Hoofdstuk 1

Inleiding

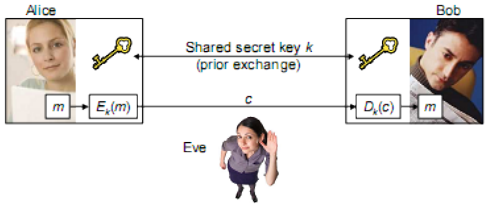
In dit inleidende hoofdstuk zal enige achtergrond informatie verschaft worden omtrent cryptografie. Verder zal het concept van identiteitsgebaseerde cryptografie duidelijk gemaakt worden. Er zal uitgelegd worden waarom de recente ondekking van pairings hier zo belangrijk voor is. Ten slotte zal een kort overzicht gegeven worden van in de literatuur terug te vinden 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.

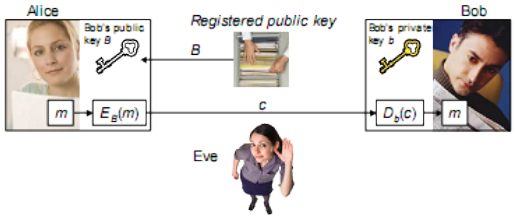
De algemene werking van symmetrische cryptografie is weergegeven in Figuur 1.1. Alice zendt een bericht B naar Bob door het te versleutelen, vercijferd met een door hen beiden gekende sleutel k. Bob op zijn beurt ontcijfert met diezelfde sleutel het bericht. Indien Eve de vooraf afgesproken sleutel kent, kan zij alle communicatie tussen Alice en Bob ontcijferen. Er is dus nood aan een manier om veilig een sleutel k te kunnen afspreken 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 crypto-



Figuur 1.1: Algemene werking van symmetrische cryptografie

grafie genoemd). Met behulp van dit type cryptografie kunnen eveneens boodschappen versleuteld worden door gestuurd. Dit wordt geïllustreerd in Figuur 1.2. Wanneer Alice een bericht naar Bob wil versturen, zoekt ze eerst zijn publieke sleutel op in een databank. Vervolgens versleutelt ze haar bericht met Bobs publieke sleutel. Enkel Bob kan met behulp van zijn private sleutel dan het bericht ontcijferen.



Figuur 1.2: Algemene werking van asymmetrische cryptografie

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.

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 aan de hand van een digitale handtekening worden nagegaan 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 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 kunnen sleutels opgenomen worden in een zogenaamde "revocation list", waardoor wordt aangegeven dat die sleutel niet meer geldig is.

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 zou dit probleem 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 er van uit gegaan dat deze levenslang gelden. Een gebruiker dient zich dus slechts éénmalig 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 methode bedacht worden om alle nieuwe sleutels bij de gebruikers te krijgen.

Door deze problemen is de toepassing van identiteitsgebaseerde crypto-

grafie 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 die extra communicatie niet meer nodig, aangezien de benodigde publieke sleutels berekend kunnen worden uit een unieke ID.

Een uitvoerig overzicht van identiteitsgebaseerde cryptografie en de mogelijkheden die ze biedt, valt buiten het bestek van thesis. Een uitstekend startpunt voor meer informatie is [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 η_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. Daarbij zal gekeken worden naar de mogelijkheden van implementaties op een microchip en in een FPGA. Implementaties van pairings op computers zijn per definitie niet in een omgeving met beperkte ressources 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] wordt het vermogenverbruik geschat op 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	η_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 zijn voor toepassingen met beperkte ressources.

Toch wordt in Tabel 1.2 een sumier 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. [slices]	f [MHz]	Rekentijd [μs]
Ronan *et al.* [21]	$\mathbb{F}_{2^{103}}$	Mod. Tate	21021	51	206
Shu *et al.* [22]	$\mathbb{F}_{2^{239}}$	Mod. Tate	25287	84	41
Keller *et al.* [23]	$\mathbb{F}_{2^{251}}$	Mod. Tate	27725	40	2370
Grabher en Page [24]	$\mathbb{F}_{3^{97}}$	Mod. Tate	4481	150	432
Beuchat *et al.* [25]	$\mathbb{F}_{3^{97}}$	η_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 opnieuw verwezen naar [11]. Het is ook uit dit werk dat de informatie uit de volgende paragrafen afkomstig is, tenzij anders vermeld.

2.1 Inleidende wiskunde

Alvorens de wiskundige theorie van pairings uit de doeken gedaan kan worden, 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 echter gedefinieerd over een eindig veld. Vandaar dat de noodzakelijk 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

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

Volgens de kleine stelling van Fermat geldt in elk eindig veld $a^q = a$. Van deze gelijkheid zal in het volgende hoofdstuk handig gebruikt gemaakt worden wanneer de inverse van een element moet berekend worden. Het voordeel van werken in een binair veld, m.a.w. $q = 2$, is dat optellingen en aftrekkingen equivalent zijn en zeer makkelijk uit te voeren zijn. Het is immers zo dat $1 + 1 = 2 \bmod 2 = 0$ en $0 - 1 = -1 \bmod 2 = 1$. Beiden kunnen dus berekend worden via een XOR operatie.

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 reductie veelterm P opgegeven te worden. Een extensieveld wordt gedefinieerd als:

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

Voorbeeld Om de constructie van een extensieveld enigszins te verduidelijken wordt een klein voorbeeld gegeven. Er wordt gewerkt in karakteristiek $q = 2$. Stel $P = z^2 + z + 1$. 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:

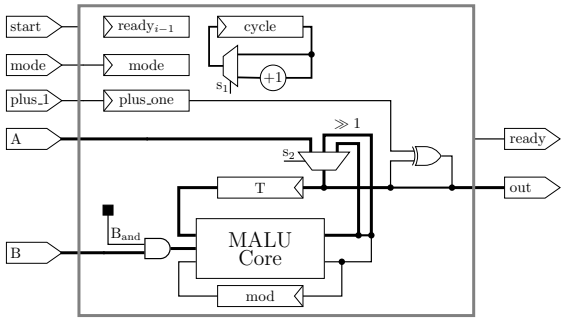
$$E : Y^2Z + a_1XYZ + a_2YZ^2 = X^3 + a_3X^2Z + a_4XZ^2 + a_5Z^3.$$

Het enige punt waarvoor $Z = 0$ en 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 $\frac{\partial P}{\partial X}$, $\frac{\partial P}{\partial Y}$ en $\frac{\partial P}{\partial Z}$ 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 neutral element het punt op oneindig $\mathcal{O}$. Afhankelijk



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

van het veld waarover de kromme gedefinieerd is, zullen de formules om de "tangent-and-chord" methode uit te voeren anders zijn.

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

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

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

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

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 inbeddingsgraad k gedefinieerd als het kleinste gehele getal waarvoor $n \mid q^k - 1$.

2.2 Definitie van pairings

Een pairing is in dit geval een functie $f(A, B)$ met als argumenten twee punten uit een additieve groep en als resultaat een punt in een multiplicatieve groep:

$$f(A, B) : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T.$$

Een pairing moet tevens volgende eigenschappen bezitten: bilineariteit, nondegeneratie en moet goed gedefinieerd zijn. De betekenis van deze drie begrippen is:

1. Bilineariteit: $\forall A_1, A_2, A \in \mathbb{G}_1$ en $\forall B_1, B_2, B \in \mathbb{G}_2$ geldt $f(A_1 + A_2, B) \equiv f(A_1, B) \cdot f(A_2, B)$ en $f(A, B_1 + B_2) \equiv f(A, B_1) \cdot f(A, B_2)$.
2. Non-degeneratief: $\forall A \in \mathbb{G}_1$, $\exists B \in \mathbb{G}_2$ waarvoor $f(A, B) \neq 1$.
3. Goed gedefinieerd: $\forall B \in \mathbb{G}_2$ is $f(A, B) = 1$ als en slechts als $A = \mathcal{O}$.

Het zijn de drie eigenschappen waaraan pairings moeten voldoen die het zo moeilijk maken ze te genereren. Ten tijde van dit schrijven waren onder meer volgende pairings bekend: Weil, Tate, η_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 η_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 definitie van de Tate pairing over elliptische krommen is als volgt:

$$e(A, B) : E(\mathbb{F}_{q^k})[n] \times E(\mathbb{F}_{q^k})/n \cdot E(\mathbb{F}_{q^k}) \rightarrow \mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^n$$

Het resultaat van de Tate pairing is niet uniek, maar een element van een equivalentieklasse in $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^n$. Voor twee resultaten $a, b \in \mathbb{F}^*_{q^k}$ geldt $a \equiv b$ als en slechts als er een element $c \in \mathbb{F}^*_{q^k}$ waarvoor $a = bc^n$. 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 $\frac{q^k-1}{n}$. Aangezien $a^{q^k-1} = 1$ voor elke $a \in \mathbb{F}^*_{q^k}$ (kleine stelling van Fermat), verkrijgt men dan dus een n-eenheidswortel.

2.3 Berekening van de Tate pairing

Reeds in 1986 stelde Miller een methode voor om de Tate pairing te berekenen [27, 28]. Algoritme **??** geeft weer hoe dat algoritme in z'n werk gaat.

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 **??**) 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 de implementatie, moeten bepaalde parameters vastgelegd worden. Pas eens dat gedaan is, zijn alle bewerkingen volledig gedefinieerd.

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 een supersinguliere elliptische kromme. Dat type krommen heeft laat een kleine inbeddingsgraad toe en geeft aanleiding tot simpele rekenregels voor de verdubbeling en optelling van punten [11, 29]. Een vergelijking van zo een kromme E over een veld $\mathbb{F}_{2^m}$ wordt gegeven door

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

waarbij $b \in \{0, 1\}$. Verder worden de volgende hulp variabelen bepaald:

$$\begin{aligned} \delta &= b & m &\equiv 1, 7 \pmod 8 \\ &= 1 - b & m &\equiv 3, 5 \pmod 8 \\ \nu &= (-1)^{\delta} \end{aligned}$$

De orde van de kromme is gelijk aan [30, 31]:

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

Om de Tate pairing 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. Die 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 [32] wordt gekozen voor $m = 163$. Dit geeft een cryptografische sterkte vergelijkbaar met 86 bits symmetrische cryptografie. Dit zou neerkomen op een geganrandeerde 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 over E vastgelegd worden. Daarbij moet er op gelet worden dat $l \mid \#E$. De eenvoudigste keuze is:

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

In dit geval is inbeddingsgraad $k = 4$ [30].

Nu het veld waarover gewerkt wordt, bepaald is, kan een reductie veelterm P gekozen worden. Bij de keuze daarvan werd uitgegaan van de aangeraden parameters in [33]. De veelterm is:

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

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 ϕ bepaald worden. Hiervoor wordt dezelfde gekozen als in [30]. Een punt $A \in \mathbb{F}_{2^m}$ ondergaat onder de distortie volgende transformatie:

$$\phi : (x_A, y_A) \rightarrow (x_A + s^2, y_A + x_A s + t^6)$$

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:

$$s = x + 1,$$
$$t = xy$$

Een distortie van A kan dus ook geschreven worden als:

$$x_\phi = x_A + x,$$
$$y_\phi = (x_A + y_A) + x_A \cdot x + xy.$$

Merk op dat de constante term van x_ϕ en de coëfficient van x van y_ϕ 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 $0,13\mu m$ 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 [34])

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 Arithmetische Logische Unit

De kern van de hardware implementatie wordt gevormd door de Modular Arithmetic Logical Unit (MALU) [35, 36]. 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:

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

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 3.1.

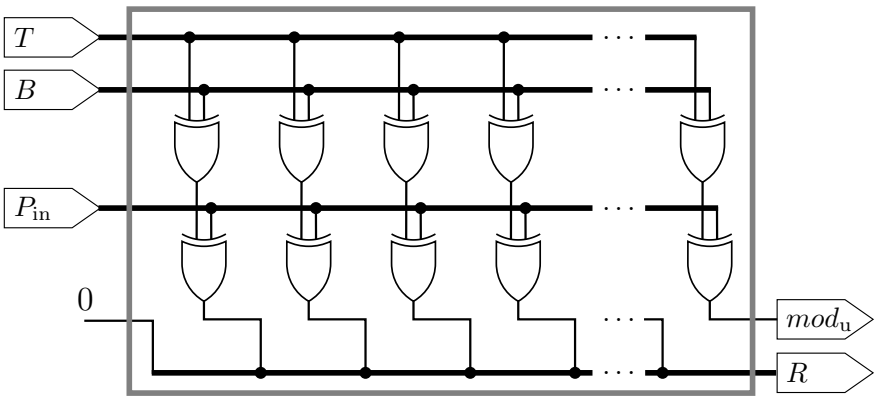
In Paragraaf 3.5.2 zal blijken dat het vaak nodig zal zijn om het resultaat R te vermenigvuldigen met z, m.a.w. alle bits 1 plaats naar links te verschuiven. Een voor de hand liggende schakeling die dit alles implementeert, is te zien in Figuur 3.1. Ingang P_{in} dient afhankelijk van de graad van T ingesteld te worden op 0 of P. De ingangen T en P_{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 3.1: Modulo optelling in $\mathbb{F}_{2^m}$

Input: $B \in \mathbb{F}_{2^m}$, $T \in \mathbb{F}_{2^{m+1}}$
Output: $R \mod P \in \mathbb{F}_{2^m}$

```
1 R ← T ⊕ B
2 if degree(R) = m then
3 |  R ← R ⊕ P
```

de shift wordt naar buiten gebracht als mod_{u}. De implementatie bestaat uit $2m$ XOR poorten.



Figuur 3.1: MALU - Basis ontwerp met shift

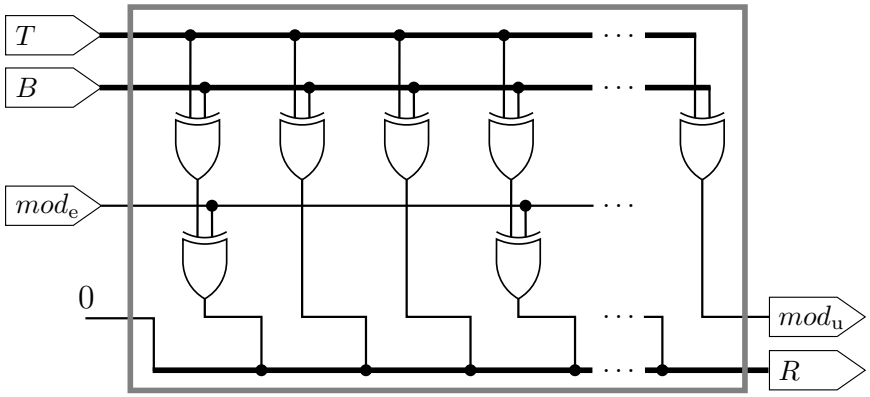
Aangezien voor het ontwerp het veld en de modulo veelterm op voorhand bepaald zijn, is het mogelijk een zeer groot aantal XOR poorten uit het ontwerp te verwijderen. De ingang P en de bijhorende m XOR poorten kunnen vervangen worden door een 1 bit 'modulo enable' ingang mod_{e} en er worden enkel XOR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

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

XOR poorten uitgespaard.

In dit geval is $m = 163$ en $P = z^{163} + z^7 + z^6 + z^3 + 1$. Er zijn dus $\mathsf{Hamm}(P) - 1 = 4$ XOR poorten nodig, wat een besparing van $163 - 4 = 159$ XOR poorten oplevert (51% minder dan het oorsponkelijk 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 3.2 is te zien hoe dit juist in z'n werk gaat. Door de modulo bewerking 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 te schakeling te weten welke van de twee bewerkingen moet uitgevoerd worden. Verder moet het mogelijk zijn de uitkomt 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 3.2) is $\lceil \log_2(m) \rceil$ bits lang en register T m bits. De waarde van T_m wordt opgeslagen in regis-

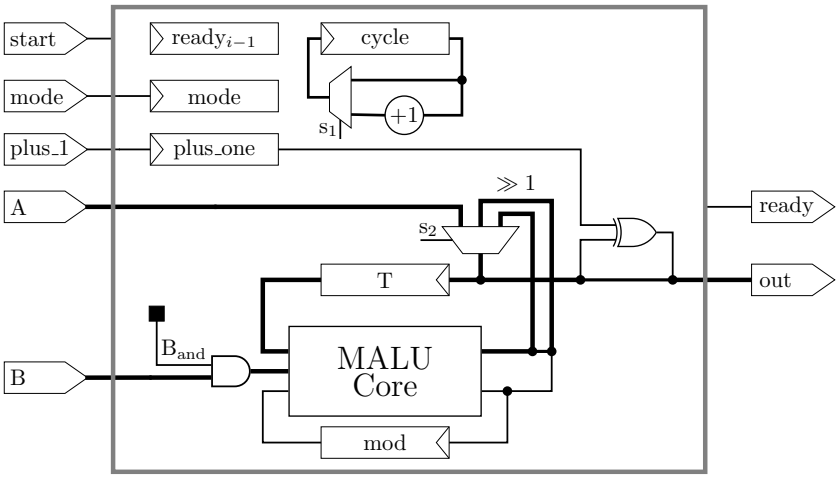
Algoritme 3.2: "Shift and add" vermenigvuldiging in $\mathbb{F}_{2^m}$

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 ← 0
2  for i ← m − 1 to 0 do
3      if A_i = 1 then
4          b ← B
5      else
6          b ← 0
7      T ← T ⊕ b
8      if degree(T) = m then
9          T ← T ⊕ P
10     T ← T ≪ 1
11 R ← T ≫ 1
```

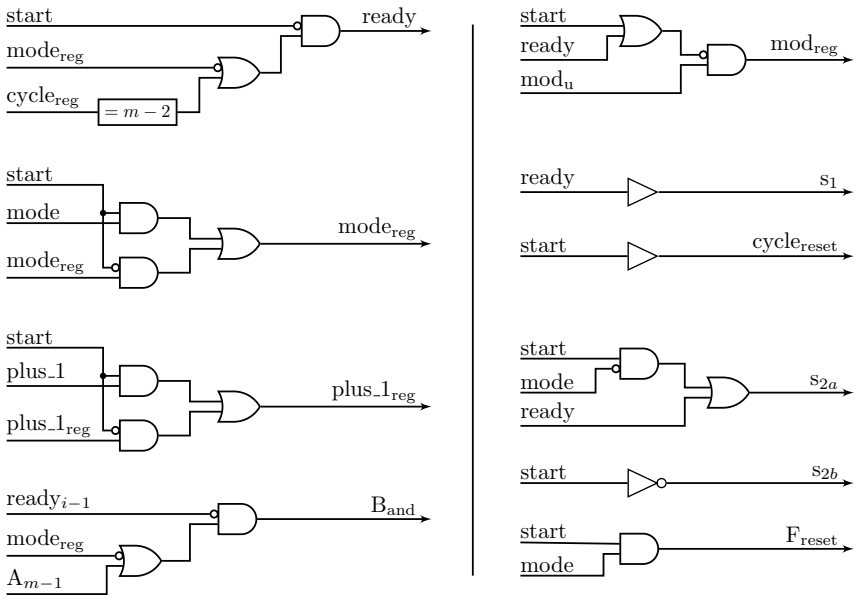
ter *mod*. Alle overige registers zijn 1 bit groot. Merk dus op dat de variabele T uit Algoritme 3.2 hier gevormd wordt door de combinatie van de registers T en *mod*.

Voor A wordt geen appart register voorzien en in plaats van A_i wordt steeds bit A_m ingelezen voor de vermenigvuldiging. De schakeling die van deze schakeling gebruik maakt, 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 implementeren, 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 $R_{\text{ready}} = R \gg 1$.

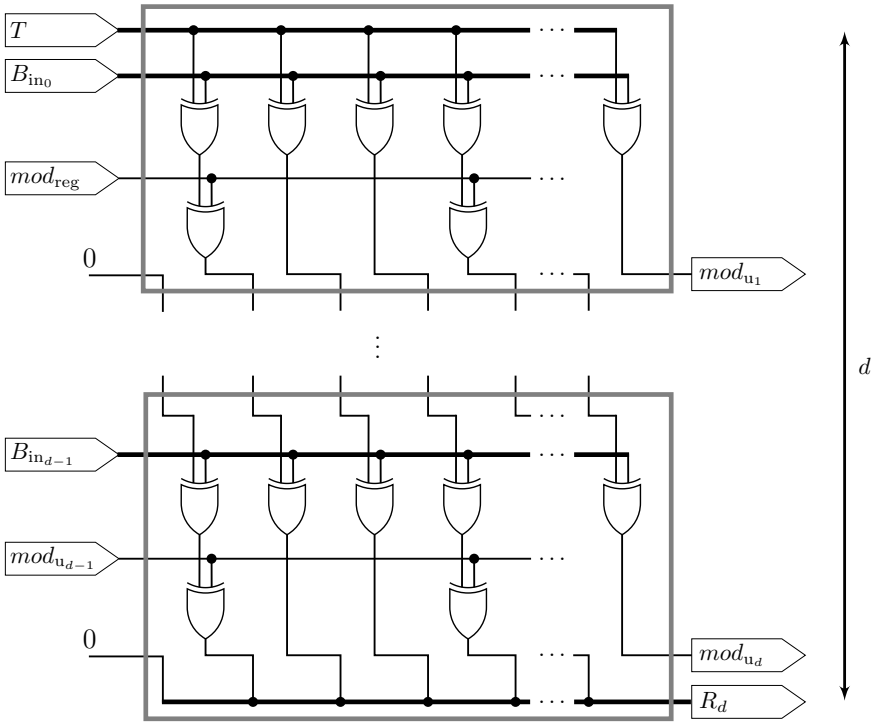


Figuur 3.4: Logica voor besturing van de $\mathbb{F}_{2^m}$ kern

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 vermenigvuldigigen er uitgevoerd dienen te worden, des te significanter de tijdswinst die geboekt kan worden. 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



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

dienen toegevoegd te worden, afhankelijk van d en m dient ook een extra multiplexer in 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

$$R_{3_{\text{ready}}} = mod_{\text{u}_3} \# R_{3_{162:1}},$$

terwijl dit voor een optelling

$$R_{1_{\text{ready}}} = mod_{\text{u}_1} \# R_{1_{162:1}}$$

is. Met andere woorden, er 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: Locatie van het resultaat van een vermenigvuldiging 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 (Algoritme **??**) uitvoert. Het algoritme met invulling van de vastgelegde parameters, zonder uitwerking van de berekeningen, wordt gegeven in Algoritme 3.3.

Algoritme 3.3: Miller algoritme voor berekening van de Tate pairing met parameters ingevuld

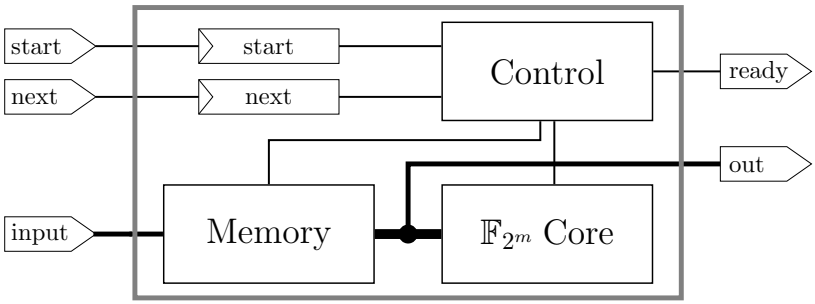
Input: $P, Q \in E(\mathbb{F}_{2^{163}})[l]$
Output: $e(P, Q)$

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

Merk op dat op lijn 6 slechts één waarde moet nagekeken worden, aan-

gezien $l = 2^{163} + 2^{82} + 1$.

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 3.3. Voor een supersinguliere kromme zijn de berekeningen als volgt [30, 29]:

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

In dit geval kan y_{2V} ook berekend worden 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:

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

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.

Tanneer dit in rekening gebracht wordt en het algoritme op register niveau wordt uitgeschreven, bekomt men uiteindelijk Algoritme 3.4. Hierbij werd specifiek gelet op een minimum gebruik van tijdelijke registers.

Buiten zes registers voor x_{2V}, y_{2V}, x_{ϕ_a}, y_{ϕ_a}, G_a en G_b, is er ook één register nodig om λ 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 3.3. Voor een supersinguliere kromme dienen de volgende bewerkingen uitgevoerd te worden [30, 29]:

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

Algoritme 3.4: 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 3.5.

In tegenstelling tot de verdubbelstap zijn hier twee tijdelijke registers nodig, een voor λ 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 3.5: 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 Galois veld 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 vermenigvuldigingen. 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 vermenigvuldigen nodig zijn.

Er is echter een derde, optimale manier die toegepast kan worden indien de exponent van de vorm $2^m - 2$ is [37, 38]. 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 $\lfloor \log_2(m-1) \rfloor + \mathsf{Hamm}(m-1) + 1$ vermenigvuldigingen en $m - 1$ kwadrateringen.

In het geval van $m = 163$ is de uiteindelijke keten van bewerkingen zoals gegeven in Algoritme 3.6. 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 3.3 moet ook telkens het kwadraat van F berekend worden. De afleiding van de formule daarvoor gaat als volgt:

Algoritme 3.6: Inversie in $\mathbb{F}_{2^{163}}$

Input: $a \in \mathbb{F}_{2^{163}}$
Output: $a^{-1} \in \mathbb{F}_{2^{163}}$

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

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

Mist 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 3.7. 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 optelstap niet van volledige rang in het extensieveld. De vermenigvuldiging van F met G kan daardoor vereenvoudigd worden, namelijk als volgt:

Algoritme 3.7: Uitwerking van van $F^2 \in \mathbb{F}_{2^{4m}}$

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

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

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

Indien hier nu de Karatsuba-Ofman techniek [39, 40] op wordt toegepast, bekomt men:

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

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 3.8 beschrijft welke berekeningen juist uitgevoerd moeten worden.

Mist 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 [31], 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 3.8: 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-loop voltooid is, moet F nog gereduceerd worden zodat het eindresultaat $e(P, Q)$ uniek is. Hoe dit gebeurt, wordt onderzocht in de volgende paragrafen. De gebruikte methodes zijn gebaseerd op die in [31], 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 $e(P, Q) = F^M$, 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

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

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 $F^{2^{2m}} = U_0 + U_1 + U_1 x + U_1 y$. Men vindt dus:

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

en

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

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

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

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 vind 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 3.9. 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 [31].

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

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

Algoritme 3.9: 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 3.10, 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 3.10: 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:

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

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 3.11 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 3.11: 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 er van uit gegaan 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 3.12 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

Algoritme 3.12: Uitwerking van $I^{2^m+1} \in \mathbb{F}_{2^{4m}}$

Input: $I = I_a + I_b x + I_c y + I_d xy \in \mathbb{F}_{2^{4m}}$
Output: $I = I^{2^m+1} \in \mathbb{F}_{2^{4m}}$
Data: $a, b, \ldots, e \in \mathbb{F}_{2^m}$

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

geval is J simpelweg $\frac{m+1}{2}$ opeenvolgende keren te kwadrateren. Hiervoor kan Algoritme 3.7 gebruikt worden. In totaal zal deze stap $2 \cdot (m+1)$ vermenigvuldigigen 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:

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

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 $e(P,Q)$ geschreven worden als:

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

Indien zowel I als J overschreven mogen worden, is de minimum benodigde opslagruimte om deze berekeningen uit te voeren vier registers voor het eindresultaat $I = e(P,Q)$ en zes tijdelijke registers voor $a, b, \ldots, f$. Dit alles samen vergt negen vermenigvuldigingen en tweeëntwintig optellingen, twee meer dan in [31]. De methode is terug te vinden in Algoritme 3.13.

Algoritme 3.13: 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: $a, b, \ldots, f \in \mathbb{F}_{2^m}$

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.

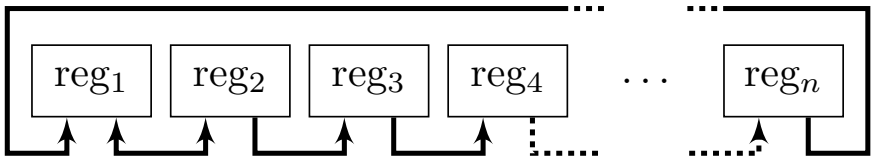
Van de vijf algoritmes die opgeroepen worden in de for-lus, vereist de de vermenigvuldiging van F en G het grootste aantal tijdelijke registers, namelijk drie. Het minimum aantal registers om alle bewerkingen in de lus te kunnen uitvoeren is dus vijftien: zes voor $P, \phi(Q)$ en V, twee voor G, vier 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 [41]. 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 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 $\overline{r}$ 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

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

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 $\overline{r} \cdot n$ doorschuif bewerkingen nodig. Dit is opnieuw een ruwe schatting, die voor kleine n niet zal kloppen. Algemeen kan echter gesteld worden dat

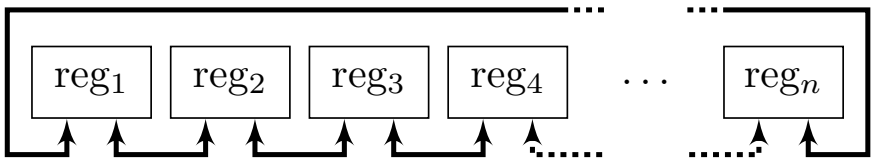
$$\bar{t} \in O\Big(\frac{n^2}{3}\Big).$$

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

$$\overline{w} \in O\Big(\frac{n^3}{3}\Big).$$

Gezien het feit dat er vijftien registers zijn (tegenover zes in [41]), 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 $\mathsf{min}(j-1, n-j+1)$ omdat nu in beide richtingen geschoven kan worden. De gemiddelde afstand $\overline{r}$ is dan

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

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

$$\overline{t} \in O\Big(\frac{n}{4}\Big).$$

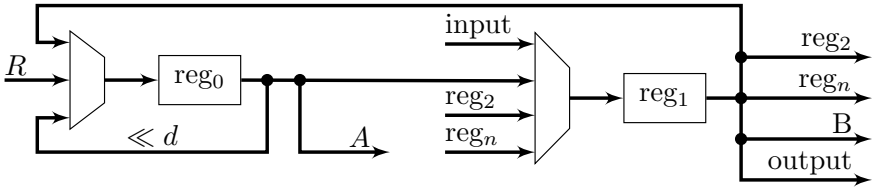
Rekening houdend met het feit dat elke omwisselbewerking twee schrijfbewerkingen vraagt en er twee variabelen naar de juiste positie gebracht moeten worden, geldt

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

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 doorschuif lus 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. Rekend 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

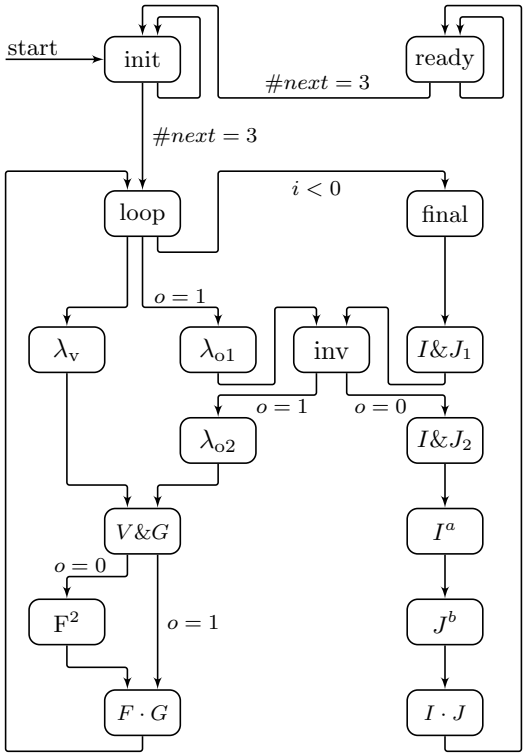
3.5.5 FSM

Nut 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. Ook zal zoveel mogelijk getracht worden reeds geïmplementeerde toestanden opnieuw te gebruiken.

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'. 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 opsplitsing van de verdubbelen optelstap. Wanneer Algoritme 3.4 en 3.5 opnieuw bekeken worden, valt het op dat buiten de berekening van λ 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 be-

staan beide algoritmes uit dezelfde bewerkingen na het berekenen van λ en x_V. Aangezien na de berekening van λ nog steeds twee registers vrij zijn voor gebruik, kunnen deze aangewend worden om de toepasselijke 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 optelstap 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 $\mathsf{log}_2(m)$ 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 Miller loop.

Een register is doorgaans opgebouwd uit een aantal D-type master-slave flip-flops. In de gebruikte bibliotheek [34] 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 er van uit gegaan 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 resetingangen 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.

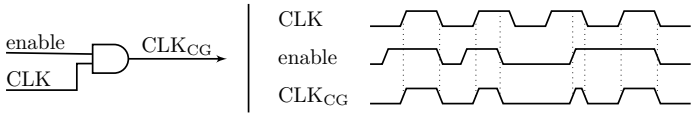
3.6.2 Clock gating

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 [42]. 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.

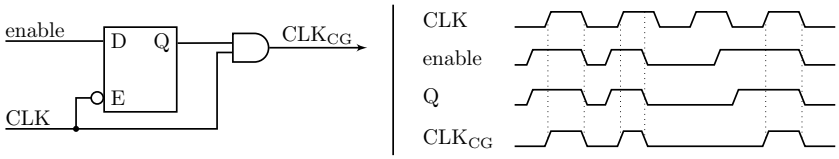
De eenvoudigste schakeling waarmee clock gating geïmplementeerd kan worden is te zien in Figuur 3.11. Er zijn echter enkele problemen geassocieerd met dit type schakeling. De uitgang is immers onderhevig aan onregelmatigheden op het enable signaal. Stel bijvoorbeeld dat de enable ingang al hoog wordt 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



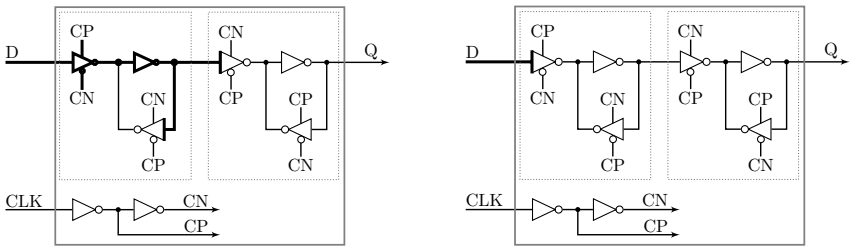
Figuur 3.11: Schakeling voor clock gating - Basis ontwerp

register kunnen bereiken.



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 [43]. In die paper wordt bewezen dat het energie verbruik 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.

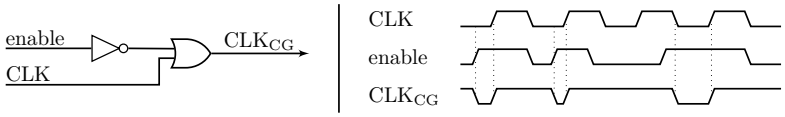


(a) $CLK = 0$ (b) $CLK = 1$

Figuur 3.13: Vermogenverbruikende onderdelen van een D-type master-slave flip-flop bij constante waarde van de klokingang [43]. 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 kloksignaalbeperkingen 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 [44]. 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 0.13μm *low leakage* bibliotheek van Faraday Technology [34]. 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 kloksnelheiden 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 syntheseresultaten te presenteren, wordt even uitgewijd over het noodzakelijk aantal klokcycli c om één pairing te berekenen. Merk op dat de formules die volgen enkel kloppen voor willekeurige m indien $\mathsf{Hamm}(l) = 3$, 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 coëfficient 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:

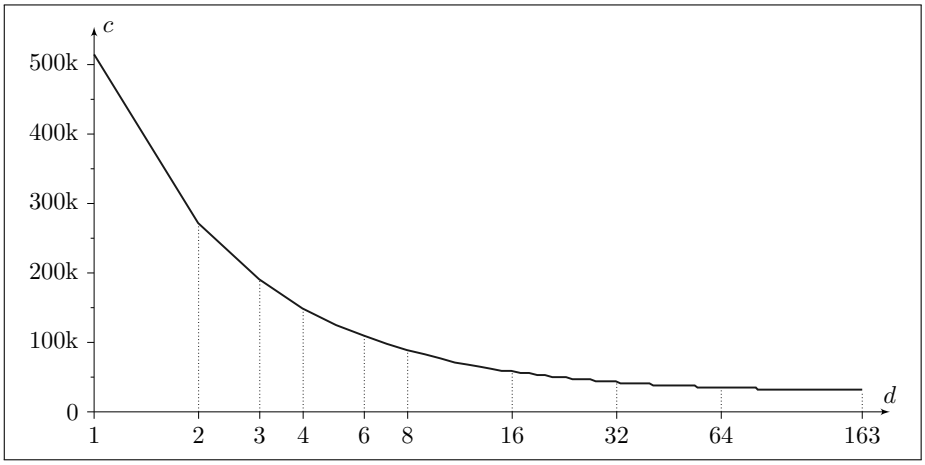
$$c_f = 18731 + 3937 + 2463 \cdot \left\lceil \frac{m}{d} \right\rceil$$
$$c_m = 2950 + 385 + 535 \cdot \left\lceil \frac{m}{d} \right\rceil,$$

waarbij de coëfficienten 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 resetingangen van zoveel mogelijk registers verwijderd. Ten slotte zal onderzocht worden hoeveel oppervlakte de individuele onderdelen van het ontwerp innemen.

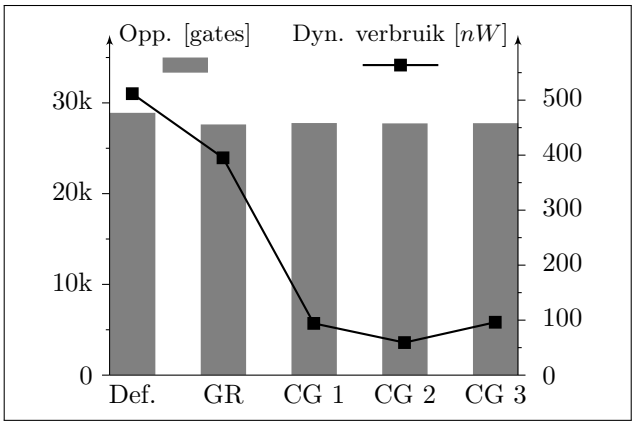
De syntheseresultaten voor de vijf verschillende implementaties worden gegeven in Tabel 4.2. De versies met clock gating (CG n) implementeren de schakelingen in de volgorde waarin ze voorkomen in Paragraaf 3.6.2. Ter verduidelijking zijn deze resultaten ook nog eens uitgezet in Figuur 4.2.

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 synthesesoftware dit niet doet, is onduidelijk. Er werd getracht de synthesesoftware 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. Het effect daarvan zal duidelijk worden in de volgende paragraaf, wanneer meerdere MALU's geïmplementeerd worden.

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. Zoals in de vorige paragraaf gezien, neemt een MALU zeer weinig oppervlakte in in de implementatie. Het toevoegen van een klein aantal extra MALU's brengt dus 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.

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 geval-

Tabel 4.4: Syntheseresultaten voor implementaties met d MALU's

d	Opp. [gates]		Vermogen @ 10 kHz $[nW]$				Tijds-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%

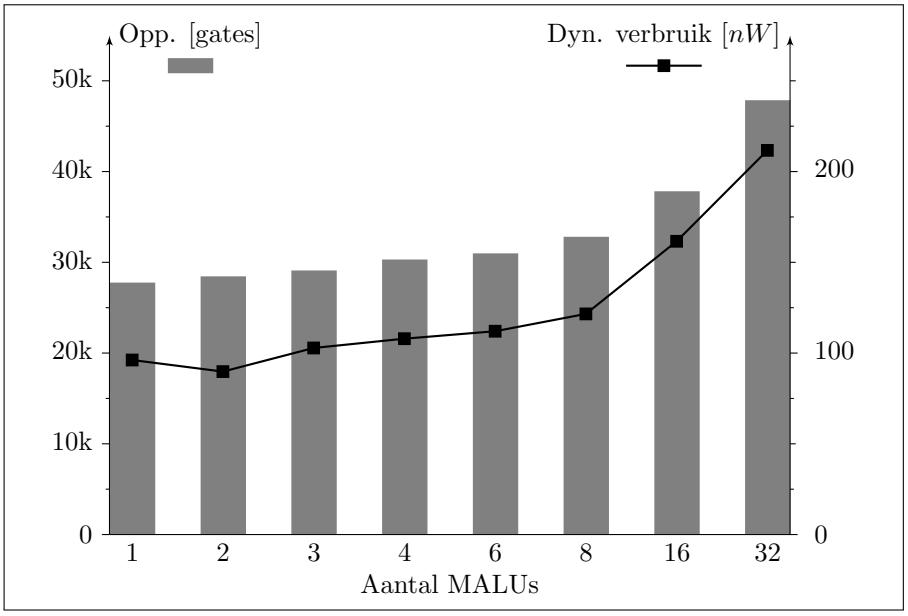
len onaanvaardbaar zijn. Daarom wordt hier onderzocht wat de effecten op de schakeling zijn indien de kloksnelheid wordt opgedreven en er eventueel meerdere MALU's gebruikt worden. Ter illustratie zal voor een implementatie met meerdere MALU's die met twee nader onderzocht worden. Opnieuw wordt in beide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

Stel een maximale rekentijd $t_{\max} \approx 50\ ms$. 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 $f = 10$ 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.



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

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 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 [45], [46] en [47]. Zowel in [45] als [47] wordt met het oog op het behalen van zo hoog mogelijke snelheden ontworpen. De implementatie uit [46] bevat naast de schakeling voor pairings tevens een RISC processor. In [47] 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 [45] het enige is waarvan alle gegevens bekend zijn, zal in de tekst die volgt enkel met dat ontwerp vergeleken worden.

De hier voorgestelde implementaties zijn niet alleen ongeveer een factor zeven kleiner dan degene voorgesteld in [45], hun vermogenverbruik per gate (genormaliseerd voor frequentie) is ook beduidend lager. Uiteraard moet er wel op gewezen worden dat het ontwerp uit [45] door het gebruik van een groter veld een grotere veiligheid biedt en, 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. Het is echter wel

Tabel 4.5: Vergelijking van syntheseresultaten 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 [μW]			
Dynamisch	98,2	48,5	49%
Lek	$107 \cdot 10^{-3}$	$111 \cdot 10^{-3}$	104%

duidelijk dat de ontwerpen voorgesteld in deze thesis met kop en schouders uitsteken boven dat van [45] indien compactheid en laag vermogenverbruik prioriteit genieten.

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

	Thesis (1 MALU, CG 3)		Pairing-Lite [45]	Kammler *et al.* [46]	Kömürcü en Savas [47]
	$d = 1$, $f = 10$kHz	$d = 2$, $f = 5.44$MHz			
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	η_T	Optimal 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	$15,8 \cdot 10^3$	$250^{\ddagger}$
Vermogen $[mW]$	$206 \cdot 10^{-6}$	$48,6 \cdot 10^{-3}$	672	?	?
Score $\left[\frac{\text{fJ}}{\text{gate}}\right]^{\S}$	0,74	0,32	17,34	?	?

† De gegeven oppervlakte is die van de complete schakeling inclusief routing.
‡ Exclusief de finale machtsverheffing.
§ Des te lager de score, des te conservatiever springt de implementatie om met energie. De score wordt berekend als score $= \frac{\text{verbruik}}{\text{frequentie} \cdot \text{oppervlakte}}$.

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 snelheiden blijft het vermogenverbruik zeer laag vergeleken met het enige andere gedocumenteerde ontwerp uit de literatuur. 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 vereenvoudingen 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 rekentijd 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 Galois veld de oppervlakte en het vermogenverbruik wijzigt. Mogelijk kunnen ook implementatie van andere types pairings, zoals de η_T en de modified Tate 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. Ruck, "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. Daguano, J. López, and R. Dahab, "TinyTate: Computing the Tate Pairing in Resource-Constrained Sensor Nodes," in *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on.* 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 *Networked Sensing Systems, 2008. INSS 2008. 5th International Conference on*, 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. OEigeartaigh, 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, J. R. Rao and B. Sunar, Eds., 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 η_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] V. S. Miller, "Short Programs for Functions on Curves," 1986.

[28] 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.

[29] D. Hankerson, A. Menezes, and S. Vanstone, *Guide To Elliptic Curve Cryptography*. Springer, 2004.

[30] G. Bertoni, L. Breveglieri, P. Fragneto, G. Pelosi, and L. Sportiello, "Software implementation of Tate pairing over 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.

[31] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. Rodrguez-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.

[32] A. K. Lenstra and E. R. Verheul, "Selecting Cryptographic Key Sizes," *Journal of Cryptology*, vol. 14, pp. 255–293, 2001.

[33] Certicom Corporation, *SEC 2: Recommended Elliptic Curve Domain Parameters*, september 2000. [Online]. Available: http://www.secg.org

[34] Faraday Technology Corporation, 0.13μm *Platinum Standard Cell Databook*, 2004. [Online]. Available: http://www.faraday-tech.com

[35] K. Sakiyama, "Secure Design Methodology and Implementation for Embedded Public-key Cryptosystems," Ph.D. dissertation, KU Leuven, december 2007.

[36] L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede, "Low-Cost Elliptic Curve Cryptography for Wireless Sensor Networks," *Security and Privacy in Ad-Hoc and Sensor Networks*, vol. 4357, pp. 6–17, 2006.

[37] L. Batina, "Arithmetic And Architectures For Secure Hardware Implementations Of Public-Key Cryptography," Ph.D. dissertation, KU Leuven, 2005.

[38] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases," *Information and Computation*, vol. 78, no. 3, pp. 171–177, 1988.

[39] A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata," *Soviet Physics Doklady*, vol. 7, p. 595, 1963.

[40] D. Zuras, "On squaring and multiplying large integers," in *Proceedings of IEEE 11th Symposium on Computer Arithmetic ARITH-93*, 1993, p. 260.

[41] Y. K. Lee and I. Verbauwhede, "A Compact Architecture for Montgomery Elliptic Curve Scalar Multiplication Processor." in *WISA*, ser. Lecture Notes in Computer Science, S. Kim, M. Yung, and H.-W. Lee, Eds., vol. 4867. Springer, 2007, pp. 115–127.

[42] 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.

[43] 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.

[44] 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

[45] 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 η_T Pairing in Characteristic Three," in *Cryptology ePrint Archive, Report 2008/280*, 2008.

[46] 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.

[47] 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.