


Figuur 3.3: Schakeling voor  $\mathbb{F}_{2^m}$  kern

Gezien de eenvoud van de schakeling is het niet nodig een FSM te imple-



menteren, de besturing kan volledig via logica gebeuren. Die wordt getoond in Figuur 3.4. De werking is zeer eenvoudig: zo lang *start* hoog is, worden de registers *mode* en *plus\_one* geladen met hun respectievelijke ingangen. Verder wordt, afhankelijk van *mode*, *F* ingeladen met de waarde van *A* (optelling, *mode* = 0) of gelijkgesteld aan nul (vermenigvuldiging, *mode* = 1). Ook wordt *cycle* gereset. Wanneer *start* laag is, wordt, afhankelijk van de gewenste bewerking en de waarde van *ready*, het register *T* geladen met de uitgang *R* van de MALU of het uiteindelijke resultaat  $R_{\text{ready}} = R \gg 1$ .

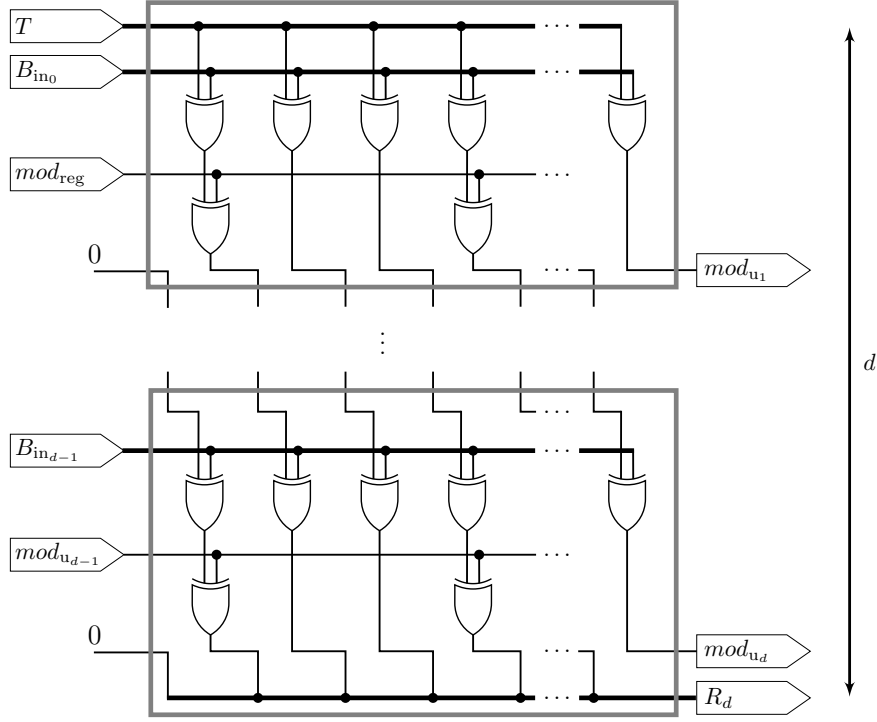


Figuur 3.4: Logica voor besturing van de  $\mathbb{F}_{2^m}$  kern

### 3.4.2 Versnelling van de vermenigvuldiging

Wanneer met behulp van de schakeling in Figuur 3.3 een vermenigvuldiging wordt berekend, zal het *m* klokcycli duren eer het resultaat beschikbaar is aan de uitgang. Het is echter mogelijk dat aantal drastisch naar beneden te halen door *d* MALU's te gebruiken en dus *d* optellingen per klokcyclus uit te voeren. Het principe hiervan wordt geïllustreerd in Figuur 3.5.

De rekentijd van het Miller algoritme zal door toepassing van deze techniek gevoelig verkort kunnen worden. Hoe groter *m* en hoe meer vermenigvuldigingen er uitgevoerd dienen te worden, des te signifikanter de geboekte tijds winst. Uiteraard gaat het gebruik van deze techniek wel in tegen de eerder opgelegde beperking aan de grootte van de uiteindelijke schakeling. Het is echter niet zo dat er enkel *d* − 1 extra MALU blokken dienen toegevoegd te worden, afhankelijk van *d* en *m* dient ook een extra multiplexer in



Figuur 3.5: Aaneenschakeling van  $d$  MALU's ter versnelling van de vermenigvuldiging

de schakeling gestoken te worden. Dit is zoals opgemerkt in Paragraaf 3.2 een zeer slechte zaak voor de oppervlakte.

Stel bijvoorbeeld  $d = 4$  (en  $m = 163$ ). Het resultaat van een optelling zal net zoals in het standaard ontwerp (Figuur 3.3) aanwezig zijn aan de uitgang van MALU n° 1. Het resultaat van een vermenigvuldiging zal echter aan de uitgang van MALU n° 3 verschijnen, aangezien  $163 \bmod 4 = 3$ . Het eindresultaat dat in register  $T$  dient opgeslagen te worden, is voor een vermenigvuldiging dus

$$R_{3_{\text{ready}}} = mod_{u_3} \# R_{3_{162:1}},$$

terwijl dit voor een optelling

$$R_{1_{\text{ready}}} = mod_{u_1} \# R_{1_{162:1}}$$

is. Met andere woorden, als ingang voor  $T$  dient nu niet enkel gekozen te kunnen worden tussen de ingangen  $A$ ,  $R_d$  of  $R_{1_{\text{ready}}}$ , maar ook voor  $R_{3_{\text{ready}}}$ .

Indien men toch wenst het vermenigvuldigen te versnellen, is het aangeraden een  $d$  te kiezen waarvoor  $m \bmod d = 1$ . Als voorbeeld worden enkele voor de hand liggende en optimale keuzes vergeleken voor  $d$  indien  $m = 163$  in Tabel 3.2.

Tabel 3.2: Nummer  $m \bmod d$  van de MALU waar het resultaat van een vermenigvuldiging zich bevindt voor  $d$  MALU's indien  $m = 163$

| Voor de hand liggende waarden voor $d$ |   |   |   |    |    |    |
|--|---|---|---|----|----|----|
| $d$                                    | 2 | 4 | 8 | 16 | 32 | 64 |
| $m \bmod d$                            | 1 | 3 | 3 | 3  | 3  | 35 |

| Optimale waarden voor $d$ |   |   |   |   |    |    |
|---------------------------|---|---|---|---|----|----|
| $d$                       | 2 | 3 | 6 | 9 | 18 | 27 |
| $m \bmod d$               | 1 | 1 | 1 | 1 | 1  | 1  |

### 3.5 Controller voor het Miller algoritme

#### 3.5.1 Algemeen ontwerp

Nu een schakeling voorhanden is die toelaat alle benodigde berekeningen uit te voeren, rest nog een schakeling te ontwerpen die het Miller algoritme (zie Paragraaf 2.3) uitvoert. Het algoritme met invulling van de vastgelegde parameters, zonder uitwerking van de berekeningen, wordt gegeven in Algoritme 3.5.

---

**Algoritme 3.5:** Miller algoritme voor de Tate pairing met parameters ingevuld

---

**Input:**  $P, Q \in E(\mathbb{F}_{2^{163}})[2^{163} + 2^7 + 2^6 + 2^3 + 1]$   
**Output:**  $\hat{e}(P, Q) \in \mathbb{F}_{2^{4 \cdot 163}}^*$

```

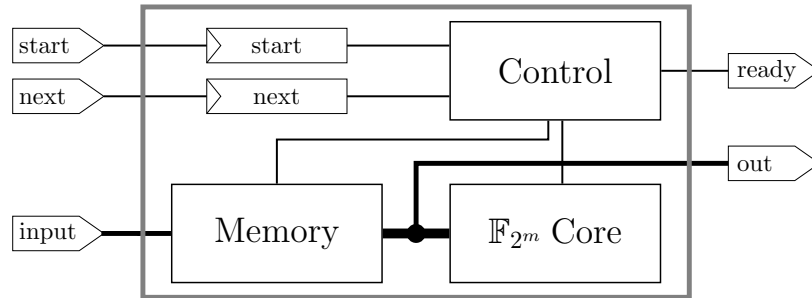
1  $F \leftarrow 1$ 
2  $V \leftarrow P$ 
3 for  $i \leftarrow 162$  tot 0 do
4    $F \leftarrow F^2 \cdot G_{V,V}(\phi(Q))$ 
5    $V \leftarrow 2 \cdot V$ 
6   if  $i = 82$  then
7      $F \leftarrow F \cdot G_{V,P}(\phi(Q))$ 
8      $V \leftarrow V + P$ 
9  $F \leftarrow F^{\frac{2^{4 \cdot 163} - 1}{2^{163} + 2^{82} + 1}}$ 
10 return  $F$ 
```

---

Merk op dat op lijn 6 slechts één waarde moet nagekeken worden, aangezien  $l = 2^{163} + 2^{82} + 1$ .

De controller zal ontworpen worden zoals het schema in Figuur 3.6. Voor het geheugenblok volledig ontworpen kan worden, moeten echter eerst de verschillende berekeningen uitgewerkt worden. Vervolgens zal aan de hand

van die uitwerkingen bepaald worden hoeveel registers ten minste noodzakelijk zijn. Afhankelijk van diezelfde uitwerkingen zullen ook enkele geheugen elementen voor tellers en statusbits toegevoegd moeten worden. Ten slotte zal een FSM ontworpen worden.



Figuur 3.6: Schakeling voor de uitvoering van het Miller algoritme

Grofweg kan het algoritme opgedeeld worden in de for-lus en een finale machtsverheffing. De for-lus kan verder onderverdeeld worden in een verdubbelstap, een optelstap, een kwadratering van  $F$  en een vermenigvuldiging  $F \cdot G$ . Elk van deze stappen zal verder uitgediept worden en er zal voor elke berekening bepaald worden hoeveel tussenresultaten minimum opgeslagen moeten worden. Het zal blijken dat een inversie in  $\mathbb{F}_{2^m}$  uitgerekend moet kunnen worden, wat ook verder uitgediept zal worden.

Bij elk van de volgende algoritmen zal aangegeven worden hoeveel en welke bewerkingen juist nodig zijn. Daarbij staat  $A$  voor een optelling,  $M$  voor een vermenigvuldiging,  $S$  voor een kwadratering en  $I$  voor een inversie. Aangezien er echter geen afzonderlijke schakeling voor kwadrateren ontworpen is, zijn  $S$  en  $M$  qua rekentijd in dit geval equivalent aan elkaar. De bewerking  $a + 1$  neemt geen extra tijd in beslag, omdat die functie parallel met een optelling of vermenigvuldiging kan uitgevoerd worden door de *plus\_one* van de  $\mathbb{F}_{2^m}$  kern hoog te maken bij de start van een berekening.

### 3.5.2 For-lus

Zoals reeds eerder vermeld kan de for-lus onderverdeeld worden in een verdubbelstap, een optelstap, een kwadratering van  $F$  en een vermenigvuldiging  $F \cdot G$ . Elk van deze onderdelen zal in de volgende paragrafen in detail aan bod komen.

### Verdubbelstap

De verdubbelstap wordt gevormd door lijnen 4 en 5 in Algoritme 3.5. Voor een supersinguliere kromme zijn de berekeningen als volgt [45, 39]:

$$\begin{cases} \lambda = x_V^2 + 1 \\ x_{2V} = \lambda^2 \\ y_{2V} = \lambda \cdot (x_{2V} + x_V) + y_V + 1 \\ G_{V,V}(\phi(Q)) = \lambda \cdot (x_\phi + x_V) + (y_\phi + y_V) \end{cases}$$

Het is ook mogelijk  $y_{2V}$  te berekenen als:

$$\begin{aligned} y_{2V} &= y_V^4 + x_V^4 \\ &= (y_V + x_V)^4 \end{aligned}$$

Aangezien dit echter twee kwadrateringen en een optelling kost tegenover een vermenigvuldiging en twee optellingen, wordt de voorkeur gegeven aan de eerste methode.

Door de specifieke vorm van  $\phi(Q)$  kan  $G$  uitgeschreven worden als:

$$\begin{aligned} G_a &= \lambda \cdot (x_{\phi_a} + x_V) + (y_{\phi_a} + y_V) & G_c &= \lambda \cdot x_{\phi_c} + y_{\phi_c} \\ G_b &= \lambda \cdot x_{\phi_b} + y_{\phi_b} & &= 0 \\ &= \lambda + y_{\phi_b} & G_d &= \lambda \cdot x_{\phi_d} + y_{\phi_d} \\ &= \lambda + x_{\phi_a} & &= 1 \end{aligned}$$

De variabele  $G$  kan dus opgeslagen worden in twee registers van grootte  $m$  in plaats van in vier. De vorm van  $G$  zal ook toelaten de vermenigvuldiging  $F \cdot G$  grotendeels te vereenvoudigen, zoals later gezien zal worden.

Wanneer dit in rekening gebracht wordt en het algoritme op register niveau wordt uitgeschreven, bekomt men uiteindelijk Algoritme 3.6. Hierbij werd specifiek gelet op een minimum gebruik van tijdelijke registers.

Buiten zes registers voor  $x_{2V}$ ,  $y_{2V}$ ,  $x_{\phi_a}$ ,  $y_{\phi_a}$ ,  $G_a$  en  $G_b$ , is er ook één register nodig om  $\lambda$  in op te slaan. In totaal moeten er zes optellingen, twee vermenigvuldigingen en twee kwadrateringen uitgerekend worden.

### Optelstap

De optelstap bestaat uit lijnen 7 en 8 van Algoritme 3.5. Voor een supersinguliere kromme dienen de volgende bewerkingen uitgevoerd te worden [45, 39]:

$$\begin{cases} \lambda = \frac{y_V + y_P}{x_V + x_P} \\ x_{V+P} = \lambda^2 + x_V + x_P \\ y_{V+P} = \lambda \cdot (x_{V+P} + x_P) + y_P + 1 \\ G_{V,P}(\phi(Q)) = \lambda \cdot (x_\phi + x_P) + (y_\phi + y_P) \end{cases}$$

---

**Algoritme 3.6:** Uitwerking van de verdubbelstap voor supersinguliere krommen in het Miller algoritme

---

**Input:**  $x_V, y_V \in E(\mathbb{F}_{2^m}); x_{\phi_a}, y_{\phi_a} \in \mathbb{F}_{2^m}$

**Output:**  $x_{2V}, y_{2V} \in E(\mathbb{F}_{2^m}); G \in \mathbb{F}_{2^{4m}}$

**Data:**  $\lambda \in \mathbb{F}_{2^m}$

|   |  |          |
|---|--|----------|
| 1 | $G_a \leftarrow x_V; G_b \leftarrow y_V$                                 |          |
| 2 | $\lambda \leftarrow G_a^2 + 1; x_{2V} \leftarrow \lambda^2$              | 2 S      |
| 3 | $y_{2V} \leftarrow x_{2V} + G_a; y_{2V} \leftarrow y_{2V} \cdot \lambda$ | 1 M, 1 A |
| 4 | $y_{2V} \leftarrow y_{2V} + G_b + 1$                                     | 1 A      |
| 5 | $G_a \leftarrow G_a + x_{\phi_a}; G_a \leftarrow G_a \cdot \lambda$      | 1 M, 1 A |
| 6 | $G_a \leftarrow G_a + y_{\phi_a}; G_a \leftarrow G_a + G_b$              | 2 A      |
| 7 | $G_b \leftarrow \lambda + x_{\phi_a}$                                    | 1 A      |

---

Net zoals bij de verdubbelstap kan  $G$  hier in twee variabelen opgeslagen worden. Hoewel de optelstap slechts één maal moet worden uitgevoerd, is het uiteraard cruciaal dat ook hier zo weinig mogelijk tijdelijke variabelen gebruikt worden. Op die manier blijft de grootte van de uiteindelijke schakeling het kleinst. De uitgewerkte versie van het algoritme wordt gegeven in Algoritme 3.7.

In tegenstelling tot de verdubbelstap zijn hier twee tijdelijke registers nodig, een voor  $\lambda$  en een voor  $a$ . Verder zijn er twee registers nodig voor  $x_P$  en  $y_P$ . Alles samen dienen er tien optellingen, drie vermenigvuldigingen, twee kwadrateringen en een inversie uitgerekend te worden.

---

**Algoritme 3.7:** Uitwerking van de optelstap voor supersinguliere krommen in het Miller algoritme

---

**Input:**  $x_V, y_V, x_P, y_P \in E(\mathbb{F}_{2^m}); x_{\phi_a}, y_{\phi_a} \in \mathbb{F}_{2^m}$

**Output:**  $x_{V+P}, y_{V+P} \in E(\mathbb{F}_{2^m}); G \in \mathbb{F}_{2^{4m}}$

**Data:**  $\lambda, a \in \mathbb{F}_{2^m}$

|   |  |          |
|---|--|----------|
| 1 | $G_a \leftarrow x_V; G_b \leftarrow y_V$                                     |          |
| 2 | $\lambda \leftarrow G_a + x_P; \lambda \leftarrow \lambda^{-1}$              | 1 I, 1 A |
| 3 | $a \leftarrow G_b + y_P; \lambda \leftarrow \lambda \cdot a$                 | 1 M, 1 A |
| 4 | $x_{V+P} \leftarrow \lambda^2 + G_a; x_{V+P} \leftarrow x_{V+P} + x_P$       | 1 S, 2 A |
| 5 | $y_{V+P} \leftarrow x_{V+P} + x_P; y_{V+P} \leftarrow y_{V+P} \cdot \lambda$ | 1 M, 1 A |
| 6 | $y_{V+P} \leftarrow y_{V+P} + y_P + 1$                                       | 1 A      |
| 7 | $G_a \leftarrow x_{\phi_a} + x_P; G_a \leftarrow G_a \cdot \lambda$          | 1 M, 1 A |
| 8 | $G_a \leftarrow G_a + y_{\phi_a}; G_a \leftarrow G_a + y_P$                  | 2 A      |
| 9 | $G_b \leftarrow \lambda + x_{\phi_a}$  | 1 A      |

---

### Inversie

De meest tijdrovende stap in de optelstap is de inversie. Een inversie in een Galoisveld kan berekend worden door toepassing van de kleine stelling van Fermat:

$$\begin{aligned} a^{2^m} &= a \\ a^{2^m-1} &= 1 \\ a^{2^m-2} &= a^{-1} \end{aligned}$$

De naïeve manier om dit te berekenen zou zijn om  $a$  in totaal  $2^m - 2$  maal met zichzelf te vermenigvuldigen. In dit geval zou dat betekenen dat er  $2^{163} - 2$  vermenigvuldigingen moeten uitgevoerd worden, wat uiteraard onhaalbaar is.

Een tweede manier bestaat er in de exponent te ontbinden in machten van 2 en 3. In dat geval zouden er nog 237 vermenigvuldigingen nodig zijn.

Er is echter een derde, optimale manier die toegepast kan worden indien de exponent van de vorm  $2^m - 2$  is [46, 47]. Dit gaat als volgt in zijn werk. Stel:

$$a^{2^m-2} = (a^{2^{m-1}-1})^2$$

Als wordt aangenomen dat  $m$  oneven is, is de exponent van twee na het gelijkheidsteken dus even. Zolang de exponent even is, kan recursief volgende formule toegepast worden:

$$a^{2^i-1} = (a^{2^{\frac{i}{2}-1})^{2^{\frac{i}{2}}} \cdot a^{2^{\frac{i}{2}-1}}$$

Indien  $a$  oneven is, dient volgende formule toegepast te worden:

$$a^{2^i-1} = (a^{2^{i-1}-1})^2 \cdot a$$

Uiteindelijk eindigt men dan bij  $a^2$ . Het totaal aantal bewerkingen voor een inversie in  $\mathbb{F}_{2^m}$  is  $\lfloor \log_2(m-1) \rfloor + \text{Hamm}(m-1) + 1$  vermenigvuldigingen en  $m-1$  kwadrateringen.

In het geval van  $m = 163$  is de uiteindelijke keten van bewerkingen zoals gegeven in Algoritme 3.8. Het aantal berekeningen in dat geval is 9 vermenigvuldigingen en 162 kwadrateringen. Er is een register nodig om  $a$  bij de houden en twee voor de tussenresultaten  $a^{2^i-1}$  en  $(a^{2^i-1})^{2^i}$ .

### Kwadratering van $F$

Bij het uitvoeren van lijn 4 van Algoritme 3.5 moet ook telkens het kwadraat van  $F$  berekend worden. De afleiding van de formule daarvoor gaat als volgt:

---

|  |   |           |
|--|---|-----------|
| <b>Algoritme 3.8:</b> Inversie in $\mathbb{F}_{2^{163}}$ |   |           |
| <b>Input:</b> $a \in \mathbb{F}_{2^{163}}$               |   |           |
| <b>Output:</b> $a^{-1} \in \mathbb{F}_{2^{163}}$         |   |           |
| 1  | $a^3 \leftarrow a^2 \cdot a$  | 1 S, 1 M  |
| 2  | $a^{2^4-1} \leftarrow (a^3)^{2^2} \cdot a^3$                          | 2 S, 1 M  |
| 3  | $a^{2^5-1} \leftarrow (a^{2^4-1})^2 \cdot a$                          | 1 S, 1 M  |
| 4  | $a^{2^{10}-1} \leftarrow (a^{2^5-1})^{2^5} \cdot a^{2^5-1}$           | 5 S, 1 M  |
| 5  | $a^{2^{20}-1} \leftarrow (a^{2^{10}-1})^{2^{10}} \cdot a^{2^{10}-1}$  | 10 S, 1 M |
| 6  | $a^{2^{40}-1} \leftarrow (a^{2^{20}-1})^{2^{20}} \cdot a^{2^{20}-1}$  | 20 S, 1 M |
| 7  | $a^{2^{80}-1} \leftarrow (a^{2^{40}-1})^{2^{40}} \cdot a^{2^{40}-1}$  | 40 S, 1 M |
| 8  | $a^{2^{81}-1} \leftarrow (a^{2^{80}-1})^2 \cdot a$                    | 1 S, 1 M  |
| 9  | $a^{2^{162}-1} \leftarrow (a^{2^{81}-1})^{2^{81}} \cdot a^{2^{81}-1}$ | 81 S, 1 M |
| 10   | $a^{-1} \leftarrow (a^{2^{162}-1})^2$                                 | 1 S       |

---

$$\begin{aligned}
F^2 &= (F_a + F_b x + F_c y + F_d xy) \cdot (F_a + F_b x + F_c y + F_d xy) \\
&= F_a^2 + F_a F_b x + F_a F_c y + F_a F_d xy + F_b F_a x + F_b^2 x^2 + F_b F_c xy \\
&\quad + F_b F_d x^2 y + F_c F_a y + F_c F_b xy + F_c^2 y^2 + F_c F_d xy^2 + F_d F_a xy \\
&\quad + F_d F_b x^2 y + F_d F_c xy^2 + F_d^2 x^2 y^2 \\
&= (F_a^2 + F_b^2 + F_c^2 + F_d^2) \\
&\quad + (F_a F_b + F_b F_a + F_b^2 + F_c F_d + F_d F_c + F_d^2) x \\
&\quad + (F_a F_c + F_b F_d + F_c F_a + F_c^2 + F_c F_d + F_d F_b + F_d F_c) y \\
&\quad + (F_a F_d + F_b F_c + F_b F_d + F_c F_b + F_c^2 + F_d F_a + F_d F_b + F_d^2) xy \\
&= (F_a^2 + F_b^2 + F_c^2 + F_d^2) + (F_b^2 + F_d^2) x + F_c^2 y + (F_c^2 + F_d^2) xy
\end{aligned}$$

Mits de originele waarde van  $F$  overschreven mag worden, is het mogelijk dit te berekenen zonder gebruik van tijdelijke variabelen. Er zijn dus enkel vier registers nodig voor  $F$ . Hoe dat in z'n werk gaat is te zien in Algoritme 3.9. Eén kwadratering van  $F$  vraagt vier optellingen en vier kwadrateringen in  $\mathbb{F}_{2^m}$ .

#### Vermenigvuldiging $F \cdot G$

Zoals eerder opgemerkt, is  $G$  in zowel de verdubbel- als optelstep niet van volledige rang in het extensieveld. De vermenigvuldiging van  $F$  met  $G$  kan daardoor vereenvoudigd worden, namelijk als volgt:



---

|  |                            |     |
|--|----------------------------|-----|
| <b>Algoritme 3.9:</b> Uitwerking van van $F^2 \in \mathbb{F}_{2^{4m}}$ |                            |     |
| <hr/>  |                            |     |
| <b>Input:</b> $F = F_a + F_bx + F_cy + F_dxy \in \mathbb{F}_{2^{4m}}$  |                            |     |
| <b>Output:</b> $F = F^2 \in \mathbb{F}_{2^{4m}}$                       |                            |     |
| 1  | $F_a \leftarrow F_a + F_c$ | 1 A |
| 2  | $F_a \leftarrow F_a^2$     | 1 S |
| 3  | $F_b \leftarrow F_b + F_d$ | 1 A |
| 4  | $F_b \leftarrow F_b^2$     | 1 S |
| 5  | $F_a \leftarrow F_a + F_b$ | 1 A |
| 6  | $F_c \leftarrow F_c^2$     | 1 S |
| 7  | $F_d \leftarrow F_d^2$     | 1 S |
| 8  | $F_d \leftarrow F_d + F_c$ | 1 A |

---

$$\begin{aligned}
F \cdot G &= (F_a + F_bx + F_cy + F_dxy) \cdot (G_a + G_bx + xy) \\
&= F_aG_a + F_aG_bx + F_axy + F_bG_ax + F_bG_bx^2 + F_bx^2y + F_cG_ay \\
&\quad + F_cG_bxy + F_cxy^2 + F_dG_axy + F_dG_bx^2y + F_dx^2y^2 \\
&= (F_aG_a + F_bG_b + F_d) \\
&\quad + (F_aG_b + F_bG_a + F_bG_b + F_c + F_d)x \\
&\quad + (F_b + F_cG_a + F_c + F_dG_b)y \\
&\quad + (F_a + F_b + F_cG_b + F_dG_a + F_dG_b + F_d)xy
\end{aligned}$$

Indien hier nu de Karatsuba-Ofman techniek [48, 49] op wordt toegepast, bekomt men:

$$\begin{aligned}
F \cdot G &= (F_aG_a + F_bG_b + F_d) \\
&\quad + ((F_a + F_b) \cdot (G_a + G_b) + F_aG_a + F_c + F_d)x \\
&\quad + (F_cG_a + F_dG_b + F_b + F_c)y \\
&\quad + ((F_c + F_d) \cdot (G_a + G_b) + F_cG_a + F_a + F_b + F_d)xy
\end{aligned}$$

Deze formule kan uitgerekend wordt met gebruik van drie tijdelijke registers ( $a$ ,  $b$  en  $c$ ). Verder zijn er vier registers nodig voor  $F$  en twee voor  $G$ . Merk op dat de oude waarde van  $F$  overschreven wordt door het resultaat. In totaal zijn er zes vermenigvuldigingen en veertien optellingen nodig. Algoritme 3.10 beschrijft welke berekeningen juist uitgevoerd moeten worden.

Mits het gebruik van een vierde tijdelijk register zou het mogelijk zijn de berekening  $G_a + G_b$  op te slaan. Die wordt nu zowel in lijn 2 als 7 berekend. Er zouden dan slechts dertien optellingen moeten berekend worden, één minder dan [50], waar  $y^2 + y + x$  gebruikt wordt als modulo veelterm voor het extensieveld. Aangezien een extra tijdelijk register echter tegen de doelstellingen ingaat, wordt voor de iets langere berekening gekozen.

---

**Algoritme 3.10:** Uitwerking van de vermenigvuldiging  $F \cdot G$  in het Miller algoritme

---

**Input:**  $F = F_a + F_b x + F_c y + F_d xy, G = G_a + G_b x + xy \in \mathbb{F}_{2^{4m}}$

**Output:**  $F = F \cdot G \in \mathbb{F}_{2^{4m}}$

**Data:**  $a, b, c \in \mathbb{F}_{2^m}$

|    |  |          |
|----|--|----------|
| 1  | $a \leftarrow F_a \cdot G_a; a \leftarrow a + F_d$                       | 1 M, 1 A |
| 2  | $b \leftarrow F_a + F_b; c \leftarrow G_a + G_b$                         | 2 A      |
| 3  | $b \leftarrow b \cdot c; b \leftarrow b + a; b \leftarrow b + F_c$       | 1 M, 2 A |
| 4  | $c \leftarrow F_b \cdot G_b; a \leftarrow a + c$                         | 1 M, 1 A |
| 5  | $c \leftarrow F_c \cdot G_a; c \leftarrow c + F_b$                       | 1 M, 1 A |
| 6  | $F_b \leftarrow b; b \leftarrow c$                                       |          |
| 7  | $c \leftarrow F_c + F_d; G_a \leftarrow G_a + G_b$                       | 2 A      |
| 8  | $c \leftarrow c \cdot G_a; c \leftarrow c + b; c \leftarrow c + F_a$     | 1 M, 2 A |
| 9  | $F_a \leftarrow a$   |          |
| 10 | $c \leftarrow c + F_d; b \leftarrow b + F_c; a \leftarrow F_d \cdot G_b$ | 1 M, 2 A |
| 11 | $F_c \leftarrow b + a; F_d \leftarrow c$                                 | 1 A      |

---

### 3.5.3 Finale machtsverheffing

Eens de for-lus voltooid is, moet  $F$  nog gereduceerd worden zodat het eindresultaat  $\hat{e}(P, Q)$  uniek is. Hoe dit gebeurt, wordt onderzocht in de volgende paragrafen. De gebruikte methodes zijn gebaseerd op die in [50], aangepast aan het gekozen extensieveld en geoptimaliseerd om het registerverbruik zo laag mogelijk te houden.

De reductie op het einde van het Miller algoritme bestaat uit de machtsverheffing  $\hat{e}(P, Q) = F^M$ , met

$$\begin{aligned}
 M &= \frac{2^{4m} - 1}{l} \\
 &= \frac{(2^{2m} + 1)(2^{2m} - 1)}{l} \\
 &= (2^{2m} - 1)(2^m - \nu 2^{\frac{m+1}{2}} + 1) \\
 &= (2^{2m} - 1)(2^m + 1) + \nu(1 - 2^{2m})2^{\frac{m+1}{2}}
 \end{aligned}$$

Aangezien  $\nu = 1$  in dit geval, kan de machtsverheffing dus berekend worden als

$$\hat{e}(P, Q) = \left(F^{2^{2m}-1}\right)^{2^m+1} \cdot \left(F^{1-2^{2m}}\right)^{2^{\frac{m+1}{2}}}$$

Er zal onderzocht worden hoe elk van deze termen berekend kan worden. Stel

$$\begin{aligned}
 F &= (F_a + F_b x) + (F_c + F_d x)y \\
 &= U_0 + U_1 y,
 \end{aligned}$$

met  $U_0, U_1 \in \mathbb{F}_{2^{2m}}$ . Met  $y^{2^{2m}} = y + x + 1$  is  $F^{2^{2m}} = U_0 + U_1 + U_1x + U_1y$ . Men vindt dus:

$$\begin{aligned} I &= F^{2^{2m}-1} = \frac{F^{2^{2m}}}{F} \\ &= \frac{U_0 + U_1 + U_1x + U_1y}{U_0 + U_1y} \\ &= \frac{(U_0 + U_1 + U_1x + U_1y)^2}{(U_0 + U_1 + U_1x + U_1y) \cdot (U_0 + U_1y)} \\ &= \frac{U_0^2 + U_1^2 + U_1^2x}{U_0^2 + U_1^2 + U_0U_1 + U_0U_1x} + \left[ \frac{U_1^2 + U_1^2x}{U_0^2 + U_1^2 + U_0U_1 + U_0U_1x} \right] y \end{aligned}$$

en

$$\begin{aligned} J &= F^{1-2^{2m}} = \frac{F}{F^{2^{2m}}} \\ &= \frac{U_0 + U_1y}{U_0 + U_1 + U_1x + U_1y} \\ &= \frac{(U_0 + U_1y)^2}{(U_0 + U_1 + U_1x + U_1y) \cdot (U_0 + U_1y)} \\ &= \frac{U_0^2 + U_1^2}{U_0^2 + U_1^2 + U_0U_1 + U_0U_1x} + \left[ \frac{U_1^2 + U_1^2x}{U_0^2 + U_1^2 + U_0U_1 + U_0U_1x} \right] y \end{aligned}$$

Er moeten dus vier termen in  $\mathbb{F}_{2^{2m}}$  berekend worden. Merk echter op dat de noemers van alle breuken in zowel  $I$  als  $J$  identiek zijn. Er zal dus slechts één inversie in  $\mathbb{F}_{2^{2m}}$  uitgerekend moeten worden. Ook zijn de tweede termen van  $I$  en  $J$  identiek. In totaal moeten dus drie elementen in  $\mathbb{F}_{2^{2m}}$  opgeslagen worden alvorens de termen vermenigvuldigd kunnen worden. Dit wil zeggen dat er dus zes registers van grootte  $m$  nodig zijn.

### Termen van de deelbreuken

Ten eerste worden de drie tellers  $I_t$ ,  $J_t$  en  $G_t$  uitgewerkt. Hierbij staan  $I_t$  en  $J_t$  voor de teller van de eerste breuk van respectievelijk  $I$  en  $J$ ,  $G_t$  staat voor de teller van de gemeenschappelijke tweede breuk en  $G_n$  voor de gemeenschappelijke noemer. Met andere woorden:

$$\begin{aligned} I &= \frac{I_t}{G_n} + \left[ \frac{G_t}{G_n} \right] y \\ J &= \frac{J_t}{G_n} + \left[ \frac{G_t}{G_n} \right] y \end{aligned}$$

Het kwadraat van een element  $A \in \mathbb{F}_{2^{2m}}$  is

$$\begin{aligned} A^2 &= a_0^2 + a_1^2x^2 \\ &= (a_0^2 + a_1^2) + a_1^2x \end{aligned}$$

en de vermenigvuldiging van  $A, B \in \mathbb{F}_{2^{2m}}$ :

$$\begin{aligned} A \cdot B &= a_0b_0 + a_0b_1x + a_1b_0x + a_1b_1x^2 \\ &= (a_0b_0 + a_1b_1) + (a_0b_1 + a_1b_0 + a_1b_1)x \end{aligned}$$

Zodoende bekomt men:

$$\begin{aligned} I_t &= [(F_a^2 + F_b^2) + F_b^2x] + [(F_c^2 + F_d^2) + F_d^2x] + [F_d^2 + F_c^2x] \\ &= (F_a^2 + F_b^2 + F_c^2) + (F_b^2 + F_c^2 + F_d^2)x \\ J_t &= [(F_a^2 + F_b^2) + F_b^2x] + [(F_c^2 + F_d^2) + F_d^2x] \\ &= (F_a^2 + F_b^2 + F_c^2 + F_d^2) + (F_b^2 + F_d^2)x \\ G_t &= [(F_c^2 + F_d^2) + F_d^2x] + [F_d^2 + F_c^2x] \\ &= F_c^2 + (F_c^2 + F_d^2)x \end{aligned}$$

Voor de gemeenschappelijke noemer  $G_n$  vindt men:

$$\begin{aligned} G_n &= [(F_a^2 + F_b^2) + F_b^2x] + [(F_c^2 + F_d^2) + F_d^2x] \\ &\quad + [(F_aF_c + F_bF_d) + (F_aF_d + F_bF_c + F_bF_d)x] \\ &\quad + [F_aF_d + F_bF_c + F_bF_d) + (F_aF_c + F_aF_d + F_bF_c)x] \\ &= (F_a^2 + F_b^2 + F_c^2 + F_d^2 + F_aF_c + F_aF_d + F_bF_c) \\ &\quad + (F_b^2 + F_d^2 + F_aF_c + F_bF_d)x \\ &= (F_a^2 + F_b^2 + F_c^2 + F_d^2 + (F_a + F_b) \cdot (F_c + F_d) + F_bF_d) \\ &\quad + (F_b^2 + F_d^2 + F_aF_c + F_bF_d)x \end{aligned}$$

Een methode om deze vier resultaten uit te rekenen is te zien in Algoritme 3.11. Er zijn vier registers nodig voor  $F$  en acht voor  $I_t$ ,  $J_t$ ,  $G_t$  en  $G_n$ . Tevens moet er nog één tijdelijk register voorzien worden voor  $a$ . De uitkomsten kunnen bepaald worden na twaalf optellingen, drie vermenigvuldigingen en vier kwadrateringen.

### Inversie in $\mathbb{F}_{2^{2m}}$

Vervolgens moet de inverse van  $G_n$  berekend worden, wat een inversie in  $\mathbb{F}_{2^{2m}}$  is [50].

Stel  $A = A_a + A_bx \in \mathbb{F}_{2^{2m}}$ ,  $A \neq 0$  met multiplicatieve inverse  $B = B_a + B_bx \in \mathbb{F}_{2^{2m}}$ . Volgens de definitie is  $A \cdot B = 1$ . Gegeven  $x^2 = x + 1$ , gelden dus de vergelijkingen:

$$\begin{cases} A_aB_a + A_bB_b = 1 \\ A_aB_b + A_bB_a + B_bA_a = 0. \end{cases}$$

De oplossing van dit stelsel is

$$\begin{cases} B_a &= w^{-1} \cdot (A_a + A_b), \\ B_b &= w^{-1} \cdot A_b, \end{cases}$$

---

**Algoritme 3.11:** Uitwerking van berekening van noemers voor de finale machtsverheffing in het Miller algoritme

---

**Input:**  $F = F_a + F_b x + F_c y + F_d xy \in \mathbb{F}_{2^{4m}}$

**Output:**  $I_t = I_{t_a} + I_{t_b} x, J_t = J_{t_a} + J_{t_b} x, G_t = G_{t_a} + G_{t_b} x,$   
 $G_n = G_{n_a} + G_{n_b} x \in \mathbb{F}_{2^{2m}}$

**Data:**  $a \in \mathbb{F}_{2^m}$

|   |  |          |
|---|--|----------|
| 1 | $I_{t_a} \leftarrow F_a^2; I_{t_b} \leftarrow F_b^2; G_{t_a} \leftarrow F_c^2; G_{t_b} \leftarrow F_d^2$ | 4 S      |
| 2 | $I_{t_a} \leftarrow I_{t_a} + I_{t_b}; J_{t_b} \leftarrow I_{t_b} + G_{t_b}$                             | 2 A      |
| 3 | $G_{t_b} \leftarrow I_{t_b} + G_{t_b}; J_{t_a} \leftarrow I_{t_a} + G_{t_b}$                             | 2 A      |
| 4 | $I_{t_a} \leftarrow I_{t_a} + G_{t_a}; I_{t_b} \leftarrow I_{t_b} + G_{t_b}$                             | 2 A      |
| 5 | $G_{n_a} \leftarrow F_a + F_b; G_{n_b} \leftarrow J_a \cdot J_c$   | 1 M, 1 A |
| 6 | $a \leftarrow F_c + F_d; G_{n_a} \leftarrow G_{n_a} \cdot a; a \leftarrow F_b \cdot F_d$                 | 2 M, 1 A |
| 7 | $G_{n_a} \leftarrow G_{n_a} + a; G_{n_b} \leftarrow G_{n_b} + a$   | 2 A      |
| 8 | $G_{n_a} \leftarrow G_{n_a} + J_{t_a}; G_{n_b} \leftarrow G_{n_b} + J_{t_b}$                             | 2 A      |

---

met  $w = A_a^2 + (A_a + A_b) \cdot A_b$ . Een inversie in  $\mathbb{F}_{2^{2m}}$  kan dus berekend worden via een inversie in  $\mathbb{F}_{2^m}$ . Uitgewerkt geeft dit Algoritme 3.12, waarbij de oorspronkelijke  $A$  wordt overschreven door zijn inverse. Het algoritme kost in totaal drie vermenigvuldigingen, één kwadratering, twee optellingen en één inversie in  $\mathbb{F}_{2^m}$ . Er zijn twee registers nodig om  $A$  in op te slaan en drie tijdelijke registers voor  $a$ ,  $b$  en  $c$ . Verder heeft het algoritme voor inversie in  $\mathbb{F}_{2^m}$  twee tijdelijke registers nodig, waarbij tijdelijk register  $b$  de taak van één van deze twee kan vervullen, aangezien dat niet meer gebruikt wordt na regel 2 in het algoritme. Dit alles samen komt dus neer op vier tijdelijke registers.

---

**Algoritme 3.12:** Uitwerking van  $A^{-1} \in \mathbb{F}_{2^{2m}}$

---

**Input:**  $A = A_a + A_b x \in \mathbb{F}_{2^{2m}}, A \neq 0$

**Output:**  $B = A^{-1} = B_a + B_b x \in \mathbb{F}_{2^{2m}}$

**Data:**  $a, b, c \in \mathbb{F}_{2^m}$

|   |  |          |
|---|--|----------|
| 1 | $a \leftarrow A_a + A_b; b \leftarrow A_a^2$           | 1 S, 1 A |
| 2 | $c \leftarrow a \cdot A_b; c \leftarrow c + b$         | 1 M, 1 A |
| 3 | $c \leftarrow c^{-1}$                                  | 1 S      |
| 4 | $B_a \leftarrow a \cdot c; B_b \leftarrow A_b \cdot c$ | 2 M      |

---

### Berekening van $I$ en $J$

Gewapend met al deze waarden is het mogelijk  $I = I_0 + I_1 y$  en  $J = J_0 + J_1 y$  te berekenen. In totaal zijn hier zes vermenigvuldigingen in  $\mathbb{F}_{2^{2m}}$  nodig.

Stel  $A = A_a + A_b x, B = B_a + B_b x \in \mathbb{F}_{2^{2m}}$ . De vermenigvuldiging kan dan geschreven worden als:

$$\begin{aligned}
C &= (A_a + A_b x) + (B_a + B_b x) \\
&= (A_a B_a + A_b B_b) + (A_a B_b + A_b B_a + A_b B_b) x \\
&= (A_a B_a + A_b B_b) + ((A_a + A_b) \cdot (B_a + B_b) + A_a B_a) x
\end{aligned}$$

In dit geval dienen zowel  $I_t$ ,  $J_t$  als  $G_t$  met  $G_n^{-1}$  vermenigvuldigd te worden. Vooropgesteld dat de tellers na vermenigvuldiging met  $G_n$  niet meer nodig zijn en dus overschreven mogen worden, kan Algoritme 3.13 gebruikt worden. Er zijn dan zes registers nodig voor de uitkomsten, twee voor  $G_n^{-1}$  en twee tijdelijke register voor  $a$  en  $b$ . De vermenigvuldiging in  $\mathbb{F}_{2^{2m}}$  kan berekend worden in drie vermenigvuldigingen en vier optellingen.

---

**Algoritme 3.13:** Uitwerking van  $A \cdot B \in \mathbb{F}_{2^{2m}}$

---

**Input:**  $A = A_a + A_b x, B = B_a + B_b x \in \mathbb{F}_{2^{2m}}$

**Output:**  $A = A \cdot B \in \mathbb{F}_{2^{2m}}$

**Data:**  $a, b \in \mathbb{F}_{2^m}$

|   |  |          |
|---|--|----------|
| 1 | $a \leftarrow A_b \cdot B_b; b \leftarrow A_a + A_b$     | 1 M, 1 A |
| 2 | $A_a \leftarrow A_a \cdot B_a; A_b \leftarrow B_a + B_b$ | 1 M, 1 A |
| 3 | $A_b \leftarrow A_b \cdot b$                             | 1 M      |
| 4 | $A_b \leftarrow A_a + A_b; A_a \leftarrow A_a + a$       | 2 A      |

---

### Berekening van $I^{2^m+1}$

Nu  $I$  en  $J$  bekend zijn, moeten ze beiden tot de correcte macht verheven worden. Voor  $I$  is dat  $2^m + 1$ , wat simpelweg mogelijk zou zijn door  $I^{2^m}$  te berekenen en dit resultaat te vermenigvuldigen met  $I$ . Dit zou een vermenigvuldiging van twee elementen in  $\mathbb{F}_{2^{4m}}$  zijn, wat, zoals verderop zal gezien worden, negen vermenigvuldigingen en tweeëntwintig optellingen kost. Er is echter een snellere manier.

Uitgaande van de gelijkheid  $A^{2^m} = A$  voor  $A \in \mathbb{F}_{2^m}$ , weet men dat de vier elementen van  $I$  na de machtsverheffing simpelweg zullen bestaan uit sommen van de beginwaarden. Via volgende gelijkheden:

$$\begin{aligned}
x^{2^m} &= 1 + x \\
y^{2^m} &= 1 + x + y + xy \\
(xy)^{2^m} &= x + xy
\end{aligned}$$

en met  $I = I_a + I_b x + I_c y + I_d xy$ , vindt men dus:

$$I^{2^m} = (I_a + I_b + I_c) + (I_b + I_c + I_d)x + I_c y + (I_c + I_d)xy.$$

Vervolgens vermenigvuldigt men dit met de originele  $I$ , wat resulteert in

$$I^{2^m+1} = r_a + r_b x + r_c y + r_d xy:$$

$$\begin{aligned} r_a &= (I_a + I_b + I_c) \cdot I_a + (I_b + I_c + I_d) \cdot I_b + I_c^2 + (I_c + I_d) \cdot I_d \\ r_b &= (I_a + I_b + I_c) \cdot I_b + (I_b + I_c + I_d) \cdot I_a + (I_b + I_c + I_d) \cdot I_b \\ &\quad + I_c I_d + (I_c + I_d) \cdot I_c + (I_c + I_d) \cdot I_d \\ r_c &= (I_a + I_b + I_c) \cdot I_c + (I_b + I_c + I_d) \cdot I - d + I_a I_c + I_c^2 + I_c I_d \\ &\quad + (I_c + I_d) \cdot I_b + (I_c + I_d) \cdot I_c \\ r_d &= (I_a + I_b + I_c) \cdot I_d + (I_b + I_c + I_d) \cdot I_c + (I_b + I_c + I_d) \cdot I_d \\ &\quad + I_b I_c + I_c^2 + (I_c + I_d) \cdot I_a + (I_c + I_d) \cdot I_b + (I_c + I_d) \cdot I_d. \end{aligned}$$

Dit kan vereenvoudigd worden tot:

$$\begin{aligned} r_a &= I_a^2 + I_a I_b + I_a I_c + I_b^2 + I_b I_c + I_b I_d + I_c^2 + I_c I_d + I_d^2 \\ r_b &= I_a I_c + I_a I_d + I_b I_d + I_c I_d + I_c^2 + I_d^2 \\ r_c &= I_c^2 + I_c I_d + I_d^2 \\ r_d &= I_a I_c + I_b I_c + I_b I_d. \end{aligned}$$

Voor de toepassing van de Karatsuba-Ofman techniek worden de volgende tussenresultaten uitgerekend:

$$\begin{aligned} m_0 &= (I_a + I_b) \cdot (I_c + I_d) & m_1 &= I_a I_b \\ m_2 &= I_a I_d & m_3 &= I_b I_c \\ m_4 &= I_c I_d \\ s_0 &= (I_a + I_b)^2 & s_1 &= (I_c + I_d)^2. \end{aligned}$$

Aan de hand van deze waarden kan het uiteindelijke resultaat berekend worden:

$$\begin{aligned} I^{2^m+1} &= (m_0 + m_1 + m_2 + m_4 + s_0 + s_1) \\ &\quad + (m_0 + m_3 + m_4 + s_1)x \\ &\quad + (m_4 + s_1)y + (m_0 + m_2)xy. \end{aligned}$$

Ook bij de implementatie van dit algoritme wordt ervan uitgegaan dat de beginvariable,  $I$  in dit geval, niet behouden dient te worden. Er zijn dan uiteindelijk vier registers nodig voor het resultaat en vijf voor tussenresultaten. De totale kost komt neer op vijf vermenigvuldigingen, twee kwadrateringen en negen optellingen. Algoritme 3.14 geeft een overzicht van de bewerkingen.

### Berekening van $J^{2^{\frac{m+1}{2}}}$

De voorlaatste berekening die gemaakt moet worden, is de machtsverheffing van  $J$ . Helaas is het in dit geval niet mogelijk om de bewerking even snel uit te voeren als de kwadratering van  $I$ . De handigste oplossing in dit geval is  $J$  simpelweg  $\frac{m+1}{2}$  opeenvolgende keren te kwadrateren. Hiervoor

---

|   |  |          |
|---|--|----------|
| <b>Algoritme 3.14:</b> Uitwerking van $I^{2^m+1} \in \mathbb{F}_{2^{4m}}$ |  |          |
| <b>Input:</b> $I = I_a + I_b x + I_c y + I_d xy \in \mathbb{F}_{2^{4m}}$  |  |          |
| <b>Output:</b> $I = I^{2^m+1} \in \mathbb{F}_{2^{4m}}$                    |  |          |
| <b>Data:</b> $a, b, \dots, e \in \mathbb{F}_{2^m}$                        |  |          |
| 1   | $a \leftarrow I_a \cdot I_b; b \leftarrow I_a + I_b$   | 1 M, 1 A |
| 2   | $c \leftarrow b^2; d \leftarrow I_a \cdot I_d$         | 1 M, 1 S |
| 3   | $I_b \leftarrow I_b \cdot I_c; e \leftarrow I_c + I_d$ | 1 M, 1 A |
| 4   | $b \leftarrow b \cdot e; e \leftarrow e^2$             | 1 M, 1 S |
| 5   | $I_c \leftarrow I_c \cdot I_d$                         | 1 M      |
| 6   | $I_c \leftarrow I_c + e; I_a \leftarrow I_c + b$       | 2 A      |
| 7   | $I_b \leftarrow I_a + I_b; I_d \leftarrow b + d$       | 2 A      |
| 8   | $b \leftarrow c + d; I_a \leftarrow I_a + b$           | 2 A      |
| 9   | $I_a \leftarrow I_a + a$                               | 1 A      |

---

kan Algoritme 3.9 gebruikt worden. In totaal zal deze stap  $2 \cdot (m + 1)$  vermenigvuldigingen en optellingen vragen, wat voor de gekozen  $m = 163$  neerkomt op 328 maal beide bewerkingen.

#### Vermenigvuldiging van $I \cdot J$

De finale stap is de vermenigvuldiging van de twee resulterende variabelen  $I$  en  $J$ . Dit is een vermenigvuldiging in  $\mathbb{F}_{2^{4m}}$  en deze kan als volgt geformuleerd worden:

$$\begin{aligned}
 \hat{e}(P, Q) &= I \cdot J \\
 &= (I_a J_a + I_b J_b + I_c J_c + I_d J_d) \\
 &\quad + (I_a J_b + I_b J_a + I_b J_b + I_c J_d + I_d J_c + I_d J_d)x \\
 &\quad + (I_a J_c + I_b J_d + I_c J_a + I_c J_c + I_d J_d + I_d J_b + I_d J_c)y \\
 &\quad + (I_a J_d + I_b J_c + I_b J_d + I_c J_b + I_c J_c + I_d J_a + I_d J_b + I_d J_d)xy
 \end{aligned}$$

Om de berekening via de Karatsuba-Ofman techniek te versnellen, worden eerst volgende tussenresultaten uitgerekend:

$$\begin{aligned}
 m_0 &= I_a J_a & m_1 &= I_b J_b \\
 m_3 &= I_c J_c & m_4 &= I_d J_d \\
 l_0 &= (I_a + I_b) \cdot (J_a + J_b) & l_1 &= (I_c + I_d) \cdot (J_c + J_d) \\
 l_2 &= (I_a + I_d) \cdot (J_a + J_d) & l_3 &= (I_a + I_c) \cdot (J_a + J_c) \\
 n_0 &= I_a + I_b + I_c + I_d & n_1 &= J_a + J_b + J_c + J_d \\
 p_0 &= n_0 \cdot n_1
 \end{aligned}$$

Gebruik makende van deze waarden, kan de vermenigvuldiging voor



$\hat{e}(P, Q)$  geschreven worden als:

$$\begin{aligned}\hat{e}(P, Q) = & (m_0 + m_1 + m_2 + m_3) + (m_0 + m_2 + l_0 + l_1)x \\ & + (m_0 + m_1 + m_2 + l_1 + l_2 + l_3)y \\ & + (m_0 + m_3 + l_0 + l_1 + l_3 + p_0)xy\end{aligned}$$

Indien zowel  $I$  als  $J$  overschreven mogen worden, is de minimum benodigde opslagruimte om deze berekeningen uit te voeren vier registers voor het eindresultaat  $I = \hat{e}(P, Q)$  en zes tijdelijke registers voor  $a, b, \dots, f$ . Dit alles samen vergt negen vermenigvuldigingen en tweeëntwintig optellingen, twee meer dan in [50]. De methode is terug te vinden in Algoritme 3.15.

---

**Algoritme 3.15:** Uitwerking van  $I \cdot J \in \mathbb{F}_{2^{4m}}$

---

**Input:**  $I = I_a + I_b x + I_c y + I_d xy, J = J_a + J_b x + J_c y + J_d xy \in \mathbb{F}_{2^{4m}}$

**Output:**  $I = I \cdot J \in \mathbb{F}_{2^{4m}}$

**Data:**  $a, b, \dots, f \in \mathbb{F}_{2^m}$

|    |  |          |
|----|--|----------|
| 1  | $a \leftarrow J_c + J_d; b \leftarrow J_b + J_d$         | 2 A      |
| 2  | $J_d \leftarrow I_d \cdot J_d; c \leftarrow I_b + I_d$   | 1 M, 1 A |
| 3  | $I_d \leftarrow I_c + I_d; d \leftarrow I_b \cdot J_b$   | 1 M, 1 A |
| 4  | $e \leftarrow I_a + I_b; f \leftarrow J_a + J_b$         | 2 A      |
| 5  | $I_b \leftarrow J_a + J_c; J_c \leftarrow I_c \cdot J_c$ | 1 M, 1 A |
| 6  | $J_a \leftarrow I_a \cdot J_a; I_a \leftarrow I_a + I_c$ | 1 M, 1 A |
| 7  | $I_c \leftarrow I_a \cdot I_b; J_b \leftarrow c \cdot b$ | 2 M      |
| 8  | $I_a \leftarrow e \cdot f; I_b \leftarrow I_d \cdot a$   | 2 M      |
| 9  | $I_d \leftarrow e + I_d; e \leftarrow J_a + I_b$         | 2 A      |
| 10 | $I_b \leftarrow f + a; I_d \leftarrow I_b \cdot I_d$     | 1 M, 1 A |
| 11 | $a \leftarrow e + I_a; I_b \leftarrow J_c + a$           | 2 A      |
| 12 | $I_a \leftarrow d + J_c; I_d \leftarrow I_d + J_d$       | 2 A      |
| 13 | $I_d \leftarrow I_d + a; I_d \leftarrow I_c + I_d$       | 2 A      |
| 14 | $I_c \leftarrow I_c + e; I_c \leftarrow I_c + J_b$       | 2 A      |
| 15 | $I_c \leftarrow I_c + I_a; I_a \leftarrow J_d + I_a$     | 2 A      |
| 16 | $I_a \leftarrow I_a + J_a$                               | 1 A      |

---

### 3.5.4 Geheugen

Nu alle nodige algoritmes en de daarbij horende geheugenvereisten gekend zijn, is het mogelijk om het geheugenblok van de schakeling te ontwerpen. Eerst zal het minimum aantal benodigde registers bepaald worden en vervolgens zal een ontwerp voor het geheugenblok voorgesteld worden.

#### Minimum aantal registers

De geheugenvereisten van de for-lus en de finale machtsverheffing kunnen los van elkaar bekeken worden. Eens de lus beëindigd is, is het immers niet

meer nodig  $P, \phi(Q), V$  en  $G$  bij te houden.

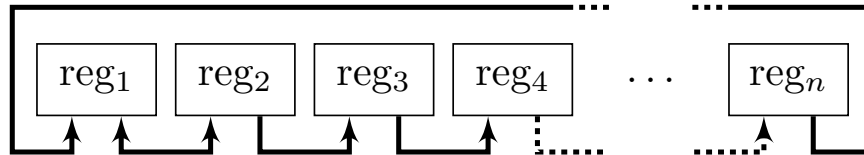
Van de vijf algoritmes die opgeroepen worden in de for-lus, vereist de vermenigvuldiging van  $F$  en  $G$  het grootste aantal tijdelijke registers, namelijk drie. Het minimum aantal registers om alle bewerkingen in de lus te kunnen uitvoeren is dus vijftien: zes voor  $P, \phi(Q)$  en  $V$ , twee voor  $G$ , vier voor  $F$  en drie tijdelijke registers.

Bij de berekening van de finale machtsverheffing gaat de vermenigvuldiging van  $I$  en  $J$  met de titel van ‘meest geheugenvereisende’ algoritme lopen. In dit geval zijn er veertien registers nodig: acht voor  $I$  en  $J$  en zes voor de tussenresultaten.

Het minimum aantal registers  $n$  is dus vijftien. In totaal bevat de complete schakeling zestien registers van grootte  $m$ : vijftien in de controller voor het Miller algoritme en één in de  $\mathbb{F}_{2^m}$  kern.

### Ontwerp van het geheugenblok

Voor het ontwerp van het geheugenblok wordt RAM al op voorhand verworpen gezien dit zowel groot is als veel vermogen verbruikt. Er wordt verder gebouwd op de ideeën aangebracht in [51]. Daarin wordt een circulair ontwerp voorgesteld dat geïllustreerd wordt in Figuur 3.7. Het voordeel van zo’n ontwerp ten opzichte van RAM is de veel kleinere oppervlakte. Bij RAM stijgt de complexiteit van de oppervlakte van de multiplexers kwadratisch met het aantal registers. Bij een ontwerp van dit type is de complexiteit van de multiplexers echter constant met als resultaat een veel kleinere implementatie.



Figuur 3.7: Circulair registerblokontwerp en mogelijke bewerkingen

Elk van de registers kan enkel de waarde van zijn voorganger aannemen (de zogenaamde *shift* bewerking). Tevens kunnen de waarden van de eerste twee registers omgewisseld worden (*swap*) en kan de waarde van register twee naar register één gekopieerd worden (*copy*). De ingangen van de wiskundige schakeling (de  $\mathbb{F}_{2^m}$  kern in dit geval) zijn rechtstreeks verbonden met registers één en twee. Aan register één zijn tevens ook de uitgang van de wiskundige schakeling en de ingang van de volledige schakeling aangesloten. De enige manier om nieuwe waarden in de registers op te slaan, is dus via de aanpassing van de waarde van register één.

Omdat de ingangen van de onderliggende  $\mathbb{F}_{2^m}$  kern rechtstreeks verbonden zijn met twee registers, is het noodzakelijk om voor elke bewerking de

nodige waarden naar die registers over te brengen. Het gemiddeld aantal klokcycli  $\bar{t}$  dat daar voor nodig is, kan op de manier die volgt, bepaald worden. Eerst wordt de gemiddelde afstand  $\bar{r}$  tussen twee variabelen  $x_0$  en  $x_1$  bepaald. Die is gelijk aan de som  $s$  van de mogelijke afstanden  $r$  gedeeld door het aantal mogelijke posities  $c$  die twee variabelen kunnen bezetten:

$$\begin{aligned} s &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n (j - i - 1) & c &= \sum_{i=0}^{n-1} i \\ &= \frac{n \cdot (n-1) \cdot (n-2)}{6} & &= n \cdot \frac{n-1}{2} \end{aligned}$$

dus

$$\bar{r} = \frac{n-2}{3}.$$

Merk op dat deze formules slechts een benadering geven. Zo wordt bijvoorbeeld niet in rekening gebracht dat wanneer beide waarden in de eerste twee registers zitten, er geen doorschuivingen moeten gebeuren.

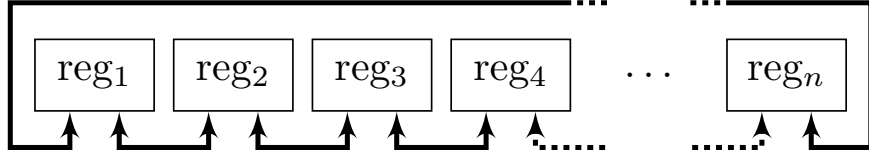
Per positie die  $x_0$  en  $x_1$  van elkaar verwijderd zijn, zal een *swap* bewerking uitgevoerd moeten worden. Daartoe dient  $x_1$  telkens eerst verplaatst te worden naar register twee. Er zijn dus  $\bar{r} \cdot n$  doorschuifbewerkingen nodig. Dit is opnieuw een ruwe schatting, die voor kleine  $n$  niet zal kloppen. Algemeen kan echter gesteld worden dat

$$\bar{t} \in O\left(\frac{n^2}{3}\right).$$

Verder is het energieverbruik recht evenredig met het aantal schrijfbewerkingen die op de registers worden uitgevoerd. Elke doorschuifbewerking kost  $n$  zulke bewerkingen, m.a.w. het gemiddeld aantal schrijfbewerkingen  $\bar{w}$  dat uitgevoerd moet worden voor gestart kan worden met de berekening van een optelling of vermenigvuldiging, is

$$\bar{w} \in O\left(\frac{n^3}{3}\right).$$

Gezien het feit dat er vijftien registers zijn (tegenover zes in [51]), zou dit zeer veel energie kosten. Verder zou het veel klokcycli in beslag nemen om beide waarden in de correcte registers op te slaan, hoewel zoals eerder bepaald tijdsduur niet van zeer groot belang is. Om dit probleem op te lossen, wordt het mogelijk gemaakt dat elk register ook de waarde van zijn voorganger kan opslaan. Dit is equivalent aan de *swap* (en *copy*) bewerking toelaten tussen alle aan elkaar grenzende registers. Verder kunnen *swap*-bewerkingen tussen twee paar registers in parallel uitgevoerd worden. Uiteraard wordt hiervoor een prijs betaald, er moet nu namelijk per register een extra multiplexer en een selectie signaal voorzien worden. Het resulterend ontwerp is te zien in Figuur 3.8.



Figuur 3.8: Circulair registerblok geoptimaliseerd voor laag energieverbruik

In dit geval zal de afstand  $r$  van een waarde  $x$  tot het eerste register gelijk zijn aan  $\min(j-1, n-j+1)$  omdat nu in beide richtingen geschoven kan worden. De gemiddelde afstand  $\bar{r}$  is dan

$$\begin{aligned}\bar{r} &= \frac{1}{n} \cdot \sum_{i=1}^n \min(j-1, n-j+1) \\ &= \frac{2}{n} \cdot \sum_{i=1}^{\frac{n}{2}} (j-1) \\ &= \frac{n-1}{4}.\end{aligned}$$

Aangezien de omwisselbewerkingen in parallel uitgevoerd kunnen worden, is het gemiddeld aantal klokcycli in dit geval

$$\bar{t} \in O\left(\frac{n}{4}\right).$$

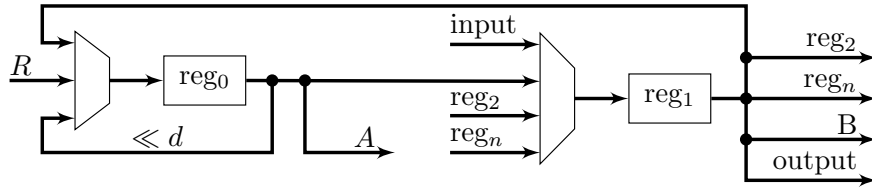
Rekening houdend met het feit dat elke omwisselbewerking twee schrijfbewerkingen vraagt en er twee variabelen naar de juiste positie gebracht moeten worden, geldt

$$\bar{w} \in O(n).$$

Tegenover het originele ontwerp voor het registerblok, zal dit ontwerp veel minder klokcycli verloren laten gaan aan het in de juiste positie brengen van de benodigde variabelen. Ook zal het, ten koste van meer multiplexers, minder energie verbruiken. De extra multiplexers zullen echter ook energie vragen, dus in hoeverre dit het totale energieverbruik naar beneden haalt, moet van implementatie tot implementatie bekeken worden.

Registers één en twee worden respectievelijk met de  $A$  en  $B$  ingang van de  $\mathbb{F}_{2^m}$  kern verbonden. Om vermenigvuldigingen tot een succesvol einde te brengen, moet het register dat met de  $A$  ingang verbonden is elke klokslag met  $d$  bits naar links doorgeschoven kunnen worden. Hierbij gaat de originele waarde van het register echter wel verloren. Wanneer de algoritmes die het meeste tijdelijke opslag vragen echter nauwkeurig worden bekeken, zal men merken dat dit geen probleem vormt. Vandaar dat er voor gekozen wordt om het derde tijdelijk register uit de doorschuiflus te halen, er moeten dan minder doorschuivingen doorgevoerd worden om elke waarde

in de juiste positie te brengen. Verder moet het ook mogelijk zijn waarden die op de ingang van de Miller controller aangelegd worden, op te slaan en het resultaat op de uitgang te zetten. Ten slotte moet het resultaat van een berekening in  $\mathbb{F}_{2^m}$  opgeslagen kunnen worden. Teneinde dat mogelijk te maken, wordt de uitgang  $R$  van de  $\mathbb{F}_{2^m}$  kern aan het derde tijdelijk register gekoppeld. Rekening houdend met deze zaken, wordt het ontwerp van de eerste twee registers aangepast zoals te zien in Figuur 3.9.



Figuur 3.9: Ontwerp van de schakeling rond de eerste twee registers in het registerblok

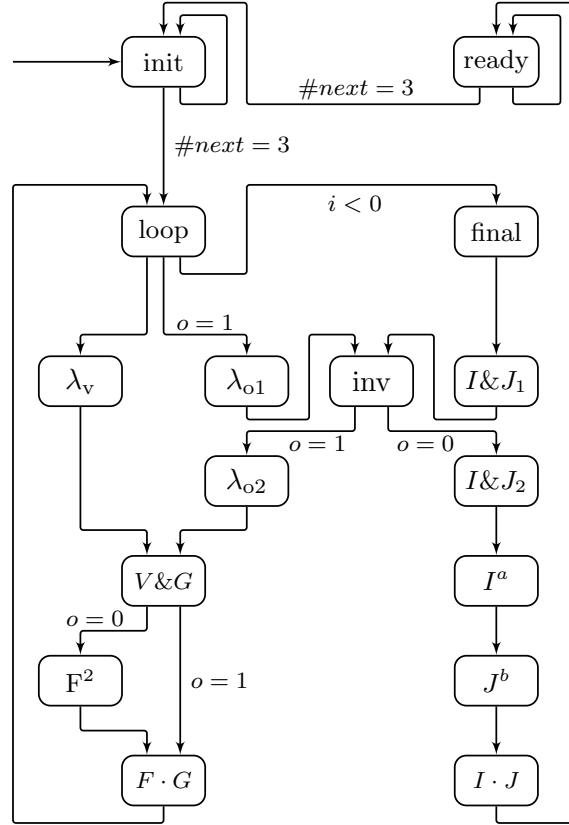
### 3.5.5 FSM

Nu alle details over de hardware bekend zijn, kan overgegaan worden tot het ontwerp van een FSM. Daarbij zal een zeer groot deel van de te implementeren toestanden voor niets anders dienen dan het verschuiven van geheugen. Het uiteindelijke resultaat bevat 553 toestanden. Om het geheel overzichtelijk te houden, zal het ontwerp dan ook niet tot op individueel state niveau gebeuren.

Aangezien een beeld meer zegt dan duizend woorden, wordt het resultaat zonder veel extra uitleg gepresenteerd in Figuur 3.10. De aanduidingen *start* en *next* staan voor de ingangen van de schakeling (zie Figuur 3.6). De aanduiding  $\#next = 3$  wil zeggen: ‘aantal maal dat *next* hoog werd is gelijk aan drie’. De *next* ingang wordt gebruikt om te signaleren dat een nieuwe ingangswaarde op de ingang van de schakeling staat of dat een volgend deel van het resultaat op de uitgang mag gezet worden. Een register *o* geeft aan of de optelstap moet uitgevoerd worden en er is een teller *i* die bijhoudt hoeveel stappen in de for-lus nog uitgevoerd moeten worden.

Wat wel enige extra aandacht verdient, is de implementatie van de verdubbel- en optelstap. Wanneer Algoritme 3.6 en 3.7 opnieuw bekeken worden, valt het op dat buiten de berekening van  $\lambda$  en de nieuwe  $x$ -coördinaat van  $V$  alle andere berekeningen zeer gelijkaardig zijn. Meer nog, indien de paren  $(x_V, y_V)$  en  $(x_P, y_P)$  vervangen worden door een algemene  $(x_A, y_A)$ , dan bestaan beide algoritmes uit dezelfde bewerkingen na het berekenen van  $\lambda$  en  $x_V$ . Aangezien na de berekening van  $\lambda$  nog steeds twee registers vrij zijn voor gebruik, kunnen deze aangewend worden om de toe-

passelijke  $x$  en  $y$  coördinaten in op te slaan. Op die manier kunnen de verdere berekeningen via dezelfde toestanden uitgewerkt worden, ongeacht of nu voordien de verdubbel- of de optelstep uitgevoerd werd.



Figuur 3.10: FSM ontwerp van de controller voor het Miller algoritme

### 3.6 Optimalisaties

Nu de schakeling volledig ontworpen is, kan overgegaan worden tot optimalisering. Daarbij wordt opnieuw in de eerste plaats getracht de oppervlakte kleiner te maken, maar er zal ook aandacht gegeven worden aan het beperken van het energieverbruik van de schakeling. De optimalisaties die in de volgende paragrafen voorgesteld worden, zullen allemaal iets te maken hebben met de registers. Omdat de enkele registers van grootte 1 of  $\log_2(m)$  bit (voor tellers) niet veel bijdragen aan de uiteindelijke oppervlakte of het energieverbruik, zullen de optimalisaties specifiek gericht zijn op het efficiënter maken van het geheugenblok in de controller voor de Millerlus.

Een register is doorgaans opgebouwd uit een aantal D-type master-slave flip-flops. In de gebruikte bibliotheek [42] is dit type flip-flops opgebouwd

uit twee latches. De eerste latch (master) is verbonden met de ingang  $D$  en laat de waarde daarvan door zolang  $CLK$  laag is. De tweede latch (slave) is verbonden met de uitgang  $Q$  en neemt de waarde van de eerste latch over eens  $CLK$  hoog is. Het effect is dus dat de uitgang  $Q$  op een stijgende klokflank de waarde van  $D$  overneemt. Er bestaan verschillende gespecialiseerde types: met resetingang, met enable ingang, met test ingang, ... In de paragrafen die volgen, wordt ervan uitgegaan dat alle registers uit dit type flip-flops zijn opgebouwd.

### 3.6.1 Registers zonder reset

Een makkelijke eerste aanpassing is het verwijderen van de resetingen van de registers. Zoals reeds gezien in Tabel 3.1 kost een D flip-flop zonder resetingang 0,5 gate minder per bit dan één met, een verkleining van 8,5%.

In het geval van  $m = 163$  kijkt men dan aan tegen een besparing van  $978 - 896,5 = 81,5$  gates per register. Merk echter wel op dat ten minste één register moet overblijven dat wel op 0 (en 1) ingesteld kan worden om  $F$  in te stellen aan het begin van het algoritme.

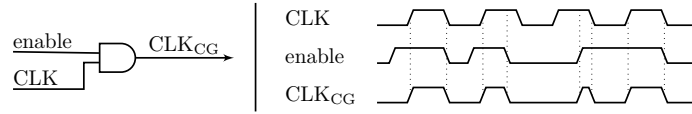
### 3.6.2 Clock gating

Normaal wordt een register elke klokslag geladen met de waarde aan zijn ingang. Het is dus noodzakelijk het register naar zichzelf terug te koppelen, zodat het mogelijk is een reeds opgeslagen waarde in het register te bewaren. Een techniek, genaamd clock gating, laat toe deze terugkoppeling (en de daarbij horende multiplexer) achterwege te laten [52]. Bij deze techniek wordt het kloksignaal enkel gepropageerd naar een register wanneer een daarbij horende enable ingang hoog is. Zolang het enable signaal laag gehouden wordt, blijft de waarde van het register dus constant.

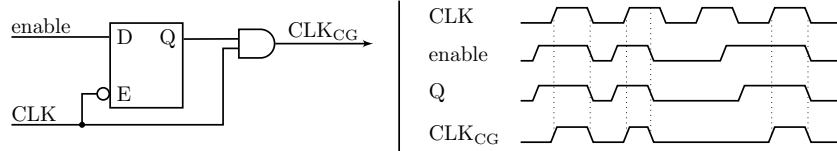
Deze techniek laat ook toe het verbruik van de schakeling drastisch te verlagen. Er zullen immers veel minder interne poorten zijn (bv. in de multiplexer) die elke klokslag van waarde dienen te veranderen.

De eenvoudigste schakeling waarmee clock gating geïmplementeerd kan worden, is te zien in Figuur 3.11. Er zijn echter enkele problemen geassocieerd met dit type schakeling. De uitgang is immers onderhevig aan onregelmatigheden op het enable signaal. Stel bijvoorbeeld dat de enable ingang al hoog wordt terwijl het kloksignaal ook nog hoog is. Dan zal het kloksignaal gepropageerd worden tot het register, wat niet de bedoeling is.

Dit probleem kan verholpen worden met de schakeling voorgesteld in Figuur 3.12. De latch zorgt er hier voor dat het enable signaal pas wordt doorgelaten nadat het kloksignaal laag geweest is. Hierdoor worden mogelijke onregelmatigheden dus tegengehouden voor de ze klokingang van het register kunnen bereiken.

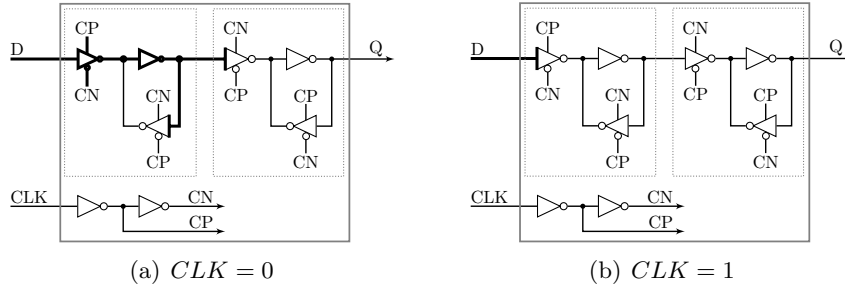


Figuur 3.11: Schakeling voor clock gating - Basis ontwerp



Figuur 3.12: Schakeling voor clock gating - Verbeterd ontwerp

Ten slotte is er een derde oplossing die toelaat een nog grotere energiebesparing door te voeren, zoals aangetoond in [53]. In die paper wordt bewezen dat het energieverbruik van een D-type master-slave flip-flop veel hoger is wanneer het kloksignaal laag is, dan wanneer het hoog is. De reden hiervan is duidelijk te zien in Figuur 3.13: wanneer de klok laag is, veranderen, telkens de ingang verandert, twee interne poorten van staat en wijzigen de gate-capaciteiten van twee andere poorten. Indien de klokingang hoog is, wijzigt een veranderend ingangssignaal enkel de gate-capaciteit van de eerste interne poort.

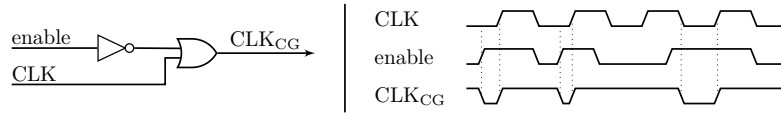


Figuur 3.13: Vermogenverbruikende onderdelen van een D-type master-slave flip-flop bij constante waarde van de klokingang [53]. De dikke lijnen geven de vermogenverbruikende onderdelen aan.

De werking van de twee vorige schakelingen is zo dat de klokingang laag gehouden wordt zolang het enable signaal laag is. De oplossing ligt in de schakeling in Figuur 3.14, die exact doet wat nodig is: de klokingang hoog houden zolang geen nieuwe waarde moet opgeslagen worden. Om het voorkomen van onregelmatigheden te garanderen, moet het enable signaal stabiel worden voor het CLK signaal hoog wordt. Indien bepaalde kloksig-



naalbeperkingen worden opgelegd bij het synthetiseren van het circuit is dit normaal gezien echter geen probleem.



Figuur 3.14: Schakeling voor clock gating - Laag vermogen ontwerp

## Hoofdstuk 4

# Resultaten

In dit hoofdstuk zullen verscheidene ASIC implementaties van de schakeling, die in Hoofdstuk 3 beschreven werd, van naderbij bestudeerd worden. Daarbij zal gelet worden op de oppervlakte en het vermogenverbruik van het resultaat. Er zal onderzocht worden wat het effect van de verschillende voorgestelde optimalisaties is op deze parameters. Ten slotte zal het ontwerp vergeleken worden met reeds bestaande implementaties.

### 4.1 Gebruikte software

Het ontwerp werd geprogrammeerd in GEZEL [54]. Simulaties en compilatie naar VHDL werden uitgevoerd met GEZEL 2.0. De optimalisaties werden manueel uitgevoerd in de VHDL code, aangezien dit in GEZEL niet mogelijk was.

Alle ontwerpen werden gesynthetiseerd met behulp van Synopsys Design Vision (versie Y-2006.06). De gebruikte bibliotheek was de  $0.13\mu m$  *low leakage* bibliotheek van Faraday Technology [42]. De maximale oppervlakte werd ingesteld op nul, wat als netto-effect een resultaat met minimum oppervlakte gaf. Verder werd voor het kloksignaal, indien niet anders vermeld, een frequentie van 10 kHz gebruikt.

### 4.2 Gerapporteerd vermogen

Voor de resultaten in verband met vermogenverbruik worden steeds twee waarden gegeven. De eerste waarde, dynamisch vermogen, geeft weer hoeveel vermogen verbruikt wordt door veranderende CMOS in- en uitgangen. De tweede waarde, lekvermogen, is vermogenverbruik dat voorkomt zelfs indien er geen signalen van waarde veranderen. De impact hiervan hangt onder meer af van de gebruikte bibliotheek. Het is voor het syntheseprogramma zeer moeilijk een nauwkeurige schatting voor het vermogenverbruik te geven, dus de gegeven waarden moeten niet als exact worden beschouwd.

Indien gewenst kan het vermogenverbruik voor hogere kloksnelheden geschat worden. Gegeven de standaard frequentie  $f = 10$  kHz, de formule voor het dynamisch vermogen van een CMOS schakeling:

$$P_d = V^2 \cdot C \cdot f$$

en het lekvermogen  $P_l$ , kan men het totale vermogen omrekenen naar dat van een willekeurige kloksnelheid via

$$P' = \frac{P_d \cdot f'}{10 \cdot 10^3} + P_l.$$

Het verkregen resultaat zal niet volledig kloppen, maar komt zeer dicht in de buurt van het door Design Vision geschatte verbruik.

### 4.3 Benodigde rekentijd

Alvorens de synthesesresultaten te presenteren, wordt even uitgeweid over het noodzakelijk aantal klokcycli  $c$  om één pairing te berekenen. Merk op dat de formules die volgen enkel kloppen voor willekeurige  $m$  indien  $\text{Ham}(l) = 3$ , wat het geval is voor de gekozen parameters (zie Paragraaf 3.1).

Het totaal aantal cycli voor een schakeling met veldgrootte  $m$  en aantal MALU's  $d$  wordt gegeven door:

$$c = 21681 + 4322 + 2998 \cdot \left\lceil \frac{m}{d} \right\rceil.$$

Daarbij staat het eerste getal voor het aantal klokcycli waarin gegevens van plaats verschoven worden in het intern geheugen. Het tweede getal is het aantal optellingen dat uitgevoerd dient te worden en de coëfficiënt voor de vermenigvuldiging is het aantal vermenigvuldigingen. Het aantal klokcycli  $c$  kan verder opgesplitst worden in het aantal cycli  $c_f$  voor de for-lus en  $c_m$  voor de finale machtsverheffing. In dat geval zijn de formules:

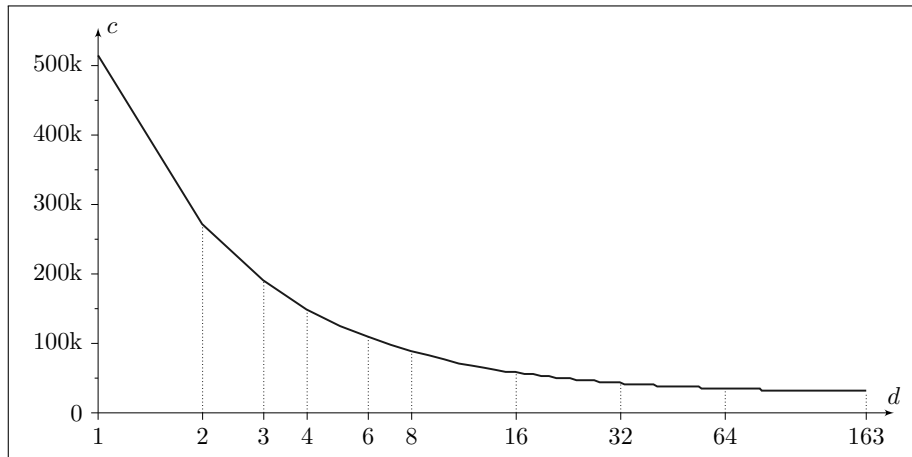
$$\begin{aligned} c_f &= 18731 + 3937 + 2463 \cdot \left\lceil \frac{m}{d} \right\rceil \\ c_m &= 2950 + 385 + 535 \cdot \left\lceil \frac{m}{d} \right\rceil, \end{aligned}$$

waarbij de coëfficiënten opnieuw voor respectievelijk geheugenverschuivingen, optellingen en vermenigvuldigingen staan.

In Tabel 4.1 wordt voor enkele waarden getoond hoeveel klokcycli nodig zijn om een berekening te voltooien in het geval  $m = 163$ . In Figuur 4.1 wordt hetzelfde weergegeven, maar dan voor elke  $d$  van 1 t.e.m. 163. Het is duidelijk dat de tijdsbesparing dankzij extra MALU's vrij snel teniet wordt gedaan door het aantal cycli dat niet door  $d$  beïnvloed wordt.

Tabel 4.1: Aantal klokcycli  $c$  nodig voor één pairing i.f.v. het aantal MALU's  $d$  voor veldgrootte  $m = 163$

|     |         |         |         |         |         |        |        |        |
|-----|---------|---------|---------|---------|---------|--------|--------|--------|
| $d$ | 1       | 2       | 3       | 4       | 6       | 8      | 16     | 32     |
| $c$ | 514 677 | 271 839 | 190 893 | 148 921 | 109 947 | 88 961 | 58 981 | 43 991 |



Figuur 4.1: Aantal klokcycli  $c$  nodig voor één pairing i.f.v. het aantal MALU's  $d$  voor veldgrootte  $m = 163$

## 4.4 Basisimplementatie & optimalisaties

Logischerwijs wordt eerst gekeken naar de syntheseresultaten voor een implementatie met één MALU. Verder zal in deze paragraaf ook onderzocht worden wat de effecten zijn van de optimalisaties voorgesteld in Paragraaf 3.6. Bij de implementaties met clock gating werden steeds ook de resetingen van zoveel mogelijk registers verwijderd. Ten slotte zal onderzocht worden hoeveel oppervlakte de individuele onderdelen van het ontwerp innemen.

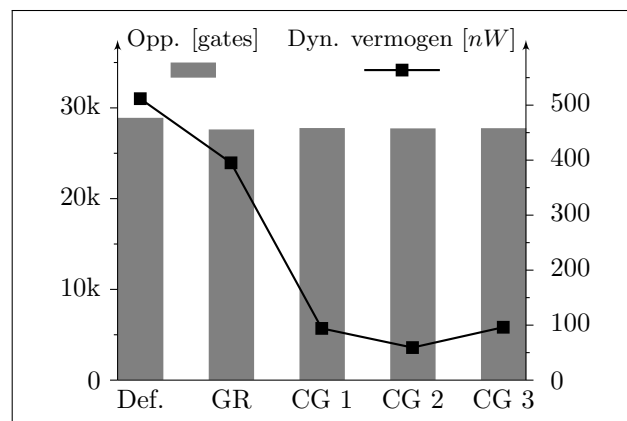
De syntheseresultaten voor de vijf verschillende implementaties worden gegeven in Tabel 4.2. De versies met clock gating (CG  $n$ ) implementeren de schakelingen in de volgorde waarin ze voorkomen in Paragraaf 3.6.2. Ter verduidelijking zijn deze resultaten ook nog eens uitgezet in Figuur 4.2.

Zoals verwacht resulteert het toepassen van clock gating in een zeer grote besparing op het vermogen. Het is zelfs zo dat na toepassing van clock gating het dynamisch vermogen lager is dan het lekvermogen. Daarbij dient opgemerkt te worden dat het dynamisch vermogen zal stijgen bij een hogere kloksnelheid terwijl het lekvermogen constant zal blijven.

Het is opmerkelijk dat het dynamisch vermogen van CG 2 zo veel lager wordt gerapporteerd dan dat van de andere twee CG implementaties. Hoe dit komt, is moeilijk te zeggen. In werkelijkheid zal implementatie CG 3 het

Tabel 4.2: Syntheseresultaten voor implementaties met één MALU. CG staat voor clock gating, de nummering komt overeen met de volgorde van de clock gating schakelingen in Paragraaf 3.6.2.

| Ontwerp    | Opp. [gates] |     | Vermogen @ 10 kHz [nW] |     |     |     |
|------------|--------------|-----|------------------------|-----|-----|-----|
|            |              |     | Dynamisch              |     | Lek |     |
| Basis      | 28 876       |     | 512                    |     | 117 |     |
| Geen Reset | 27 596       | 96% | 395                    | 77% | 107 | 92% |
| CG 1       | 27 751       | 96% | 94                     | 18% | 109 | 94% |
| CG 2       | 27 713       | 96% | 59                     | 12% | 102 | 88% |
| CG 3       | 27 734       | 96% | 96                     | 19% | 110 | 94% |



Figuur 4.2: Syntheseresultaten voor implementaties met één MALU en  $f = 10$  kHz. GR staat voor geen reset, CG staat voor clock gating, de nummering komt overeen met de volgorde van de clock gating schakelingen in Paragraaf 3.6.2.

minste verbruiken.

Verder valt het ook op dat de implementaties met clock gating niet kleiner zijn dan die zonder. Normaal gezien zou de toepassing van deze schakeling per register één multiplexeringang moeten elimineren. Waarom de synthesoftware dit niet doet, is onduidelijk. Er werd getracht de synthesoftware zelf clock gating te laten implementeren, maar de resulterende implementatie was noch kleiner, noch minder verbruikend dan het niet-geoptimaliseerde ontwerp.

Ten slotte wordt nog ontleed hoeveel oppervlakte de individuele onderdelen van de schakeling innemen. Tabel 4.3 geeft een overzicht van de grootte van de verschillende delen van implementatie CG 3. Het valt op dat de zestien registers en de FSM, die bestaat uit 553 toestanden, samen goed zijn

voor 95,5% van de oppervlakte.

Tabel 4.3: Oppervlakte van de individuele onderdelen in de implementatie met één MALU

| Onderdeel               | Opp. [gates] |      |
|-------------------------|--------------|------|
| MALU                    | 458          | 1,7% |
| $\mathbb{F}_{2^m}$ kern |              |      |
| Logica                  | 783          | 2,8% |
| Registers               | 962          | 3,5% |
| Controller              |              |      |
| Logica                  | 13 044       | 47%  |
| Registers               | 12 487       | 45%  |
| Totaal                  | 27 734       | 100% |

## 4.5 Meerdere MALU's

Mits de toevoeging van extra MALU's is het, zoals aangetoond, mogelijk de totale rekeningtijd drastisch te verlagen. Hoewel het gebruik van meerdere MALU's de uiteindelijke schakeling vergroot en dat dus enigszins in gaat tegen de originele doelstelling, wordt hier toch onderzocht in welke mate de interessante parameters hierdoor juist worden beïnvloed.

Op de implementaties met meerdere MALU's werd steeds de derde clock gating techniek (en het verwijderen van resetingangen) toegepast, aangezien dit in werkelijkheid de grootste energiebesparing teweeg zou moeten brengen. Implementaties met een aantal MALU's gaande van twee t.e.m. tweeëndertig werden gesynthetiseerd. Een nog hoger aantal MALU's zou immers compleet ingaan tegen de originele doelstelling.

De resultaten van de synthese zijn te zien in Tabel 4.4 en Figuur 4.3. Het toevoegen van een klein aantal extra MALU's brengt weinig extra oppervlakte en vermogenverbruik met zich mee. Het is dus een aangeraden manier om de snelheid van de schakeling op te drijven. Dit wordt nog verder onderzocht in de volgende paragraaf. Het gerapporteerde dynamisch verbruik van de implementatie met twee MALU's is lager dan dat van de implementatie met één MALU, dit is opnieuw een fout van de software.

## 4.6 Hogere kloksnelheid vs. meerdere MALU's

Aan een kloksnelheid  $f = 10$  kHz doet een schakeling met één MALU er 51.5 seconden over om één pairing te berekenen. Dit zal in de meeste gevallen onaanvaardbaar zijn. Daarom wordt hier onderzocht wat de effecten op

Tabel 4.4: Syntheseresultaten voor implementaties met  $d$  MALU's

| $d$ | Opp. [gates] |      | Vermogen @ 10 kHz [ $nW$ ] |      |     |      | Tijds-<br>winst |
|-----|--------------|------|----------------------------|------|-----|------|-----------------|
|     |              |      | Dynamisch                  |      | Lek |      |                 |
| 1   | 27 734       |      | 96                         |      | 110 |      |                 |
| 2   | 28 423       | 102% | 90                         | 94%  | 113 | 103% | 47,2%           |
| 3   | 29 071       | 105% | 103                        | 107% | 118 | 107% | 62,9%           |
| 4   | 30 278       | 109% | 108                        | 113% | 122 | 111% | 71,1%           |
| 6   | 30 956       | 112% | 112                        | 117% | 127 | 115% | 78,6%           |
| 8   | 32 782       | 118% | 122                        | 127% | 136 | 124% | 82,7%           |
| 16  | 37 798       | 136% | 162                        | 169% | 163 | 148% | 88,5%           |
| 32  | 47 833       | 172% | 212                        | 221% | 213 | 194% | 91,5%           |

de schakeling zijn indien de kloksnelheid wordt opgedreven en er eventueel meerdere MALU's gebruikt worden. Ter illustratie zal voor een implementatie met meerdere MALU's die met twee nader onderzocht worden. Opnieuw wordt in beide gevallen clock gating schakeling drie gebruikt en worden zoveel mogelijk resetingen verwijderd.

Stel een maximale rekentijd  $t_{\max} \approx 50 \text{ ms}$ . Voor een implementatie met één MALU moet de klokfrequentie  $f_1$  dan 1030 maal verhoogd worden. Wanneer men de schakeling met twee MALU's even snel wenst te maken als die met één, dan dient de kloksnelheid  $f_2$  van die eerste vermenigvuldigd te worden met  $\Delta f = 1 - 0,472$ . De kloksnelheden van de respectievelijke schakelingen zijn dan:

$$f_1 \approx 10,3 \text{ MHz}$$

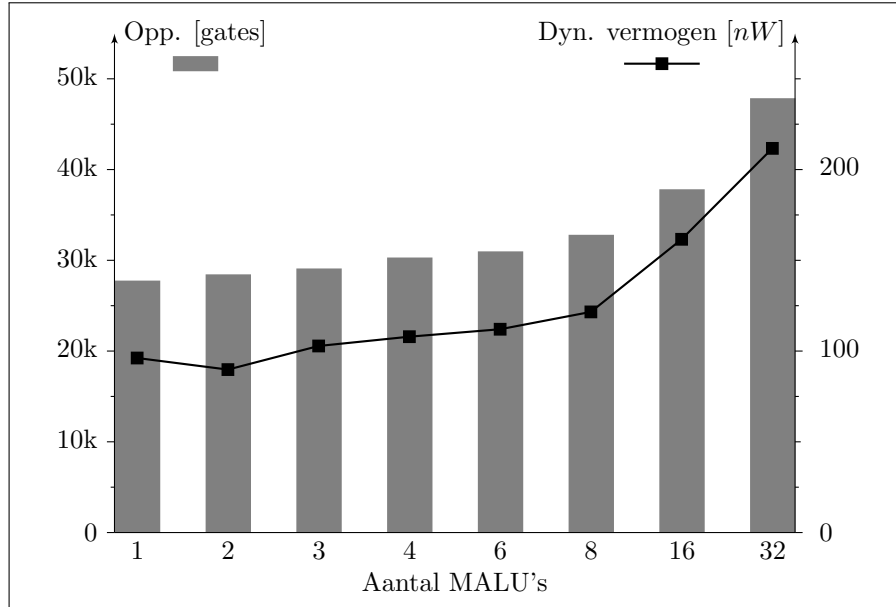
$$f_2 \approx 5,44 \text{ MHz}.$$

Ter vergelijking worden de resulterende parameters van beide implementaties gegeven in Tabel 4.5. Beiden werden gehersynthetiseerd met aangepaste parameters voor de klok. Dat de schakeling met één MALU in dit geval kleiner is dan in het geval  $f = 10 \text{ kHz}$ , is te wijten aan enige willekeurigheid in de algoritmes van de synthesesoftware.

Hoewel de implementatie met twee MALU's 3% groter is dan die met één, verbruikt ze slechts half zoveel vermogen. Het lijkt dus vanzelfsprekend om steeds voor een implementatie met meerdere MALU's te kiezen indien de snelheid moet opgedreven worden. Hoeveel MALU's ideaal zijn, zal afhankelijk zijn van de toepassing.

## 4.7 Vergelijking met bestaande implementaties

Gezien de vrij recente ontdekking van pairings is het beschikbare aantal implementaties voor ASIC's ook vrij beperkt. Er werden slechts drie papers



Figuur 4.3: Syntheseresultaten voor implementaties met meerdere MALU's en  $f = 10$  kHz

in de literatuur gevonden waarin het voorgestelde ontwerp naar een ASIC gesynthetiseerd werd. Op andere implementaties werd reeds ingegaan in Paragraaf 1.3.

De drie ASIC ontwerpen worden voorgesteld in respectievelijk [55], [56] en [57]. Zowel in [55] als [57] wordt met het oog op het behalen van zo hoog mogelijke snelheden ontworpen. De implementatie uit [56] bevat naast de schakeling voor pairings tevens een RISC processor. In [57] wordt de finale machtsverheffing niet uitgevoerd. Tabel 4.6 geeft een vergelijkend overzicht van het ontwerp voorgesteld in deze thesis en de vermelde ontwerpen. Aangezien het ontwerp uit [55] het enige is waarvan alle gegevens bekend zijn, zal in de tekst die volgt enkel met dat ontwerp vergeleken worden.

Alvorens te vergelijken moet er uiteraard wel op gewezen worden dat het ontwerp uit [55] zoals reeds vermeld niet ontworpen is met compactheid of laag vermogenverbruik in gedachten. Het is dus zeker niet de bedoeling het ontwerp af te breken.

De hier voorgestelde implementaties zijn niet alleen ongeveer een factor zeven kleiner dan degene voorgesteld in [55], hun vermogenverbruik is ongelofelijk veel lager. Het energieverbruik per bit veiligheid, ofwel de energie-efficiëntie, kan berekend worden als:

$$EE = \frac{\text{vermogen} \times \text{rekentijd}}{\text{veiligheid}}.$$

Opgegeven waarden voor veiligheid zijn zeer veranderlijk van paper tot pa-



Tabel 4.5: Vergelijking van synthesesresultaten voor twee verschillende implementaties met dezelfde uitvoeringstijd voor één pairing

|                      | 1 MALU              | 2 MALU's            |      |
|----------------------|---------------------|---------------------|------|
| $f$ [MHz]            | 10,3                | 5,44                | 53%  |
| Opp. [gates]         | 27 430              | 28 155              | 103% |
| Vermogen [ $\mu W$ ] |                     |                     |      |
| Dynamisch            | 98,2                | 48,5                | 49%  |
| Lek                  | $107 \cdot 10^{-3}$ | $111 \cdot 10^{-3}$ | 104% |

Tabel 4.6: Vergelijking van de implementatie voorgesteld in deze thesis met ASIC implementaties uit de literatuur

|                       | Thesis (CG 3) <sup>†</sup> |                        | Pairing-Lite [55]     | Kammler <i>et al.</i> [56] | Kömürçü en Savas [57] |
|-----------------------|----------------------------|------------------------|-----------------------|----------------------------|-----------------------|
|                       | 1 MALU                     | 2 MALUs                |                       |                            |                       |
| Veld                  | $\mathbb{F}_{2^{163}}$     | $\mathbb{F}_{2^{163}}$ | $\mathbb{F}_{3^{97}}$ | $\mathbb{F}_p$ 256 bit     | $\mathbb{F}_{3^{97}}$ |
| Pairing               | Tate                       | Tate                   | $\eta_T$              | Ate                        | Tate                  |
| Technologie           | 0,13 $\mu m$               | 0,13 $\mu m$           | 0,18 $\mu m$          | 0,13 $\mu m$               | 0,25 $\mu m$          |
| Opp. [gates]          | 27 734                     | 28 155                 | 193 765               | 97 000                     | $10mm^{2\dagger}$     |
| $f$ [MHz]             | 0,01                       | 5,44                   | 200                   | 338                        | 78                    |
| Rekentijd [ $\mu s$ ] | $51,5 \cdot 10^6$          | $50 \cdot 10^3$        | 46,7                  | $15,8 \cdot 10^3$          | 250 <sup>§</sup>      |
| Vermogen [ $mW$ ]     | $206 \cdot 10^{-6}$        | $48,6 \cdot 10^{-3}$   | 672                   | <i>onbekend</i>            | <i>onbekend</i>       |

<sup>†</sup> Implementatie zonder resets en met toepassing van clock gating schakeling drie.

<sup>‡</sup> De gegeven oppervlakte is die van de complete schakeling inclusief routing.

<sup>§</sup> Exclusief de finale machtsverheffing.

per. In dit geval zullen waarden gebruikt worden die door de auteurs van [55] worden opgegeven in [50]. Daar wordt gesteld dat het ontwerp uit deze thesis een veiligheid heeft van 652 bit en dat van [55] 922 bit.

De resultaten van de berekening zijn te zien in Tabel 4.7. Er dient op gewezen te worden dat in het ontwerp uit [55] met de  $\eta_T$  pairing gewerkt wordt, die ongeveer een factor twee sneller berekend kan worden. Het is duidelijk dat het ontwerp uit deze thesis het potentieel heeft veel efficiënter te zijn. Het gebruik van meer MALU's doet de efficiëntie duidelijk stijgen. Dit is te verklaren door het feit dat meer MALU's relatief gezien zeer weinig extra vermogen verbruiken, maar de rekentijd wel gevoelig verlagen. Indien een iets grotere oppervlakte dus geen bezwaar is, kan de efficiëntie van de schakeling nog flink verhoogd worden door meer MALU's toe te voegen. Tevens kan, indien een lange rekentijd geen punt is, tegelijkertijd de frequentie verlaagd worden, wat de efficiëntie nog meer ten goede zal komen.

Het is duidelijk dat de ontwerpen voorgesteld in deze thesis met kop en

Tabel 4.7: Vergelijking van energie-efficiëntie van de implementatie voorgesteld in deze thesis met ASIC implementaties uit de literatuur

|  | Thesis (CG 3) <sup>†</sup>     |                                  | Pairing-Lite [55] |
|--|--------------------------------|----------------------------------|-------------------|
|  | $d = 1,$<br>$f = 10\text{kHz}$ | $d = 2,$<br>$f = 5.44\text{MHz}$ |                   |
| Vermogen [ $mW$ ]                                    | $206 \cdot 10^{-6}$            | $48,6 \cdot 10^{-3}$             | 672               |
| Rekentijd [ $\mu s$ ]                                | $51,5 \cdot 10^6$              | $50 \cdot 10^3$                  | 46,7              |
| Veiligheid [bit]                                     | 652                            | 652                              | 922               |
| Efficiëntie [ $\frac{nJ}{\text{bit}}$ ] <sup>‡</sup> | 16,3                           | 3,73                             | 34,0              |

<sup>†</sup> Implementatie zonder resets en met toepassing van clock gating schakeling drie.

<sup>‡</sup> Des te lager, des te conservatiever springt de implementatie om met energie.

schouders uitsteken boven andere gepubliceerde ontwerpen indien compactheid en laag vermogen- en/of energieverbruik prioriteit genieten.

## Hoofdstuk 5

# Algemeen besluit

### 5.1 Besluit

Deze thesis handelde over pairings, een recente ontwikkeling op gebied van cryptografie. Pairings laten identiteits-gebaseerde cryptografie toe, wat zeer interessant is voor bijvoorbeeld netwerken van sensoren.

Er werd onderzocht hoe de Tate pairing geïmplementeerd kan worden in hardware. Meer specifiek werd de nadruk gelegd op een compacte implementatie die daarenboven nog eens zo weinig mogelijk vermogen verbruikte. Een implementatie van dat type zou toegepast kunnen worden in de eerder vermelde netwerken van sensoren of toestellen met een beperkt vermogen.

Verscheidene bestaande algoritmen werden aangepast en geoptimaliseerd zodat ze met een minimum gebruik aan geheugen uitgevoerd konden worden. Er werd een geheugenontwerp voorgesteld dat een goed compromis gaf tussen grootte en vermogenverbruik. Tevens werden verschillende oppervlakte- en vermogensbesparende technieken toegepast om het uiteindelijke circuit zo goed mogelijk aan de doelstellingen te laten voldoen. Door aanpassing van de kloksnelheid en het aantal gebruikte rekenschakelingen (MALU's) kunnen de parameters van het ontwerp aangepast worden aan de individuele vereisten van een toepassing.

Het resulterende ontwerp is uniek in zijn soort. Ten eerste is de ASIC implementatie met zijn <30k gates zeer klein. Verder is het mogelijk een gemiddeld vermogenverbruik te behalen van enkele tientallen nanowatt, indien rekensnelheid geen punt is. Ook aan hogere snelheden blijft het vermogenverbruik zeer laag vergeleken met het enige andere gedocumenteerde ontwerp uit de literatuur. Tevens is de energie-efficiëntie van het ontwerp zeer hoog. Ten tijde van dit schrijven waren nog geen andere ontwerpen gepubliceerd waarbij de nadruk op compactheid en laag vermogenverbruik lag.

## 5.2 Toekomstig onderzoek

Als toekomstig onderzoek kan het interessant zijn na te gaan of er nog significante optimalisaties aan het ontwerp mogelijk zijn. Daarbij zou onderzocht moeten worden of de FSM verder te vereenvoudigen valt, aangezien die zowat de helft van de oppervlakte van de totale implementatie in beslag neemt.

Het zou ook nuttig zijn te onderzoeken of het geheugenontwerp nog verder geoptimaliseerd kan worden. Een optimale plaatsing van variabelen in het geheugen zou helpen om het verbruik en de rektijd nog verder te verlagen. Er dient ook nagegaan te worden waarom bij de synthese de oppervlakte van de schakeling niet kleiner wordt na het toepassen van clock gating technieken en hoe dit opgelost kan worden.

Verder zou ook onderzocht moeten worden in welke mate werken over een groter Galoisveld de oppervlakte en het vermogenverbruik wijzigt. Mogelijk kunnen ook implementatie van andere types pairings, zoals de  $\eta_T$  en de Ate pairing, nader bestudeerd worden.

# Bibliografie

- [1] C. D. Isbell, “Some Cryptograms in the Aramaic Incantation Bowls,” *Journal of Near Eastern Studies*, vol. 33, no. 4, pp. 405–407, 1974.
- [2] C. A. Deavours, D. Kahn, L. Kruh, G. Mellen, and B. Winkel, Eds., *Cryptology: yesterday, today, and tomorrow*. Norwood, MA, USA: Artech House, Inc., 1987.
- [3] D. Kahn, *The Codebreakers: The Story of Secret Writing*. New York: The Macmillan Company, 1967.
- [4] W. C. Barker, *Recommendation for the Triple Data Encryption Algorithm (TDEA) block cipher*. National Institute for Standards and Technology, 2004 .
- [5] J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard*. Springer-Verlag, 2002.
- [6] W. Diffie and M. E. Hellman, “New Directions in Cryptography,” *Transaction on Information Theory*, vol. IT-22, no. 6, pp. 644–654, 1976.
- [7] D. R. L. Brown, “Generic Groups, Collision Resistance, and ECDSA,” *Des. Codes Cryptography*, vol. 35, no. 1, pp. 119–152, 2005.
- [8] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [9] P. Zimmermann, *PGP source code and internals*. Cambridge, MA, USA: MIT Press, 1995.
- [10] A. Shamir, “Identity-based cryptosystems and signature schemes,” in *Proceedings of CRYPTO 84 on Advances in cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 47–53.
- [11] M. Maas, “Pairing-Based Cryptography,” Master’s thesis, Technische Universiteit Eindhoven, januari 2004.

- [12] D. Boneh and M. Franklin, "Identity-Based Encryption from the Weil Pairing," *SIAM J. Comput.*, vol. 32, no. 3, pp. 586–615, 2003.
- [13] V. S. Miller, "The Weil Pairing, and Its Efficient Calculation," *Journal of Cryptology*, vol. 17, no. 4, pp. 235–261, 2004.
- [14] G. Frey, M. Muller, and H. Rück, "The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystem," *IEEE Transactions on Information Theory*, vol. 45, no. 5, pp. 1717–1719, 1999.
- [15] P. S. L. M. Barreto, S. D. Galbraith, C. ÓhÉigeartaigh, and M. Scott, "Efficient pairing computation on supersingular Abelian varieties," *Des. Codes Cryptography*, vol. 42, no. 3, pp. 239–271, 2007.
- [16] J. L. Hill and D. E. Culler, "Mica: a wireless platform for deeply embedded networks," *Micro, IEEE*, vol. 22, no. 6, pp. 12–24, 2002.
- [17] L. B. Oliveira, D. F. Aranha, E. Morais, F. Daguno, J. López, and R. Dahab, "TinyTate: Computing the Tate Pairing in Resource-Constrained Sensor Nodes," in *Sixth IEEE International Symposium on Network Computing and Applications*. IEEE Computer Society, 2007, pp. 318–323.
- [18] L. B. Oliveira, M. Scott, J. López, and R. Dahab, "TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks," in *5th International Conference on Networked Sensing Systems*, 2008, pp. 173–180.
- [19] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab, "NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks." *European conference on Wireless Sensor Networks (EWSN08)*, 2008.
- [20] Atmel, *ATMega128 datasheet*. [Online]. Available: <http://www.atmel.com>
- [21] R. Ronan, C. ÓhÉigeartaigh, C. C. Murphy, M. Scott, and T. Kerins, "Hardware acceleration of the Tate pairing on a genus 2 hyperelliptic curve," *Journal of Systems Architecture*, vol. 53, no. 2-3, pp. 85–98, 2007.
- [22] C. S. Soonhak, C. Shu, S. Kwon, and K. Gaj, "FPGA Accelerated Tate Pairing Based Cryptosystems over Binary Fields," in *Cryptology ePrint Archive, Report 2006/179*, 2006.
- [23] M. Keller, T. Kerins, F. M. Crowe, and W. P. Marnane, "FPGA Implementation of a  $GF(2^m)$  Tate Pairing Architecture." in *ARC*, ser. Lecture Notes in Computer Science, K. Bertels, J. a. M. P. Cardoso, and S. Vassiliadis, Eds., vol. 3985. Springer, 2006, pp. 358–369.

- [24] P. Grabher and D. Page, “Hardware Acceleration of the Tate Pairing in Characteristic Three.” in *CHES*, ser. Lecture Notes in Computer Science, vol. 3659. Springer, 2005, pp. 398–411.
- [25] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, and T. Takagi, “Algorithms and Arithmetic Operators for Computing the  $\eta_T$  Pairing in Characteristic Three.” *IEEE Trans. Computers*, vol. 57, no. 11, pp. 1454–1468, 2008.
- [26] F. Hess, N. P. Smart, and F. Vercauteren, “The Eta Pairing Revisited,” *IEEE Transactions on Information Theory*, vol. 52, pp. 4595–4602, 2006.
- [27] G. Frey and H.-G. Rück, “A remark concerning  $m$ -divisibility and the discrete logarithm in the divisor class group of curves,” *Mathematics of Computation*, vol. 62, no. 206, pp. 865–874, 1994.
- [28] F. Hess, “A note on the Tate pairing of curves over finite fields,” *Arch. Math.*, vol. 82, pp. 28–32, August 2004.
- [29] V. S. Miller, “Short Programs for Functions on Curves,” 1986.
- [30] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott, “Efficient Algorithms for Pairing-Based Cryptosystems,” in *CRYPTO 02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*. Springer-Verlag, 2002, pp. 354–368.
- [31] D. Shanks, “Class number, a theory of factorization, and genera,” *Proceedings of Symposia in Pure Mathematics*, vol. 20, pp. 415–440, 1969.
- [32] J. Pollard, “Monte Carlo Methods for Index Computation (mod  $p$ ),” *Math. Computation*, vol. 32, no. 143, pp. 918–924, 1978.
- [33] A. Menezes, S. Vanstone, and T. Okamoto, “Reducing elliptic curve logarithms to logarithms in a finite field,” in *STOC 91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1991, pp. 80–89.
- [34] J. C. Cha and J. H. Cheon, “An Identity-Based Signature from Gap Diffie-Hellman Groups,” in *PKC 03: Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography*. London, UK: Springer-Verlag, 2003, pp. 18–30.
- [35] R. Sakai, K. Ohgishi, and K. Kasahara, “Cryptosystems based on Pairing,” in *2000 Symposium on Cryptography and Information Security (SCIS2000)*, Okinawa, Japan, 2000, pp. 26–28.

- [36] N. P. Smart, "An identity based authenticated key agreement protocol based on the Weil pairing," *Electronics Letters*, vol. 38, no. 13, pp. 630–632, 2002.
- [37] A. Joux, "A One Round Protocol for Tripartite Diffie-Hellman," in *ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory*. Springer-Verlag, 2000, pp. 385–394.
- [38] S. S. Al-riyami and K. G. Paterson, "Authenticated Three Party Key Agreement Protocols from Pairings," in *Cryptology ePrint Archive, Report 2003/167*, 2003.
- [39] D. Hankerson, A. Menezes, and S. Vanstone, *Guide To Elliptic Curve Cryptography*. Springer, 2004.
- [40] A. K. Lenstra and E. R. Verheul, "Selecting Cryptographic Key Sizes," *Journal of Cryptology*, vol. 14, pp. 255–293, 2001.
- [41] Certicom Corporation, *SEC 2: Recommended Elliptic Curve Domain Parameters*, september 2000. [Online]. Available: <http://www.secg.org>
- [42] Faraday Technology Corporation, *0.13 $\mu$ m Platinum Standard Cell Databook*, 2004. [Online]. Available: <http://www.faraday-tech.com>
- [43] K. Sakiyama, "Secure Design Methodology and Implementation for Embedded Public-key Cryptosystems," Ph.D. dissertation, KU Leuven, december 2007.
- [44] L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede, "Low-Cost Elliptic Curve Cryptography for Wireless Sensor Networks." in *ESAS*, ser. Lecture Notes in Computer Science, vol. 4357. Springer, 2006, pp. 6–17.
- [45] G. Bertoni, L. Breveglieri, P. Fragneto, G. Pelosi, and L. Sportiello, "Software implementation of Tate pairing over  $\text{GF}(2^m)$ ," in *DATE 06: Proceedings of the conference on Design, automation and test in Europe*. European Design and Automation Association, 2006, pp. 7–11.
- [46] L. Batina, "Arithmetic And Architectures For Secure Hardware Implementations Of Public-Key Cryptography," Ph.D. dissertation, KU Leuven, 2005.
- [47] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in  $\text{GF}(2^m)$  using normal bases," *Information and Computation*, vol. 78, no. 3, pp. 171–177, 1988.
- [48] A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata," *Soviet Physics Doklady*, vol. 7, p. 595, 1963.



- [49] D. Zuras, “On squaring and multiplying large integers,” in *Proceedings of IEEE 11th Symposium on Computer Arithmetic ARITH-93*, 1993, p. 260.
- [50] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. Rodriguez-Henrquez, “A Comparison Between Hardware Accelerators for the Modified Tate Pairing over  $\mathbb{F}_{2^m}$  and  $\mathbb{F}_{3^m}$ ,” in *Lecture Notes in Computer Science*, vol. 5209. Springer, 2008, pp. 297–315.
- [51] Y. K. Lee and I. Verbauwhede, “A Compact Architecture for Montgomery Elliptic Curve Scalar Multiplication Processor.” in *WISA*, ser. Lecture Notes in Computer Science, vol. 4867. Springer, 2007, pp. 115–127.
- [52] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low Power Methodology Manual: For System-on-Chip Design*. Springer Publishing Company, Incorporated, 2007.
- [53] M. Müller, A. Wortmann, S. Simon, M. Kugel, and T. Schoenauer, “The impact of clock gating schemes on the power dissipation of synthesizable register files,” in *ISCAS (2)*, 2004, pp. 609–612.
- [54] P. Schaumont, D. Ching, H. Chan, J. Steensgaard-Madsen, A. Vad-Lorentzen, and E. Simpson, *GEZEL User Manual*, 2007. [Online]. Available: <http://rijndael.ece.vt.edu/gezel2>
- [55] J.-L. Beuchat, H. Doi, K. Fujita, A. Inomata, A. Kanaoka, M. Katouno, M. Mambo, E. Okamoto, T. Okamoto, T. Shiga, M. Shirase, R. Soga, T. Takagi, A. Vithanage, and H. Yamamoto, “FPGA and ASIC Implementations of the  $\eta_T$  Pairing in Characteristic Three,” in *Cryptology ePrint Archive, Report 2008/280*, 2008.
- [56] D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, R. Leupers, R. Mathar, and H. Meyr, “Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves,” in *Cryptology ePrint Archive, Report 2009/056*, 2009.
- [57] G. Kömürcü and E. Savas, “An Efficient Hardware Implementation of the Tate Pairing in Characteristic Three.” in *ICONS*. IEEE Computer Society, 2008, pp. 23–28.