Hoofdstuk 1

Implementatie

In dit hoofdstuk wordt de implementatie van een schakeling voor de berekening van de Tate pairing uit de doeken gedaan. 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 veld $\mathbb{F}_{2^m}$. Ten slotte is er dan nog de schakeling die alle berekeningen voor het Miller algoritme in goede banen leidt. Allereerst wordt echter gekeken welke beperkingen aan de implementatie opgelegd moeten worden.

1.1 Beperkingen

Het doel is de uiteindelijke schakeling zo klein mogelijk te maken, zodat ze gebruikt kan worden in bv. netwerken van sensoren of smartcards. Beperking van de oppervlakte is dus de belangrijkste factor. Een tweede belangrijke factor is stroomverbruik, maar dat is helaas zeer moeilijk te berekenen. Het verbruik hangt echter samen met de oppervlakte, dus het beperken daarvan zal ook het verbruik ten goede komen. Het verbruik kan 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 kan dit aspect bij het ontwerp van de schakelingen genegeerd worden. Op dit alles zal dieper ingegaan worden in Hoofdstuk **??**

Algemeen kan 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 1.1 is de grootte van de belangrijkste elementen te vinden. Deze cijfers gelden enkel bij gebruik van $0.13nm$ low leakage technologie. De ordening van de elementen blijft echter behouden voor andere technologiën. Uit de tabel blijkt dat het gebruik van flip-flops (registers), adders en multiplexers zoveel mogelijk beperkt moet worden.

1.2 Modular Arithmetic Logical Unit

De kern van de hardware implementatie wordt gevormd door de Modular Arithmetic Logical Unit (MALU)[2]. Dit circuit laat toe basis bewerkingen uit te voeren op getallen. Gezien de beperking die is opgelegd aan de oppervlakte

Tabel 1.1: Grootte van elementen in een ASIC schakeling in gates/bit ($0.13nm$ low leakage technologie)[**?**]

Element	Gates/bit
D flip-flop met reset	6
D flip-flop zonder reset	5.5
D latch	4.25
full adder	5.5
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

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:

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

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 $\leq m$ is en T van graad $\leq m+1$, is de modulo bewerking te implementeren als in Algoritme 1.1.

Algoritme 1.1: Modulo optelling in $\mathbb{F}_{2^m}$

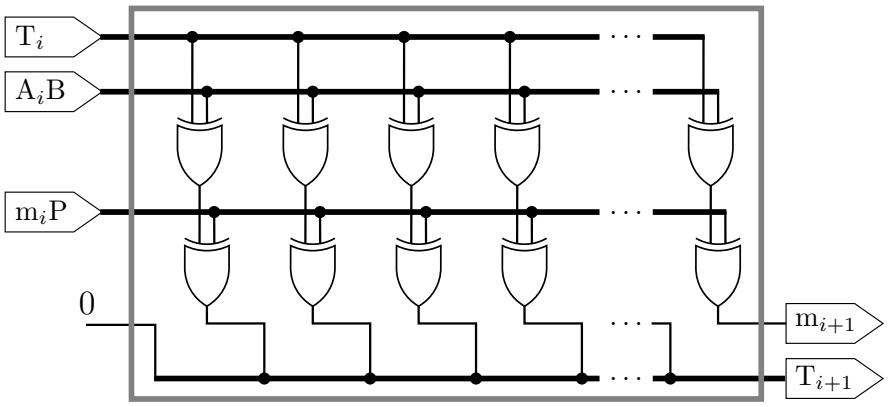
Input: $B \in \mathbb{F}_{2^m}$, $T \in \mathbb{F}_{2^{m+1}}$
Output: $R \mod P \in \mathbb{F}_{2^m}$

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

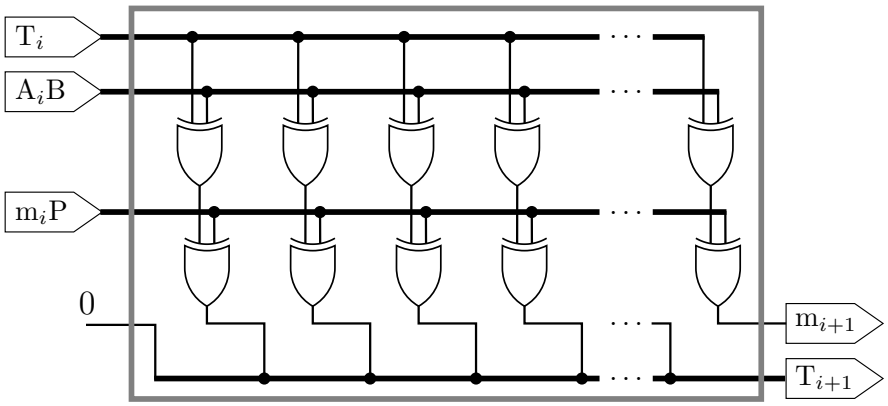
Een voor de hand liggende schakeling die dit alles implementeert, is te zien in Figuur 1.1. Ingang P_{in} dient afhankelijk van T_m ingesteld te worden op 0 of P. P_m kan genegeerd worden, aangezien in het resultaat de graad $< m$ is.

In Sectie 1.3 zal blijken dat het vaak nodig zal zijn om het resultaat R te vermenigvuldigen met z, maw. alle bits 1 plaats naar links te verschuiven. Indien die bewerking wordt toegevoegd, bekomt men de schakeling uit Figuur 1.2. Net als de vorige implementatie bestaat deze uit $2m$ XOR poorten.

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 en de bijhorende m XOR poorten kunnen vervangen worden door een 1 bit 'modulo enable' ingang mod_{in} en er worden enkel XOR



Figuur 1.1: MALU - Basis ontwerp



Figuur 1.2: MALU - Basis ontwerp met shift

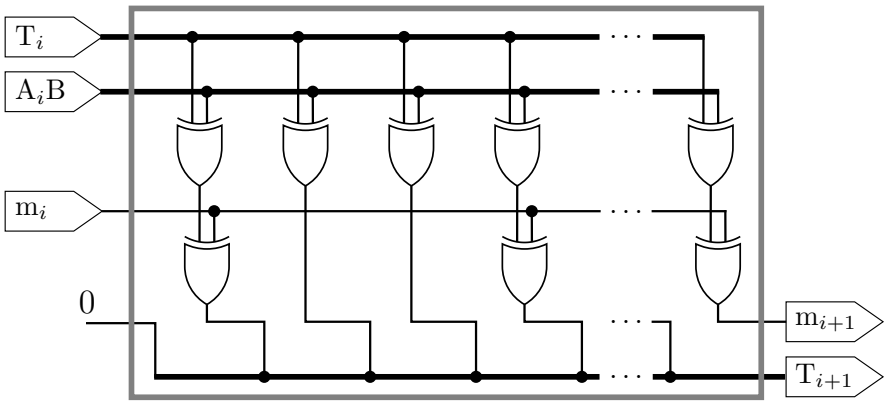
poorten geplaatst voor de bits i waarvoor $P_i = 1$. Hierdoor wordt het aantal ingangen drastisch verkleind en worden

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

XOR poorten uitgespaard, met $\text{hamm}(P)$ gelijk aan het Hamming gewicht van de binaire representatie van P.

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

De resulterende schakeling is te zien in Figuur 1.3.



Figuur 1.3: MALU - Geoptimaliseerd ontwerp met shift

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

1.3.1 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 1.2 is te zien hoe dit juist in z'n werk gaat. Door de modulo operatie 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, is het noodzakelijk een onderscheid te kunnen maken tussen beide bewerkingen. Ook moet kunnen aangegeven worden wanneer de berekening klaar is. Ten slotte moet het resultaat opgeslagen kunnen worden, zodat de uitgang van de schakeling correct blijft. Omdat in het Miller algoritme verscheidene keren de som $R + 1$ moet berekend worden, wordt ook een ingang *plus_one* voorzien. De uiteindelijke schakeling is te zien in Figuur 1.4. Het register *cycle* is $\log_2(m)$ bits groot en T is m bits. De waarde van F_m wordt opgeslagen in register *mod*. Alle overige registers zijn 1 bit groot.

Gezien de eenvoud van de schakeling is het niet nodig een FSM te implementeren, de besturing kan volledig via logica gebeuren. Die logica wordt getoond in Figuur 1.5.

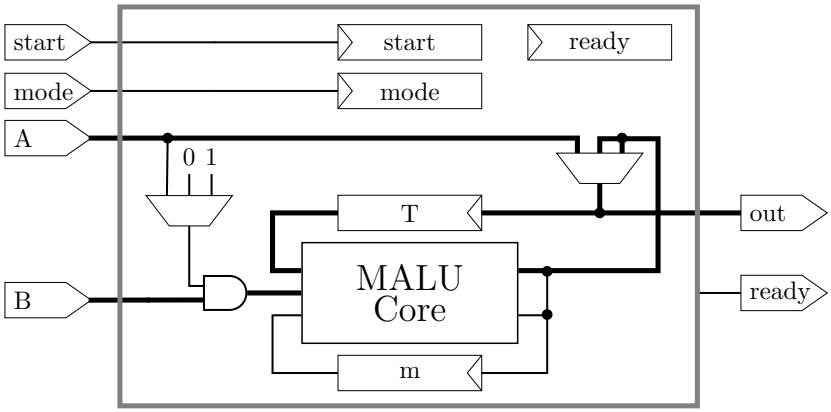
1.3.2 Versnelling van de vermenigvuldiging

Wanneer met behulp van de schakeling in Figuur 1.4 een vermenigvuldiging wordt berekend, zal het m klokcycles duren eer het resultaat beschikbaar is aan

Algoritme 1.2: "Shift and add" vermenigvuldiging in $\mathbb{F}_{2^m}$

Input: $A, B \in \mathbb{F}_{2^m}/[P]$
Output: $R \in \mathbb{F}_{2^m}/[P]$
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
```

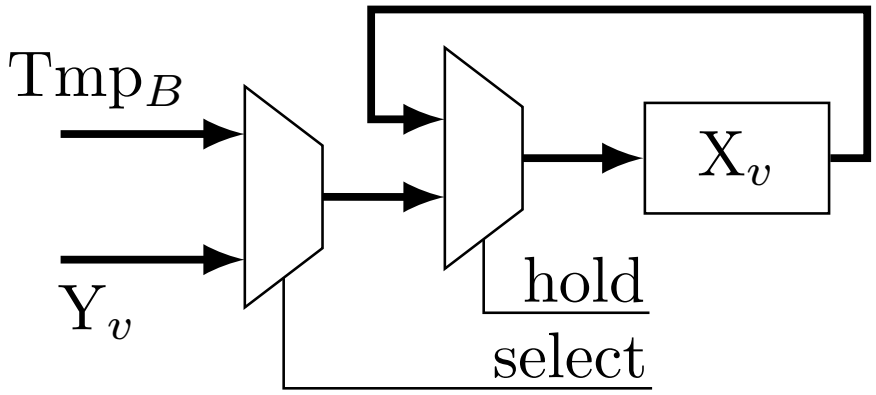


Figuur 1.4: Schakeling voor berekeningen in $\mathbb{F}_{2^m}$

de uitgang. Het is echter mogelijk dat aantal drastisch naar beneden te halen door d MALU's te gebruiken en dus d optellingen per klokcycle uit te voeren. Het principe hiervan wordt geïllustreerd in Figuur 1.6.

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 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 Sectie 1.1 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 aanwezig aan de uitgang van MALU n° 1.



Figuur 1.5: Logica voor besturing van de schakeling voor berekeningen in $\mathbb{F}_{2^m}$

Het resultaat van een vermenigvuldiging zal echter aan de uitgang van MALU n° 3 verschijnen, aangezien $163 \bmod 4 = 3$. Het eindresultaat dat in T dient opgeslagen te worden, is voor een vermenigvuldiging dus

$$T = mod3 \ \# \ T3_{162:1}$$

, terwijl dit voor een optelling

$$T = mod1 \ \# \ T1_{162:1}$$

is. Met andere woorden, er dient nu niet enkel gekozen te kunnen worden voor de ingang A, T_{out} of $T1_{\text{ready}}$, maar ook voor $T3_{\text{ready}}$.

Indien men toch wenst het vermenigvuldigen te versnellen, is het aangeraden een d te kiezen waarvoor

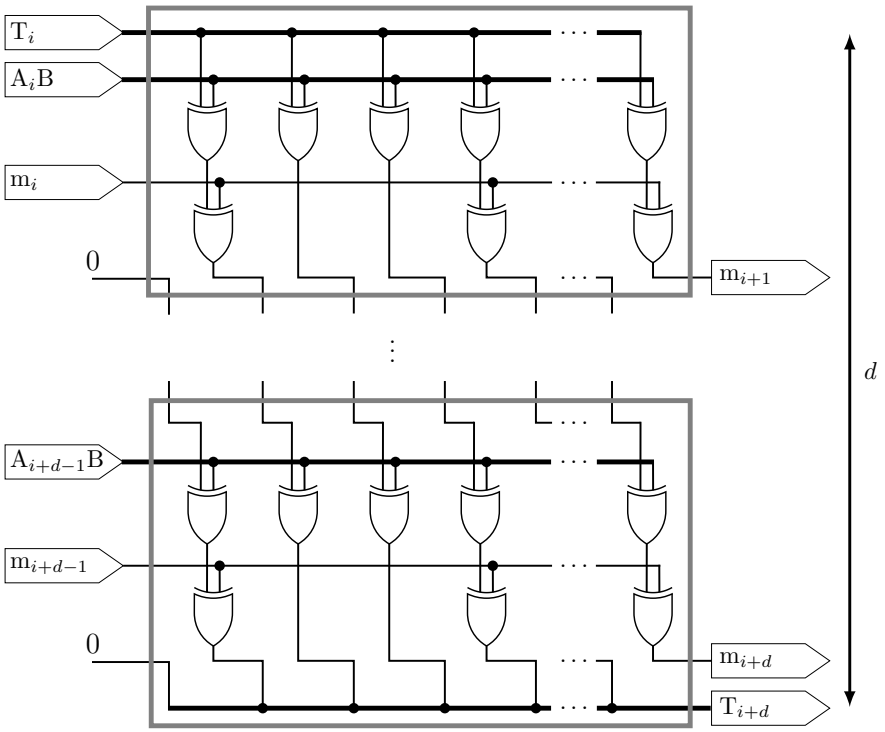
$$m \bmod d = 1$$

Als voorbeeld worden enkele voor de hand liggende en optimale keuzes vergeleken voor d indien $m = 163$ in Tabel 1.2.

Tabel 1.2: Voor de hand liggende versus optimale waarden voor woordbreedte d 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

Ideale waarden voor d						
d	2	3	6	9	18	27
$m \bmod d$	1	1	1	1	1	1



Figuur 1.6: Schakeling voor berekeningen in $\mathbb{F}_{2^m}$ met woordbreedte d

1.4 Controller voor het Miller algoritme

1.4.1 Inleiding

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 gekende parameters, zonder uitwerking van de berekeningen, wordt gegeven in Algoritme 1.3.

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

Om Algoritme 1.3 te kunnen implementeren, moeten eerst de verschillende berekeningen uitgewerkt worden. Vervolgens zal aan de hand van die uitwerkingen bepaald worden welke registers de uiteindelijke schakeling ten minste nodig heeft en ten slotte zal een FSM ontworpen worden.

1.4.2 Uitwerking berekeningen

Grofweg kan het algoritme opgedeeld worden in een verdubbelingsstap, een optellingstap, de vermenigvuldiging $F \cdot G$ en een finale exponentiatie. 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 voor de optellingstap een inversie in $\mathbb{F}_{2^m}$ uitgerekend moet worden. Uiteraard wordt ook dit verder onderzocht.

Algoritme 1.3: Miller algoritme voor berekening van de Tate pairing - Algemene versie

Input: $P, Q \in E(\mathbb{F}_{2^{163}})[l]$
Output: $\mathrm{e}(P, Q)$
Data: $V \in E(\mathbb{F}_{2^{163}})[l]$; $F, G \in \mathbb{F}_{2^{4 \cdot 163}}$

```
1  F ← 1
2  V ← P
3  for i ← 162 to 0 do
4  |   F ← F^2 · G_{V,V}(φ(Q))
5  |   V ← 2V
6  |   if i = 82 then
7  |   |   F ← F · G_{V,P}(φ(Q))
8  |   |   V ← V + P
9  F ← F^{(2^{4·163} − 1)/l}
10 return F
```

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 vorige schakeling hoog te maken bij de start van een berekening.

Verdubbelingsstap

De verdubbelingstap wordt gevormd door lijnen 4 en 5 in Algoritme 1.3. Voor een hyperelliptische kromme worden de berekeningen gegeven in Algoritme 1.4[1].

Algoritme 1.4: Verdubbelingstap voor hyperelliptische krommen in het Miller algoritme

Input: $x_V, y_V \in E(\mathbb{F}_{2^m})$
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 $\lambda \leftarrow x_V^2 + 1$	1 S
2 $x_{2V} \leftarrow \lambda^2$	1 S
3 $y_{2V} \leftarrow \lambda(x_{2V} + x_V) + y_V + 1$	2 A, 1 M
4 $G_{V,V}(\phi(Q)) \leftarrow \lambda(x_\phi + x_V) + (y_\phi + y_V)$	4 A, 1 M

Lijn 3 in dit algoritme kan ook berekend worden als

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

Aangezien dit echter 2 kwadrateringen en 1 optelling kost, wordt de voorkeur gegeven aan de eerste methode.

Door de specifieke vorm van $\phi(Q)$ kan lijn 4 uitgeschreven worden als

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

De variabele G kan dus opgeslagen worden in twee registers van grootte m in plaats van in vier.

Wanneer dit in rekening gebracht wordt en het algoritme op register niveau wordt uitgeschreven, bekomt men uiteindelijk Algoritme 1.5. Hierbij werd specifiek gelet op een minimum gebruik van tijdelijke registers.

Algoritme 1.5: Uitwerking van de verdubbelingstap voor hyperelliptische krommen in het Miller algoritme

Input: $x_V, y_V \in E(\mathbb{F}_{2^m})$
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 A, 1 M
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 A, 1 M
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

Buiten registers voor x_{2V}, y_{2V}, x_{ϕ_a}, y_{ϕ_a}, G_a en G_b is er dus ook nog een register nodig om λ in op te slaan.

Optellingstap

De optellingstap bestaat uit lijnen 7 en 8 van Algoritme 1.3. Voor een hyperelliptische kromme dienen de bewerkingen in Algoritme 1.6 uitgevoerd te worden[1].

Algoritme 1.6: Optellingstap voor hyperelliptische krommen in het Miller algoritme

Input: $x_V, y_V, x_P, y_P \in E(\mathbb{F}_{2^m})$
Output: $x_{V+P}, y_{V+P} \in E(\mathbb{F}_{2^m})$; $G \in \mathbb{F}_{2^{4m}}$
Data: $\lambda \in \mathbb{F}_{2^m}$

1	$\lambda \leftarrow \frac{y_V + y_P}{x_V + x_P}$	2 A, 1 I, 1 M
2	$x_{V+P} \leftarrow \lambda^2 + x_V + x_P$	2 A, 1 S
3	$y_{V+P} \leftarrow \lambda(x_{V+P} + x_P) + y_P + 1$	2 A, 1 M
4	$G_{V,V}(\phi(Q)) \leftarrow \lambda(x_\phi + x_P) + (y_\phi + y_P)$	4 A, 1 M

Net zoals bij de verdubbelingstap kan G hier in 2 variabelen opgeslagen worden. Hoewel de optellingstap 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 1.7. De meest tijdrovende stap hier is de inversie, waar in het volgende deel verder op ingegaan zal worden.

Algoritme 1.7: Uitwerking van de optellingstap voor hyperelliptische krommen in het Miller algoritme

Input: $x_V, y_V, x_P, y_P \in E(\mathbb{F}_{2^m})$
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 A, 1 I
3	$a \leftarrow G_b + y_P$; $\lambda \leftarrow \lambda \cdot a$	1 A, 1 M
4	$x_{V+P} \leftarrow \lambda^2 + G_a$; $x_{V+P} \leftarrow x_{V+P} + x_P$	2 A, 1 S
5	$y_{V+P} \leftarrow x_{V+P} + x_P$; $y_{V+P} \leftarrow y_{V+P} \cdot \lambda$	1 A, 1 M
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 A, 1 M
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

In tegenstelling tot de verdubbelingstap 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.

Inversie

Zoals reeds vermeld in Sectie **??**, kan een inversie in een Galois veld berekend worden door toepassing van Fermats kleine theorema:

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

De naieve manier om dit te berekenen zou zijn om a $2^m - 2$ keer met zichzelf te vermenigvuldigingen. In dit geval zou dat betekenen dat er $2^{163} - 2 = 11692013098647223345629478661730264157247460343806$ vermenigvuldigingen zouden moeten uitgevoerd worden. Zoiets is uiteraard onhaalbaar.

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. Dit gaat als volgt in zijn werk:

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

Als wordt aangenomen dat m oneven is, is de macht van twee na het gelijkheidsteken dus even. Zolang de macht van twee 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 eindigd men dan bij a^2 of a^3. Het totaal aantal bewerkingen is $m+1$ kwadrateringen en $\lceil \log_2(m) \rceil + 1$ vermenigvuldigingen.

In het geval van $m = 163$ is de uiteindelijke keten zoals gegeven in Algoritme 1.8.

Algoritme 1.8: 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

Het aantal berekeningen in dit geval is 162 kwadrateringen en 9 vermenigvuldigingen. Er is een register nodig om a bij de houden en twee voor de tussenresultaten a^{2^i-1} en $(a^{2^i-1})^{2^i}$.

Bibliografie

[1] G. Bertoni et al - Software Implementation of Tate Pairing over GF(2m). 2004.

[2] K. Sakiyama. *Secure Design Methodology and Implementation for Embedded Public-key Cryptosystems.* PhD thesis, Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, 3001 Heverlee, december 2007.