# 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 een smartcard. 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) [7][3]. 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)[1]

| 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:

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

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

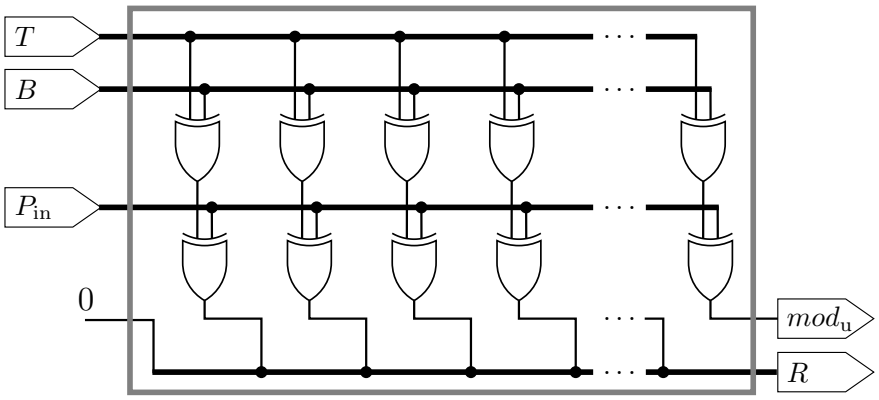
**Algoritme 1.1**: Modulo optelling in $\mathbb{F}_{2^m}$

**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
```

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. Een voor de hand liggende schakeling die dit alles implementeert, is te zien in Figuur 1.1. Ingang $P_{\text{in}}$ dient afhankelijk van de graad van $T$ ingesteld te worden op 0 of $P$. De ingangen $T$ en $P_{\text{in}}$ zijn $m$ bits aangezien het resultaat voor de vermenigvuldiging met $z$ steeds van graad $< m$ is en bit $m+1$ dus toch steeds 0 zou zijn. De hoogste graad term na de shift wordt naar buiten gebracht als $mod_{\text{u}}$. De implementatie bestaat 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 $P$ en de bijhorende $m$ XOR poorten kunnen vervangen worden door een 1 bit 'modulo enable' ingang $mod_{\text{e}}$ en er worden




Figuur 1.1: MALU - Basis ontwerp met shift

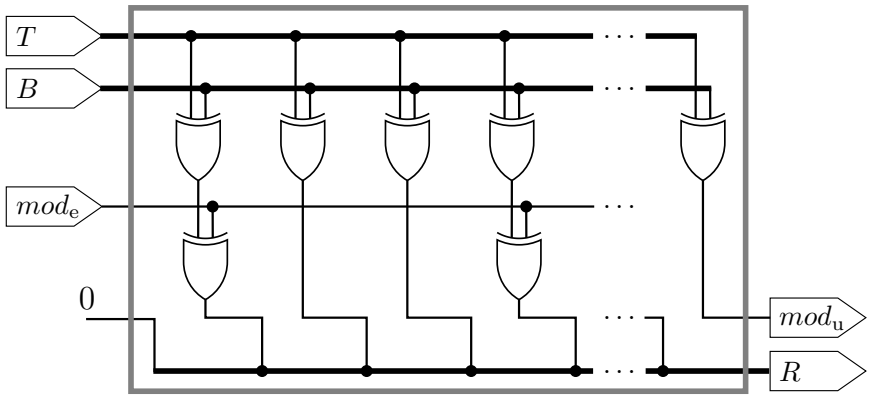
enkel XOR poorten geplaatst voor de bits $i$ waarvoor $P_i = 1$. Hierdoor wordt het aantal ingangen drastisch verkleind en worden

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

XOR poorten uitgespaard, met $\mathsf{Hamm}(P)$ 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 $\mathsf{Hamm}(P) - 1 = 4$ 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 1.2.




Figuur 1.2: 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.

**Algoritme 1.2**: "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
```

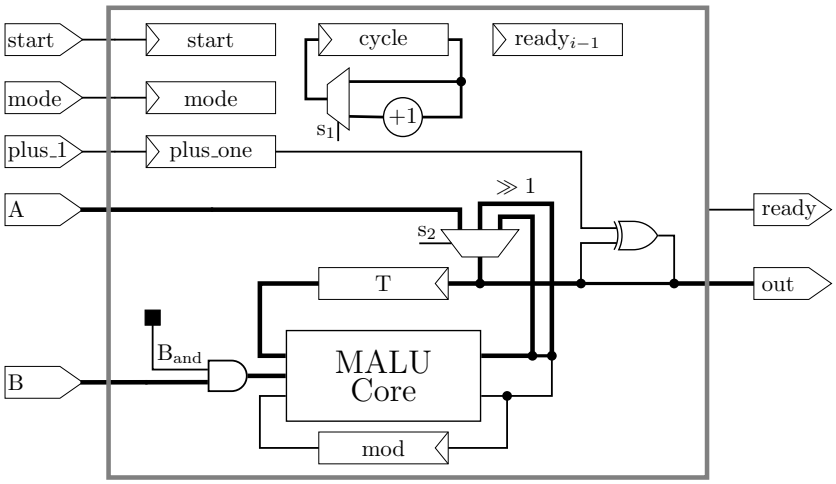
Wanneer de optelling en vermenigvuldiging nu in een schakeling gegoten worden, dient te schakeling te weten welke van de twee bewerkingen moet uitgevoerd worden. Verder moet het mogelijk zijn de uitkomt $R$ in het register $T$ op te slaan. Op die manier is het mogelijk de uitgang van de schakeling gelijk 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 . Er wordt zo veel mogelijk bespaard op registers. Het register *cycle* (equivalent aan $i$ in Algoritme 1.2) is $\lceil \log_2(m) \rceil$ bits lang en $T$ $m$ bits. De waarde van $T_m$ wordt opgeslagen in register *mod*. Alle overige registers zijn 1 bit groot. 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.

De uiteindelijke schakeling is te zien in Figuur 1.3. 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 1.4.

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 $R_{\text{ready}} = R \gg 1$.

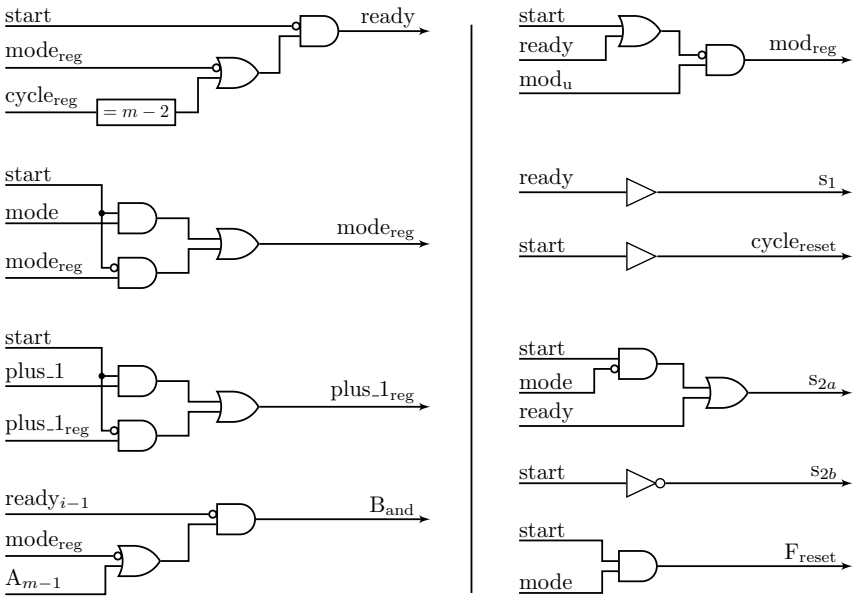



Figuur 1.3: Schakeling voor berekeningen in $\mathbb{F}_{2^m}$

### 1.3.2 Versnelling van de vermenigvuldiging

Wanneer met behulp van de schakeling in Figuur 1.3 een vermenigvuldiging wordt berekend, zal het $m$ klokcycles 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 klokcycle uit te voeren. Het principe hiervan wordt geïllustreerd in Figuur 1.5.

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 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.




Figuur 1.4: Logica voor besturing van de schakeling voor berekeningen in $\mathbb{F}_{2^m}$

Stel bijvoorbeeld $d = 4$ (en $m = 163$). Het resultaat van een optelling zal net zoals in het standaard ontwerp (Figuur 1.3) aanwezig aan de uitgang van MALU n° 1. Het resultaat van een vermenigvuldiging zal echter aan de uitgang van MALU n° 3 verschijnen, aangezien 163 mod 4 = 3. Het eindresultaat dat in $T$ 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

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

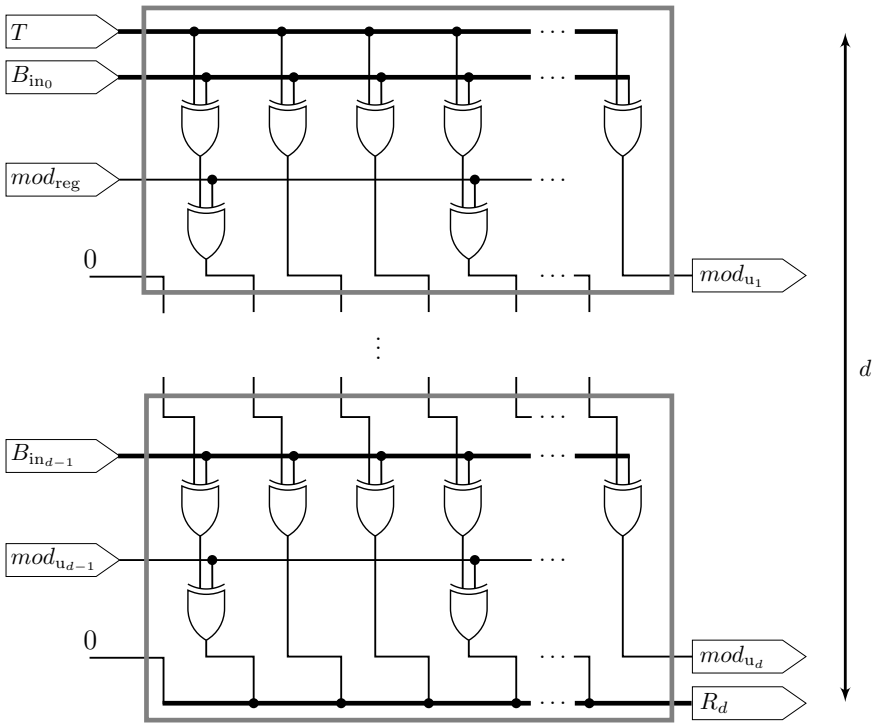
is. Met andere woorden, er dient nu niet enkel gekozen te kunnen worden tussen de ingangen $A$, $R_d$ of $R_{1_{\text{ready}}}$, maar ook voor $R_{3_{\text{ready}}}$.

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 1.2.

## 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.




Figuur 1.5: Schakeling voor berekeningen in $\mathbb{F}_{2^m}$ met woordbreedte $d$

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 |

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.

Grofweg kan het algoritme opgedeeld worden in de for-lus en een finale exponentiatie. De for-lus kan verder onderverdeeld worden in een verdubbelingsstap, een optellingstap, 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

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

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

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 schakeling voor berekeningen in $\mathbb{F}_{2^m}$ hoog te maken bij de start van een berekening.

### 1.4.2 For-lus

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

#### Verdubbelingsstap

De verdubbelingstap wordt gevormd door lijnen 4 en 5 in Algoritme 1.3. Voor een hyperelliptische kromme zijn de berekeningen als volgt[4]:

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

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:

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

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

Wanneer dit in rekening gebracht wordt en het algoritme op register niveau wordt uitgeschreven, bekomt men uiteindelijk Algoritme 1.4. Hierbij werd specifiek gelet op een minimum gebruik van tijdelijke registers.

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

**Algoritme 1.4**: 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 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 |

### Optellingstap

De optellingstap bestaat uit lijnen 7 en 8 van Algoritme 1.3. Voor een hyperelliptische kromme dienen de volgende bewerkingen uitgevoerd te worden[4]:

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

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 1.5.

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

**Algoritme 1.5**: 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 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 optellingstap is de inversie. Zoals reeds vermeld in Sectie **??**, kan een inversie in een Galois veld berekend worden door toepassing van Fermats kleine theorema:

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

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 [2][6]. 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 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 eindigd men dan bij $a^2$ of $a^3$. 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. $\mathsf{Hamm}(x)$ staat daarbij voor het Hamming gewicht van de binaire voorstelling van $x$.

In het geval van $m = 163$ is de uiteindelijke keten van bewerkingen zoals gegeven in Algoritme 1.6. 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 $(a^{2^i-1})^{2^i}$.

**Algoritme 1.6**: 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 |

### Kwadratering van $F$

Bij het uitvoeren van lijn 4 van Algoritme 1.3 moet ook telkens het kwadraat van $F$ berekend worden. De afleiding van de formule daarvoor gaat als volgt:

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

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 1.7. Eén kwadratering van $F$ vraagt vier optellingen en vier kwadrateringen in $\mathbb{F}_{2^m}$.

**Algoritme 1.7**: Uitwerking van de kwadratering van $F$ in het Miller algoritme

**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 |

**Vermenigvuldiging** $F \cdot G$

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

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

Indien hier nu de Karatsuba techniek 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 1.8 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 [5], 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 1.8**: 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 |

### 1.4.3 Finale exponentiatie

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 reductie op het einde van het Miller algoritme bestaat uit de exponentiatie $e(P, Q) = F^M$, met

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

De exponentiatie kan 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. De methode uit [5] wordt hier aangepast aan het gekozen extensieveld. 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

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

en

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

Er moeten dus vier termen in $\mathbb{F}_{2^{2m}}$ berekend worden. Merk verder op dat de noemers van alle breuken in zowel $V$ als $W$ identiek zijn. Er zal dus slechts één inversie in $\mathbb{F}_{2^{2m}}$ uitgerekend moeten worden. Ten slotte zijn de tweede termen van $V$ en $W$ identiek. Er moeten in totaal dus drie elementen in $\mathbb{F}_{2^{2m}}$ opgeslagen worden voor de uiteindelijke vermenigvuldiging van beide resultaten. Dit wil zeggen dat er dus zes registers van grootte $m$ nodig zijn.

### Termen van de deelbreuken

Ten eerste worden de drie te berekenen tellers $V_t$, $W_t$ en $G_t$ uitgewerkt. Hierbij staan $V_t$ en $W_t$ voor de teller van de eerste breuk van respectievelijk $V$ en $W$, $G_t$ staat voor de teller van de gemeenschappelijke tweede breuk. Met andere woorden:

$$\begin{aligned} V &= \frac{V_t}{G_n} + \left[\frac{G_t}{G_n}\right] y \\ W &= \frac{W_t}{G_n} + \left[\frac{G_t}{G_n}\right] y \end{aligned}$$

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 $A, B \in \mathbb{F}_{2^{2m}}$:

$$\begin{aligned} A \cdot B &= a_0 b_0 + a_0 b_1 x + a_1 b_0 x + a_1 b_1 x^2 \\ &= (a_0 b_0 + a_1 b_1) + (a_0 b_1 + a_1 b_0 + a_1 b_1) x \end{aligned}$$

Zodoende bekomt men:

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

Voor de gemeenschappelijke noemer $G_n$ vind men:

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

Een algoritme om deze vier resultaten uit te rekenen is te zien in Algoritme 1.9. Er zijn vier registers nodig voor $F$ en acht voor $V_t$, $W_t$, $G_t$ 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.

**Algoritme 1.9**: Uitwerking van berekening van noemers voor de finale exponentiate in het Miller algoritme

**Input**: $F = F_a + F_b x + F_c y + F_d xy \in \mathbb{F}_{2^{4m}}$
**Output**: $V_t = V_{t_a} + V_{t_b} x, W_t = W_{t_a} + W_{t_b} x, G_t = G_{t_a} + G_{t_b} x,$
$G_n = G_{n_a} + G_{n_b} x \in \mathbb{F}_{2^{2m}}$
**Data**: $a \in \mathbb{F}_{2^m}$

| | | |
|---|---|---|
| 1 | $V_{t_a} \leftarrow F_a^2$; $V_{t_b} \leftarrow F_b^2$; $G_{t_a} \leftarrow F_c^2$; $G_{t_b} \leftarrow F_d^2$ | 4 S |
| 2 | $V_{t_a} \leftarrow V_{t_a} + V_{t_b}$; $W_{t_b} \leftarrow V_{t_b} + G_{t_b}$ | 2 A |
| 3 | $G_{t_b} \leftarrow V_{t_b} + G_{t_b}$; $W_{t_a} \leftarrow V_{t_a} + G_{t_b}$ | 2 A |
| 4 | $V_{t_a} \leftarrow V_{t_a} + G_{t_a}$; $V_{t_b} \leftarrow V_{t_b} + G_{t_b}$ | 2 A |
| 5 | $G_{n_a} \leftarrow F_a + F_b$; $G_{n_b} \leftarrow T_a \cdot T_c$ | 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 | $G_{n_a} \leftarrow G_{n_a} + W_{t_a}$; $G_{n_b} \leftarrow G_{n_b} + W_{t_b}$ | 2 A |

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

Vervolgens moet de inverse van $G_n$ berekend worden, wat een inversie in $\mathbb{F}_{2^{2m}}$ is [5].

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

$$\begin{cases} A_a B_a + A_b B_b = 1 \\ A_a B_b + A_b B_a + B_b A_a = 0 \end{cases}$$

De oplossing van dit stelsel is

$$B_a = w^{-1} \cdot (A_a + A_b)$$
$$B_b = w^{-1} \cdot A_b$$

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 1.10, waarbij de oorspronkelijk $A$ 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 $c$.

**Algoritme 1.10**: Uitwerking van multiplicatieve inversie 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 I |
| 4 | $B_a \leftarrow a \cdot c$; $B_b \leftarrow A_b \cdot c$ | 2 M |

### Berekening van $V$ en $W$

Gewapend met al deze waarden is het mogelijk $V = V_0 + V_1 y$ en $W = W_0 + W_1 y$ te berekenen. In totaal zijn hier zes vermenigvuldigingen in $\mathbb{F}_{2^m}$ 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 $V_t$, $W_t$ als $G_t$ met $G_n$ vermenigvuldigd te worden. Vooropgesteld dat de tellers na vermenigvuldiging met $G_n$ niet meer nodig zijn en dus overschreven mogen worden, kan Algoritme 1.11 gebruikt worden. Er zijn dan zes registers nodig voor de uitkomsten, twee voor $G_n$ 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 1.11**: Uitwerking van vermenigvuldiging 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 |

# Bibliografie


[1] *0.13 $\mu m$ Platinum Standard Cell Databook*. Faraday, 2004.

[2] L. Batina. *Arithmetic And Architectures For Secure Hardware Implementations Of Public-Key Cryptography*. PhD thesis, KU Leuven, 2005.

[3] 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*, 4357:6–17, 2006.

[4] G. Bertoni, L. Breveglieri, P. Fragneto, G. Pelosi, and L. Sportiello. Software implementation of Tate pairing over GF(2m). In *DATE 06: Proceedings of the conference on Design, automation and test in Europe*, pages 7–11. European Design and Automation Association, 2006.

[5] 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*, volume 5209, pages 297–315. Springer, 2008.

[6] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in GF($2^m$) using normal bases. *Information and Computation*, 78(3):171–177, 1988.

[7] K. Sakiyama. *Secure Design Methodology and Implementation for Embedded Public-key Cryptosystems*. PhD thesis, KU Leuven, december 2007.