Radboud Universiteit Nijmegen

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Factoriseren met de getallenlichamenzeef

Naam: Elena Fuentes Bongenaar

Studie: Bachelor Wiskunde

Begeleider: Dr. W. Bosma

10 juli 2014

Inhoudsopgave

1 Samenvatting

Factoriseren van een groot getal is vrij moeilijk en juist deze eigenschap maakt het zeer geschikt om er gebruik van te maken in encryptiealgoritmen. Dit gebeurt bijvoorbeeld bij RSA. Het principe van Fermat kan helpen bij het factoriseren: wanneer er een paar bekend is waarvoor $a^2 \equiv b^2 \bmod n$ dan kan $\text{ggd}(a-b, n)$ een deler zijn. Diverse algoritmen maken gebruik van dit gegeven en hiervan zullen er een aantal in deze scriptie voorbij komen. De focus ligt op de getallenlichamenzeef: dit is momenteel het meest efficiënte algoritme voor getallen vanaf 130 cijfers. Deze zal uitgebreid worden besproken. Daarnaast zijn de kwadratische zeef en de getallenlichamenzeef geïmplementeerd in Magma; de code hiervoor is toegevoegd als bijlage.

2 Voorwoord

Deze scriptie vormt de afsluiting van mijn bachelor Wiskunde. Door mijn interesse in cryptografie en informatica heb ik geprobeerd te kiezen voor een onderwerp dat wiskunde met deze gebieden verbindt. Daarnaast had ik het doel om een onderwerp te kiezen dat is uit te leggen in een paar zinnen aan een niet-wiskundige; dit om simpelweg aan vrienden en familie te kunnen vertellen waar ik eigenlijk mee bezig was. Dit onderwerp heeft voor mij beide doelen vervuld.

Naar idee van Wieb Bosma heb ik de factorisatie algoritmen niet alleen bestudeerd, maar er ook 2 geïmplementeerd in Magma, een computer algebra systeem. Om dit systeem beter te leren kennen heb ik het vak Computer algebra gevolgd. Met erg veel plezier (en moeite) heb ik gewerkt aan de tekst en de code. Het was een grote uitdaging om een begrijpelijke tekst te schrijven en presentaties te maken die hopelijk te volgen waren. Als publiek heb ik altijd iemand zoals ikzelf voor ogen gehad: een derdejaars wiskunde student. Na mijn bachelor wil ik de master Computer Security doen en hoop later op dit gebied juist te kunnen bijdragen door mijn wiskunde achtergrond.

Ik wil mijn begeleider Wieb Bosma bedanken en dan in het bijzonder voor alles wat ik tot nu toe heb geleerd over Magma en het idee om het hierin te programmeren. Volgens mij heb ik nog maar een glimp opgevangen van de daadwerkelijke kracht van het gebruik van Magma voor dit soort toepassingen. Naar aanleiding van deze kennismaking met Magma ben ik van plan wiskundige algoritmes te blijven bestuderen en implementeren. Daarnaast wil ik Dries Schulten bedanken voor zijn steun tijdens het soms stressvolle traject en het zich beschikbaar stellen als oefenpubliek voor de presentaties en mijn uiteindelijke scriptie.

3 Introductie

3.1 RSA

In de cryptografie is factoriseren een belangrijk onderwerp. De veiligheid van verschillende encryptiemethoden berust op het feit dat het factoriseren van een groot samengesteld getal erg moeilijk is. RSA is zo'n cryptosysteem en kan gebruikt worden voor het versleutelen van handtekeningen of hele boodschappen.

Het versleutelingsalgoritme RSA [RRA78], vernoemd naar de bedenkers Rivest, Shamir en Adleman, is een public key systeem, wat betekent dat de encryptiesleutel openbaar is maar de decryptiesleutel niet. Het algoritme begint met het kiezen van grote priemgetallen p, q en de berekening van $n = p \cdot q$ die publiek bekend mag worden. De volgende stap is het bepalen van de publieke sleutel e en privé sleutel d. Hiervoor moet gelden: $e \cdot d \equiv 1 \bmod \phi(n)$ en $\text{ggd}(e, \phi(n)) = 1$, waarbij $\phi(n) = \{x : 1 < x < n, \text{ggd}(x, n) = 1\}$. De encryptie van een bericht m gaat dan als volgt:

$$E(m) = m^e \bmod n$$

De decryptie van een versleuteld bericht c gebeurt met de privé sleutel d:

$$D(c) = c^d \bmod n$$

Nu geldt $D(E(m)) = D(m^e) \equiv m^{e \cdot d} \bmod n$. Immers we weten dat $e \cdot d \equiv 1 \bmod \phi(n)$ en $\phi(n) = \phi(p) \cdot \phi(q) = (p-1) \cdot (q-1)$ dus $e \cdot d \equiv 1 \bmod (p-1) \cdot (q-1)$ [Keu11]. Uit Fermat's kleine stelling weten we dat voor een priemgetal p en een natuurlijk getal a, niet deelbaar door p, geldt: $a^p \equiv a \bmod p$ oftewel $a^{p-1} \equiv 1 \bmod p$. De puzzelstukjes vallen in elkaar wanneer we ons herinneren dat er geldt $n = p \cdot q$ met deze beide getallen priem. We krijgen dan $m^{e \cdot d} \equiv m^{k \cdot (p-1) \cdot (q-1)+1} \bmod n$ voor een $k \in \mathbb{Z}$ waardoor de ontsleuteling compleet is: $m^{k \cdot (p-1) \cdot (q-1)} \cdot m \equiv 1^k \cdot m \equiv m \bmod p \cdot q$.

In het hele proces mag iedereen n en e kennen. Uit deze getallen kan privé sleutel d worden bepaald met het uitgebreide Euclidische algoritme want voor een zekere $b \in \mathbb{Z}$ geldt $d \cdot e + b \cdot \phi(n) = \text{ggd}(e, \phi(n)) = 1$. Nu lijkt deze methode helemaal niet zo veilig! Maar het bepalen van $\phi(n)$ is niet makkelijk, behalve wanneer je kennis hebt van de ontbinding van n. We zagen namelijk dat $\phi(n) = \phi(p) \cdot \phi(q) = (p-1) \cdot (q-1) = p \cdot q - (p+q) + 1 = n - (p+q) + 1$. Andersom geldt dat $p + q$ bekend is wanneer je kennis hebt van $\phi(n)$. Het is duidelijk dat we p en q nodig hebben om d te weten te komen, waardoor de veiligheid van deze methode dus berust op het feit dat de ontbinding van n moeilijk te vinden wanneer deze bestaat uit twee 'grote' priemen.

A. Shamir schatte in 1999 [Sha99] dat zo'n 95% van alle handel waarbij gebruik werd gemaakt van computernetwerken was beveiligd met RSA en dan wel met een sleutel van 512 bit, wat een getal van ongeveer 154 cijfers is. Ter illustratie: bij een wedstrijd uitgeroepen door RSA laboratories [TKZ10] had

het grootst ontbonden getal 232 cijfers (768 bits) maar het heeft wel meer dan $2,5$ jaar gekost om dit voor elkaar te krijgen met meerdere computers. Het team dat deze factorisatie voor elkaar heeft gekregen, schatte dat het factoriseren van een sleutel van 1024 bits zo'n 1000 keer moeilijker is maar dacht wel dat dit mogelijk zou zijn voor het jaar 2020. Tegenwoordig wordt voor versleuteling met RSA een sleutel ter lengte 1024 of 2048 bits aangeraden [Lab14].

Het is natuurlijk niet onmogelijk om voor $n = p \cdot q$ de factorisatie te vinden. Voor gegeven n is de makkelijkste methode om een deler te vinden het simpelweg uitproberen van alle getallen tot aan $\lfloor \sqrt{n} \rfloor$. Een kleine optimalisatie is in dit geval aan te brengen door alleen naar de priemgetallen te kijken. Bij grote n kan een dergelijke methode extreem lang duren: een computer moet dan orde $\sqrt{(}n)$ stappen uitvoeren om dit voor elkaar te krijgen. Er moet dus een slimmere manier gevonden worden om delers te vinden. In deze scriptie worden enkele factorisatie algoritmen behandeld waaronder als laatst het momenteel snelste algoritme; de getallenlichamenzeef. Hiermee is ook het getal van 768 bits ontbonden.

4 Principe van Fermat

4.1 Fermat

Stap voor stap zullen er slimmere methodes en optimalisaties besproken worden om grote getallen te factoriseren, met als afsluiter het meest efficiënte algoritme vandaag de dag: de getallenlichamenzeef. De weg begint bij Fermats factorisatie methode.

Stel dat n het te factoriseren getal is. Fermat [Pom08] zag in dat het handig is n te schrijven als $n = a^2 - b^2$. Omschrijven geeft dan $n = a^2 - b^2 = (a+b)(a-b)$, waarbij $(a + b)$ en $(a - b)$ mogelijk niet-triviale delers zijn.

We kunnen dit als volgt omschrijven; er wordt gezocht naar waarden x waarvoor $f(x) = x^2 - n$ een kwadraat is. Begin het zoeken bij $a_1 = \lceil \sqrt{n} \rceil$ om direct positieve uitkomsten te vinden en ga steeds 1 stapje verder: $a_{i+1} = a_i + 1$. Het algoritme stopt wanneer er een a_i is gevonden waarvoor $f(a_i) = b^2$ voor een $b \in \mathbb{Z}$.

Fermat's algoritme
Input: samengestelde n
Output: niet triviale deler van n

1. Bereken het startpunt $x = \lceil \sqrt{n} \rceil$.
2. Bepaal $f(x) = x^2 - n$.
3. Is $f(x)$ een kwadraat?

(a) Ja: return $q = x - \sqrt{f(x)}$.

(b) Nee: laat $x = x + 1$ en ga weer naar stap 2.

Maar ook voor dit algoritme geldt weer dat voor oplopende n, het erg veel tijd zal kosten om een deler te vinden. Toch is dit principe het onderliggende idee voor alle volgende algoritmen en was dit inzicht van Fermat dus essentieel.

4.2 Kraitchiks verbetering

Een eerste stap om Fermats methode te optimaliseren is door niet alleen te zoeken naar getallen a en b waarvoor $a^2 - b^2$ gelijk is aan n, maar waarvoor het een veelvoud is van n [Pom96]. In dit geval hebben geldt dus $a^2 \equiv b^2 \bmod n$. We noemen dat a^2 en b^2 congruent modulo n. Nu gelden de volgende interessante implicaties:

$$a^2 \equiv b^2 \bmod n \Leftrightarrow a^2 - b^2 \equiv 0 \bmod n \Leftrightarrow n|(a^2 - b^2) \Leftrightarrow n|(a-b)(a+b)$$

Als $n = pq$ dan geldt dus $p|(a-b)(a+b)$ en $q|(a-b)(a+b)$. Er geldt als $p|(a-b)(a+b)$ en $\text{ggd}(p,(a+b)) = 1$ dat $p|(a-b)$ en zo natuurlijk ook voor q. We krijgen in ieder geval dat $\text{ggd}((a-b),n) \in \{1,p,q,n\}$. Als verder geldt dat $a \neq \pm b \ mod \ n$ hebben we dat $(a-b)$ en $(a+b)$ niet deelbaar zijn door n. In dit geval geldt $\text{ggd}(a-b,n) \in \{p,q\}$ en vinden we een niet triviale deler van n. De kans dat dit gebeurt is groter dan $1/2$ [Pom96]. We gaan dus op zoek naar paren waarvoor deze eigenschappen gelden. Het volgende voorbeeld laat zien dat deze aangepaste eigenschap zorgt voor een verandering in het opsporen van congruente paren (a,b).

Voorbeeld 4.2.1. Stel dat we $n = 1651$ willen ontbinden. Eerst bepalen we $\lceil\sqrt{1651}\rceil = 41$. Dan zien we na het berekenen van een aantal waarden $f(x) = x^2 - n$:

$$f(41) = 30$$

$$f(49) = 750$$

Deze functiewaarden zijn zelf geen kwadraten. Maar door vermenigvuldiging wordt er wel een kwadraat gevonden: $30 \cdot 750 = 22500 = 150^2$. Hierdoor vinden we $(41 \cdot 49)^2 \equiv 150^2 \bmod 1651$, oftewel $(a,b) = (41 \cdot 49, 150)$. Dan vinden we nu $\text{ggd}((41 \cdot 49) - 150, 1651) = \text{ggd}(1859, 1651) = 13$, dus we hebben de niet triviale deler 13 gevonden.

Fermat zou deze combinatie niet verder bekijken, immers $(41 \cdot 49)^2 - 150^2 = 4013581 \neq 1651$. Toch hebben we via deze weg een deler gevonden, wat laat zien dat de methode van Kraitchik meer paren in overweging kan nemen die tot de oplossing kunnen leiden.

We gaan dus op zoek naar een verzameling $\{x_1, \ldots, x_m\}$ waarvoor geldt dat $\prod_{i \in 1 \ldots m} f(x_i) = b^2$ voor een $b \in \mathbb{Z}$. Noem $a = \prod_{i \in 1 \ldots m} x_i$, dan krijgen we:

$$\begin{aligned} a^2 = x_1^2 \cdots x_m^2 &\equiv (x_1^2 - n) \cdots (x_m^2 - n) \bmod n \\ &= f(x_1) \cdots f(x_m) = b^2 \end{aligned}$$

Om te bepalen welke $f(x)$ geschikt zijn om een kwadraat mee te vormen, introduceren we de exponent vector.

Definitie 4.2.2. Voor een positief getal n met priemfactorisatie $n = \prod_{i \in \mathbb{N}} p_i^{a_i}$ definiëren we de *exponent vector* $v(n) = (a_1, a_2, \ldots)$.

Omdat voor eindig veel i geldt $a_i \neq 0$, kunnen we $v(n)$ schrijven als een eindige vector door de oneindige rij nullen aan het einde weg te laten. Verder is het enkel van belang of een macht even of oneven is; we zijn immers op zoek naar een product waarvan de exponent vector uitsluitend even getallen bevat. Daarom kunnen we de exponent vector modulo 2 bekijken, vanzelfsprekend geeft een kwadraat dan de nulvector. Aan het komende voorbeeld uit [Pom96] is te zien hoe we kunnen werken met deze exponent vectoren.

Voorbeeld 4.2.3. Neem nu als $n = 2041$. We berekenen eerst weer het startpunt $\lceil \sqrt{2041} \rceil = 46$. We krijgen na enkele berekeningen:

$$\begin{aligned} f(46) &= 75 & v(75) &= (0,1,2,0) \equiv (0,1,0,0) \bmod 2 \\ f(47) &= 168 & v(168) &= (3,1,0,1) \equiv (1,1,0,1) \bmod 2 \\ f(49) &= 360 & v(360) &= (3,2,1,0) \equiv (1,0,1,0) \bmod 2 \\ f(51) &= 560 & v(560) &= (4,0,1,1) \equiv (0,0,1,1) \bmod 2 \end{aligned}$$

De exponent vectoren hebben lengte 4 omdat de priemfactoren van de functiewaarden kleiner of gelijk aan 7 (het 4e priemgetal) zijn. Het optellen van de 4 exponentvectoren geeft de nulvector; hierdoor kunnen we inzien dat $75 \cdot 168 \cdot 360 \cdot 560 \bmod 2041$ een kwadraat moet zijn. We krijgen $(75 \cdot 168 \cdot 360 \cdot 560) = 50400^2 \equiv (46 \cdot 47 \cdot 49 \cdot 51)^2 \bmod 2041$, waardoor een congruent paar $(a, b) = (46 \cdot 47 \cdot 49 \cdot 51, 50400)$ gevonden is. Nu rest nog het berekenen van $\text{ggd}(46 \cdot 47 \cdot 49 \cdot 51 - 50400, 2041)$ waarvan we hopen dat het ons een niet-triviale deler geeft. Er geldt $\text{ggd}(46 \cdot 47 \cdot 49 \cdot 51 - 50400, 2041) = 13$, dus een geschikte deler is gevonden! En inderdaad: $2041 = 13 \cdot 157$.

In het algemeen kan een samenstelling van de nulvector gevonden worden wanneer het stelsel van exponent vectoren afhankelijk is. Dit wordt in ieder geval bereikt wanneer er meer exponentvectoren zijn dan het aantal posities in de exponentvectoren. Het kan natuurlijk ook voor komen dat het stelsel al eerder afhankelijk is, zoals in het voorbeeld hierboven. Door middel van Gauss eliminatie kan bepaald worden welke combinatie van vectoren de nulvector oplevert.

Algoritme van Kraitchik
Input: samengestelde n
Output: niet-triviale deler van n

1. Maak lege lijsten L en Q.
2. Bereken het startpunt $x = \lceil \sqrt{n} \rceil$.
3. Bepaal $f(x) = x^2 - n$ en voeg het paar $(x, f(x))$ toe aan Q.
4. Bepaal $v(x^2 - n) \bmod 2$ en voeg toe aan L.
5. Geeft L een afhankelijk stelsel?
   (a) Nee; laat $x = x + 1$ en ga naar punt 3.
   (b) Ja; laat $(x_i, y_i) \in R$ met R een deelverzameling Q die correspondeert met de geschikte exponentvectoren. Bepaal $(a, b) = (\prod_i x_i, \sqrt{(\prod_i y_i)})$. Geeft $\text{ggd}((a - b), n)$ een niet triviale deler?
      i. Ja: klaar en return deze deler.
      ii. Nee: laat $x = x + 1$ en ga naar punt 3.

4.3 Nog enkele aanpassingen

4.3.1 Factor -1 toevoegen

We hebben alleen gekeken naar positieve waarden $f(x)$ om een kwadraat mee te vormen. Het is mogelijk het algoritme van Kraitchik een beetje aan te passen door niet alleen te kijken naar de getallen $\lceil \sqrt{n} \rceil$ en hoger, maar ook waarden onder $\sqrt{n}$ toe te staan. Hierdoor verschijnen ook negatieve waarden $f(x)$. Dan is het nodig de exponent vector met 1 positie uit te breiden, want we kunnen de factor -1 immers niet negeren. Als we nu opnieuw naar het voorbeeld $n = 2041$ kijken met deze toegevoegde mogelijkheid zien we:

$$\begin{aligned} f(43) &= -192 & v(-192) &= (1,6,1,0) \equiv (1,0,1,0) \bmod 2 \\ f(45) &= -16 & v(-16) &= (1,4,0,0) \equiv (1,0,0,0) \bmod 2 \\ f(46) &= 75 & v(75) &= (0,0,1,2) \equiv (0,0,1,0) \bmod 2 \end{aligned}$$

Nu zijn er zelfs maar 3 exponent vectoren nodig om tot een geschikt paar te komen aangezien het optellen van de 3 vectoren de nulvector geeft. Dit paar is $(a, b) = (43 \cdot 45 \cdot 46, 480)$, waardoor we zien: $480^2 = 230400 = (192 \cdot 16 \cdot 75)$. Verdere berekening geeft wederom $\text{ggd}(43 \cdot 45 \cdot 46 - 480, 2041) = 13$.

4.3.2 Gladde waarden

We zagen al dat de exponentvector helpt bij het bepalen van een combinatie van waarden die een kwadraat oplevert. Het is duidelijk dat er vectoren nodig zijn van ongeveer gelijke lengte, oftewel: we zoeken $f(x)$ die uiteen vallen in priemfactoren onder een bepaalde grens y. Stel dat y het m-de priemgetal is. Hoeveel waarden $f(x)$ hebben we dan nodig om een kwadraat te vormen? Zoals eerder gezegd, moet er een afhankelijk stelsel vectoren gevormd worden van de exponent vectoren $f(x) \bmod 2$. Aangezien al deze vectoren lengte m hebben, zijn er hoogstens $m+1$ exponent vectoren van $f(x)$ nodig om een afhankelijk stelsel te krijgen. Als de exponentvector wordt uitgebreid met de factor -1 zoals hierboven beschreven, worden deze aantallen uiteraard met 1 verhoogd. Maar hoe bepalen we de grens y optimaal en hoe verkrijgen we vervolgens de juiste verzameling functiewaarden?

Definitie 4.3.1. Een niet lege verzameling priemgetallen $F = \{p : p \text{ priem}, p \leq y\}$ noemen we een *factorbasis*. Getallen die uiteenvallen over deze factorbasis noemen we *y-glad*.

In andere woorden zijn we op zoek naar een geschikte factorbasis die vervolgens gebruikt wordt om vast te stellen welke waarden $f(x)$ glad over deze basis zijn. Met deze gladde waarden wordt dan geprobeerd een kwadraat te maken. Wanneer de grens y te klein wordt gekozen, is de kans erg klein dat we snel $f(x)$ vinden die y-glad zijn. Tegelijk is het zaak y minimaal te houden omdat er dan minder exponent vectoren nodig zijn om een kwadraat te krijgen. In het algoritme van Dixon [Dix81] wordt deze extra verificatie op gladde waarden toegevoegd.

Dixon's algoritme
Input: samengestelde n
Output: niet-triviale deler van n

1. Bepaal factorbasis $F = \{p : p \text{ priem}, p \leq y\}$ en maak lijst L aan.
2. Kies r random tussen 1 en n en bereken $f(r) = r^2 \bmod n$.
3. Bepaal de priemfactorisatie $f(r)$. Is $f(r)$ y-glad?
   (a) Nee: ga naar stap 2.
   (b) Ja: voeg $f(r)$ toe aan L. Geldt $\#L > m$?
      i. Ja: ga naar stap 4.
      ii. Nee: ga naar stap 2.
4. Bepaal verzameling R waarvoor $\sum_{r \in R}(v(f(r)) \bmod 2)$ een niet triviale lineaire combinatie van de nulvector is.

5. Laat $(a, b) = (\prod_{r \in R} f(r), \prod_{r \in R} r^2)$.

6. Geldt $a \equiv -b \bmod n$?

   (a) Ja: ga terug naar stap 2.

   (b) Nee: return ggd$(a - b, n)$.

Stap 4 is uit te voeren omdat er eerst een stelsel $v(f(r))$ wordt gemaakt dat zeker afhankelijk is. Het zou dus kunnen zijn dat er al eerder een kwadraat gevormd kan worden, maar dan moeten er tussendoor extra checks uitgevoerd worden. Zo zou er na elke toegevoegde waarde $f(x)$ dat een kwadraat is, rijreductie uitgevoerd kunnen worden om te kijken of er al een afhankelijk stelsel is ontstaan. Het is ook mogelijk dat wanneer stap 6a bereikt wordt, er nog een ander kwadraat te vinden is in de reeds gemaakte lijst L. Aangezien dit niet zeker is, wordt er weer een nieuwe lijst gemaakt met een nieuwe kans op een geschikt paar (a, b) waar een deler uit voort kan komen.

5 Kwadratische zeef

5.1 Zeeftechniek

Wat opmerkelijk is aan het algoritme van Dixon is het *random* kiezen van $r \in [1, n]$ waarvan we vervolgens hopen dat $f(r)$ geschikte priemfactoren heeft; namelijk dat die allemaal vallen in gekozen factorbasis F. Door het random element kunnen er voor de looptijd slechts schattingen worden gegeven. Zou er een betere methode te vinden zijn dan r random kiezen? Carl Pomerance kwam inderdaad met een optimalisatie op dat punt toen hij de kwadratische zeef bedacht in 1981.

Naast dat het algoritme duidelijk voortbouwt op de eerder gezette stappen, dient de zeef van Eratosthenes ter inspiratie voor de kwadratische zeef. Dit simpele, eeuwenoude algoritme om priemgetallen onder een bepaalde grens te vinden werkt als volgt: neem een lijst getallen van 2 tot en met een gekozen bovengrens B. Laat het eerste getal staan en kruis alle veelvouden door. Bekijk nu het eerstvolgende niet doorgekruiste getal in de lijst, laat deze staan en kruis wederom alle veelvouden door. Herhaal dit tot het einde van de lijst. De priemgetallen zijn nu precies de getallen die niet zijn doorgekruist.

Wij zijn natuurlijk niet op zoek naar de priemen in een lijst, maar juist naar de getallen die uiteenvallen over een rij priemen. Dit uiteenvallen kunnen we herkennen aan hoe vaak een getal wordt doorgekruist. Als we bijvoorbeeld een lijst $[2 \ldots 10]$ nemen en het algoritme van Eratosthenes uitvoeren krijgen we na het doorkruisen $[2, 3, \not{4}, 5, \not{6}, 7, \not{8}, \not{9}, \not{10}]$. Zoals verwacht blijven de priemgetallen staan en verder zien we dat $4, 8$ en 9 éénmaal zijn doorgekruist, waar 6 en 10

tweemaal zijn doorgekruist. Dit algoritme toont dus het aantal verschillende delers van een getal, maar niet de macht van deze delers in de priemfactorisatie.

Laten we de tactiek nu iets aanpassen en in plaats van getallen doorkruisen, de getallen te delen door de priemgetallen. En steeds zo vaak als mogelijk, om een hogere priemmacht er uit te kunnen halen. Het lijstje $[2 \dots 10]$ zou dan worden: $[2, 3, 1, 5, 1, 7, 1, 1, 1]$. In dit geval blijven de priemgetallen natuurlijk staan en worden alle overige getallen 1, omdat ze wel uiteen moéten vallen over de voorgaande priemgetallen. Maar als we deze methode toepassen op een eigen gekozen interval van waarden $f(x)$ en dan delen door alle priemen uit de factorbasis, is snel te zien welke getallen uit het interval uiteenvallen over deze basis. Dit inzicht gaf aanleiding tot een algoritme dat veel sneller is dan zijn voorgangers.

5.2 Factorbasis

Zoals in de voorafgaande algoritmen gaan we op zoek naar een verzameling R waarvoor $\prod_{r \in R} f(r)$ een kwadraat is voor $f(r) = r^2 - n$. Hiervoor moet bepaald worden welke functiewaarden glad zijn over een gekozen factorbasis en de genoemde zeeftechniek kan hierbij helpen. Het is duidelijk dat er een aanpassing in deze techniek nodig zal zijn; het wegstrepen van de veelvouden zal niet opgaan omdat de lijst functiewaarden geen 'nette' oplopende lijst getallen is. Door de vorm van functie is er toch een makkelijke manier om te voorspellen welke waarden weggekruist moeten worden. Stel priemgetal p deelt een zekere $f(r) = r^2 - n$, dan deelt p ook elke $f(r + qp)$ voor $q \in \mathbb{Z}$, want

$$f(r + qp) = r^2 - 2pqr + p^2 + q^2 - n \equiv r^2 - n \equiv 0 \bmod p \ \Leftrightarrow p \mid f(r + pq)$$

Kies een vaste p en vind daarbij een r waarvoor geldt $r^2 \equiv n \bmod p$, vervolgens kun je als het ware 'voorspellen' welke $f(x)$ in het interval deelbaar zullen zijn door deze p. Het vinden van een dergelijke r houdt direct verband met de volgende definities.

Definitie 5.2.1. We noemen n een *kwadraatrest* modulo p als geldt $n \not\equiv 0 \bmod p$ en $\exists x \in \{1 \dots p-1\}$ met $x^2 \equiv n \bmod p$.

Definitie 5.2.2. Zij p priem en $n \in \mathbb{Z}$, dan is het *Legendresymbool* gedefiniëerd als: $\left(\frac{n}{p}\right) = \begin{cases} 0 & \text{als } p \mid n \\ 1 & \text{als } n \text{ een kwadraatrest modulo } p \text{ is.} \\ -1 & \text{anders} \end{cases}$

De priemfactorbasis moet dus bestaan uit priemgetallen p waarvoor het Legendresymbool gelijk is aan 1. Is dit niet het geval, dan is er geen oplossing voor $r^2 \equiv n \bmod p$. Bepaal voor elke $p_i \in F$ een r_i waarvoor $r_i^2 \equiv n \bmod p_i$. Vervolgens weten we dat elke $f(r)$ met $r = r_i + qp_i, q \in \mathbb{Z}$, deelbaar is door p_i en wordt deze zo vaak mogelijk hierdoor gedeeld. Wanneer dit voor alle priemgetallen in

de factorbasis wordt herhaald hebben alle gladde getallen de waarde 1 gekregen en kunnen deze gebruikt worden om een kwadraat mee te maken aan de hand van exponentvectoren.

5.3 Algoritme

Samen geeft dit het volgende algoritme:

Kwadratische zeef
Input: samengestelde n
Output: deler van n

1. Bepaal factorbasis $F = \{p : p \text{ priem}, p \leq y\}$ waarbij voor elke $p \in F$ geldt $\left(\frac{n}{p}\right) = 1$.
2. Bepaal B, de lengte van het interval waarover we verwachten te moeten zeven.
3. Maak een lijst L van functiewaarden $f(x_j)$ voor $x_1 = \lceil \sqrt{n} \rceil$ en $x_{j+1} = x_j + 1$, voor $1 \leq j \leq B$.
4. $\forall p \in F$: bepaal r waarvoor $r^2 \equiv n \bmod p$. Deel vervolgens in $f(x) \in L$ waarvoor $x = r + qp, q \in \mathbb{Z}$, zo vaak mogelijk door p.
5. Laat R bestaan uit alle x waarvan $f(x) \in L$ de waarde 1 heeft. Geeft R een afhankelijk stelsel van exponentvectoren?
   (a) Ja: maak een niet triviale lineaire combinatie $\sum_{x \in R}(v(f(x)) \bmod 2)$ van de nulvector.
   (b) Nee: verhoog B en ga terug naar stap 3 f verhoog y en ga naar stap 1.
6. Laat $(a, b) = (\sqrt{\prod_{x \in R} f(x)}, \prod_{x \in R} x)$ en return ggd$(a - b, n)$.

Een optimalisatie die bij de implementatie van de zeef toegepast zou kunnen worden is het opslaan van de logaritmen (grondgetal 2) van de functiewaarden [CP01], in plaats van de functiewaarden zelf. Vervolgens wordt voor elke p die deze functiewaarde deelt, $\lfloor \log(p) \rfloor$ eraf getrokken en dit voor alle p. Deze methode geeft een benadering, waardoor gladde waarden misschien niet helemaal netjes 0 zullen worden; hiervoor moet een marge bepaald worden.

6 Getallenlichamenzeef

Ook in het algoritme dat de naam getallenlichamenzeef draagt wordt er gezocht naar een paar (a, b) dat congruent is modulo n. In voorgaande algoritmen is beschreven dat er gezocht werd naar $x, y \in \mathbb{Z}$ waarvoor de kwadraten ervan congruent modulo n zijn, oftewel waarvoor $\phi(x^2, y^2)) = (x^2 \bmod n,\ x^2 \bmod n)$ voor het homomorfisme:

$$\phi : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$$

Verder willen we dat (x, y) niet valt in $\{(x, \pm x) : x \in \mathbb{Z}/n\mathbb{Z}\}$, zodat we een niet-triviale deler van n vinden. Merk op dat door de gebruikte methoden steeds maar één kwadraat gevonden hoefde te worden, en het andere kwadraat dan direct duidelijk was; deze kwam er als het ware gratis bij.

Bij de getallenlichamenzeef wordt het domein van ϕ aangepast naar $\mathbb{Z} \times \mathbb{Z}[\alpha]$ voor een zekere geschikte $\alpha \in \mathbb{C}$. Door het zoekdomein aan te passen hopen we sneller gladde waarden te vinden waar we vervolgens kwadraten uit kunnen samenstellen. Deze aanpassing zorgt ervoor dat het algoritme een stuk complexer wordt dan die tot nu toe besproken, maar ook voor een verkorte looptijd; de getallenlichamenzeef is momenteel het snelste algoritme voor getallen vanaf $110-130$ cijfers. Eerst zal een ruwe schets gegeven worden van hoe de getallenlichamenzeef in elkaar zit, daarna wordt er dieper op de werking en achtergronden ingegaan.

6.1 Ruwe schets

Zoals hierboven benoemd worden alle waarden om de kwadraten samen te stellen in $\mathbb{Z}$ gezocht. Kies een irreducibel monisch polynoom $f \in \mathbb{Z}[x]$ van graad $d > 1$. Bepaal hierbij een $m \in \mathbb{Z}$ waarvoor $f(m) \equiv 0 \bmod n$. Laat $\alpha \in \mathbb{C}$ een nulpunt van f zijn en

$$\mathbb{Z}[\alpha] = \{a_0 + a_1\alpha + \cdots + a_{d-1}\alpha^{d-1} : a_0, \ldots, a_{d-1} \in \mathbb{Z}\}$$

We hebben een natuurlijk homomorfisme $\theta : \mathbb{Z}[\alpha] \to \mathbb{Z}/n\mathbb{Z}$, geinduceerd door $\theta(1) = 1 \bmod n$, $\theta(\alpha) = m \bmod n$. Hiervoor zijn de eigenschappen $f(\alpha) = 0$ en $f(m) \equiv 0 \bmod n$ nodig. Dit hebben we nodig voor het volgende homomorfisme:

$$\begin{aligned} \phi : \mathbb{Z} \times \mathbb{Z}[\alpha] \ &\to\ \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \\ (z_1, z_2) &\mapsto (z_1 \bmod n,\ \theta(z_2)) \end{aligned}$$

Precies zoals bij de kwadratische zeef willen we nu een paar (c^2, γ^2) waarvoor $\phi(c^2, \gamma^2)$ een congruent paar modulo n geeft. Het verschil is nu dat $\gamma^2 \in \mathbb{Z}[\alpha]$, waardoor we nog wat extra werk moeten doen om dit om te zetten naar een congruent paar in $\mathbb{Z}$.
De strategie om $c^2 \in \mathbb{Z}$ en $\gamma^2 \in \mathbb{Z}[\alpha]$ te vinden is als volgt: vind een verzameling $R = \{(a, b) : a, b \in \mathbb{Z} \text{ en } \gcd(a, b) = 1\}$ waarvoor tegelijkertijd voor zekere c en γ het volgende geldt:

$$\prod_{(a,b)\in R} a - bm = c^2 \in \mathbb{Z}$$
$$\prod_{(a,b)\in R} a - b\alpha = \gamma^2 \in \mathbb{Z}[\alpha]$$

De eerste stap in het samenstellen van R gaat weer met een zeeftechniek; hiermee wordt bepaald welke waarden glad zijn. Omdat we nu over zowel $\mathbb{Z}$ en $\mathbb{Z}[\alpha]$ werken, hebben we te maken met twee factorbases en gebeurt het zeven ook over beide. Wanneer we voldoende gladde waarden hebben gevonden wordt er wederom gebruik gemaakt van exponentvectoren die ditmaal een stuk groter zijn doordat er meer factorbases zijn. Alle gladde waarden worden dus uitgeschreven in exponentvectoren modulo 2 en vervolgens wordt er gezocht naar een lineaire afhankelijkheid die ons een kwadraat zal geven. Noem nu $\theta(\gamma) = d \in \mathbb{Z}$. Dan geldt:

$$c^2 \equiv \theta(\gamma^2) \equiv \theta(\gamma)^2 \equiv d^2 \bmod n$$

Op dit punt hebben we dus weer twee congruente kwadraten gevonden en komt hier mogelijk een niet triviale deler uit.

In de komende paragrafen worden de details verder uit gewerkt. Allereerst wordt de keuze van f uitgelegd. Vervolgens wordt ingegaan op de zeeftechniek die uitgevoerd moet worden en de keuze voor de factorbases hierbij. De volgende stap is het vinden van de kwadraten met behulp van exponentvectoren die in een grote matrix worden opgeslagen. Tot slot wordt ingegaan op het bepalen van d wanneer γ^2 is gevonden.

6.2 Polynoom keuze

Het idee is om tegelijkertijd kwadraten te vinden in $\mathbb{Z}$ en $\mathbb{Z}[\alpha]$. Deze laatste is een uitbreiding van $\mathbb{Z}$, die we construeren met behulp van een zelf gekozen polynoom f. De keuze voor f die hier gegeven wordt, is gelijk aan die initieel door Buhler, Lenstra en Pomerance [JBP93] is voorgesteld toen zij een generalisatie van de speciale getallenlichamenzeef van Pollard gaven. (De speciale getallenlichamenzeef werkt, zoals de naam al suggereert, enkel voor getallen van een speciale vorm, namelijk: $n = b^c + 1$, waar b klein is.) De keuze voor het polynoom wordt gerechtvaardigd door de bewezen waarde in de praktische toepassing; dit polynoom werkt! Toch zijn er goed andere keuzes mogelijk waar hier niet verder op zal worden ingegaan, zie bijvoorbeeld [SBE].

Het polynoom f moet monisch en irreducibel zijn in $\mathbb{Q}[x]$. Eerst maken we de keuze voor $\deg f = d$, die we zetten op $d = 5$ voor getallen rond de 130 cijfers [CP01]. Vervolgens definieren we $m = \lfloor n^{1/d} \rfloor$ en schrijven we:

$$n = c_0 + c_1 m + \cdots + c_{d-1} m^{d-1} + m^d,$$

waarbij $c_i \in [0, \ldots, m-1]$. Nu hebben we ons polynoom gevonden:

$$f(x) = c_0 + c_1 x + \cdots + c_{d-1} x^{d-1} + x^d$$

Merk op dat het polynoom in $\mathbb{Q}[x]$ dus alleen gehele coëfficienten heeft. Door de manier waarop we het hebben samengesteld is dit polynoom per definitie monisch. Stel dat het polynoom niet irreducibel is, dan $f(x) = f_1(x)f_2(x)$. Maar dan is volgens [JBO81] de factorisatie van $n = f(m) = f_1(m)f_2(m)$ ook niet triviaal, en zijn we dus klaar met het uitvoeren van het algoritme! Er bestaan algoritmen die in polynomiale tijd verifiëren of f te factoriseren valt (bijvoorbeeld [ALL82]).

We weten dus dat f monisch en irreducibel is. Per constructie geldt ook: $f(m) = c_0 + c_1 m + \cdots + c_{d-1} m^{d-1} + m^d = n \equiv 0 \bmod n$. Nu de keuze voor het polynoom gemaakt is, gaan we kijken hoe het toegepast wordt in het algoritme. Uiteindelijk willen we terecht komen in een nog te definiëren ring $\mathbb{Z}[\alpha]$ en daarvoor wordt eerst het lichaam gemaakt waar deze ring in ligt. Houd voor de volgende definities [Cam08] in gedachten dat f een zeker nulpunt $\alpha \in \mathbb{C}$ heeft.

Definitie 6.2.1. Laat $\alpha \in \mathbb{C}$. Dan is α *algebraïsch* over $\mathbb{Q}$ als er een monisch polynoom $f \in \mathbb{Q}[x]$ ongelijk aan nul bestaat waarvoor $f(\alpha) = 0$. Als f in dit geval irreducibel is, noemen we deze het *minimumpolynoom*. Wanneer de coëfficienten van f bevat zijn in $\mathbb{Z}$, noemen we α een *algebraïsch geheel getal*.

Definitie 6.2.2. Voor $\alpha \in \mathbb{C}$ is de *lichaamsuitbreiding* $\mathbb{Q}(\alpha)$ van $\mathbb{Q}$ het kleinste deellichaam van $\mathbb{C}$ dat $\mathbb{Q}$ en α bevat.

Met de eerste definitie is in te zien dat het genoemde nulpunt α inderdaad algebraïsch is over $\mathbb{Q}$. Hierbij hoort het minimumpolynoom f. Met deze gegevens kunnen we meer te weten komen over $\mathbb{Q}[\alpha]$;

Stelling 6.2.3. *Laat K een deellichaam van het lichaam F. Zij $\alpha \in F$ algebraïsch over K en f het minimumpolynoom. Dan geldt:*

(i) $K(\alpha) \cong K[x]/fK[x]$;

(ii) $[K(\alpha) : K] = \deg f$;

(iii) $\{1, \alpha, \ldots, \alpha^{d-1}\}$ *is een basis voor* $K(\alpha)$ *als vectorruimte over* K

Bewijs. Voor een volledig bewijs zie [R.L97] p.33. Deel i volgt door te kijken naar het ringhomomorfisme $\tau : K[x] \to K(\alpha), f \mapsto f(\alpha)$ en gebruik te maken van $K[x]/\ker(\tau) \cong \mathrm{Im}(\tau)$. Voor iii is in te zien dat een element $\beta \in K(\alpha)$ te schrijven is als $g(\alpha) = f(\alpha)q(\alpha)+r(\alpha)$ voor $g, q, r \in K[x]$ met $\deg r < \deg f = d$. Omdat $f\alpha) = 0$ is dit gelijk aan $\beta = r(\alpha)$. Dus β is een lineaire combinatie van $1, \alpha, \ldots, \alpha^{d-1}$ met coëfficienten in K. Het is makkelijk in te zien dat dit stelsel afhankelijk is, vanwege de eigenschap dat het minimumpolynoom f elk ander polynoom $g \in K[x]$ deelt waarvoor α het nulpunt is ([Cam08] p.221).

Stel dat $g(\alpha) = 0$ dan is er een niet triviale lineaire combinatie van het genoemde stelsel gevonden. Maar omdat geldt $f|g$ en $degf = n > defg$ volgt er een tegenspraak. Dus dit stelsel in de basis voor $K(\alpha)$. Per definitie geldt nu $ii : [K(\alpha) : K] = d = \deg f$.
Er is nu nog een stapje te zetten waardoor we inzien dat we nu een getallenlichaam hebben [ST01].

Definitie 6.2.4. $\mathbb{Q}(\alpha)$ is een *getallenlichaam* $\Leftrightarrow [\mathbb{Q}(\alpha) : \mathbb{Q}]$ is eindig.

We hebben $[\mathbb{Q}(\alpha) : \mathbb{Q}] = \deg f = d$ en dus is $\mathbb{Q}(\alpha)$ een getallenlichaam. Verder geldt met deel *(iii)* van stelling 6.2.3 dat elk element $\beta \in \mathbb{Q}(\alpha)$ uniek geschreven kan worden als:

$$\beta = s_0 + s_1\alpha + \cdots + s_{d-1}\alpha^{d-1},$$

waarbij $s_0, \ldots, s_{d-1} \in \mathbb{Q}$. Optelling en vermenigvuldiging gaat zoals bij polynomen, waar gereduceerd wordt door de eigenschap $f(\alpha) = 0$. Aangezien in het algoritme wordt gezocht naar een verzameling $a - b\alpha$ met $a, b \in \mathbb{Z}$ is de ring die deze elementen bevat $\mathbb{Z}[\alpha] \subset \mathbb{Q}(\alpha)$. We gebruiken de notatie $s_0 + s_1\alpha + \cdots + s_{d-1}\alpha^{d-1}$ met $s_0, \ldots, s_{d-1} \in \mathbb{Z}$ voor elementen in $\mathbb{Z}[\alpha]$. De volgende stap is gladde getallen in $\mathbb{Z}$ en $\mathbb{Z}[\alpha]$ te vinden om kwadraten mee samen te kunnen stellen. Net zoals in de kwadratische zeef worden de gladde waarden gevonden met een zeeftechniek. Hier zijn factorbases voor nodig; deze worden in de volgende paragraaf toegelicht.

6.3 Factorbases

Het doel is een verzameling $R = \{(a, b) : a, b \in \mathbb{Z} \text{ en } \text{ggd}(a, b) = 1\}$ te vinden waarvoor tegelijkertijd voor zekere c en γ het volgende geldt:

$$\prod_{(a,b)\in R} a - bm = c^2 \in \mathbb{Z}$$

$$\prod_{(a,b)\in R} a - b\alpha = \gamma^2 \in \mathbb{Z}[\alpha]$$

Noem nu $\theta(\gamma) = r \in n\mathbb{Z}$, waarbij

$$\theta : \mathbb{Z}[\alpha] \to \mathbb{Z}/n\mathbb{Z}$$

$$\theta(s_0 + s_1\alpha + \cdots + s_{d-1}\alpha^{d-1}) \mapsto (s_0 + s_1m + \cdots + s_{d-1}m^{d-1}) \bmod n$$

Oftewel $\theta(\gamma)$ is de restklasse modulo n van γ na de substitutie $\alpha = m$. Er geldt $\theta(\gamma^2) = \theta(\prod_{(a,b)\in R} a - b\alpha) = (\prod_{(a,b)\in R} a - bm \bmod n) = (c^2 \bmod n)$ en dus:

$$c^2 \equiv \theta(\gamma^2) \equiv \theta(\gamma)^2 \equiv r^2 \bmod n$$

Zoals gezegd wordt er gebruik gemaakt van een zeeftechniek om gladde waarden te vinden waarmee vervolgens kwadraten worden samengesteld. Waar we eerder aan één factorbasis genoeg hadden, zullen we nu voor $\mathbb{Z}$ en $\mathbb{Z}[\alpha]$ een aparte basis nodig hebben en wordt er over beide simultaan gezeefd worden. In de volgende paragraaf zullen de keuzes voor deze bases toegelicht worden en daarna wordt ingegaan op hoe het zeven precies wordt gedaan.

6.3.1 Priemfactorbasis

Voor het zeven over $\mathbb{Z}$ zal gebruik gemaakt worden van een priemfactorbasis zoals dat ook het geval was bij de kwadratische zeef. Ditmaal wordt er een grens B gekozen en is de priemfactorbasis $F_1 = \{p : p \text{ priem en } p \leq B\}$ (merk op dat het Legendresymbool hier geen rol speelt). Het doel is om waarden van de vorm $a - bm$ te vinden die glad zijn over de factorbasis F_1. Er geldt $\forall p \in F_1 : p|a - bm \Leftrightarrow a = bm + kp$ voor een $k \in \mathbb{Z}$. Tijdens het zeven zal b vastgehouden worden en laten we a over een gekozen interval $[-M, M]$ lopen. Als er dan voor een a geldt $p|a - bm$, dan: $p|(a + qp) - bm\ \forall q \in \mathbb{Z}$ dus is direct duidelijk welke waarden deelbaar zijn door p. Het wordt hierdoor duidelijk dat de priemfactorbasis niet gefilterd hoeft te worden op Legendre symbool, zoals bij de kwadratische zeef.

6.3.2 Algebraïsche factorbasis

Het bepalen van de algebraïsche factorbasis is een stuk moeilijker dan de priemfactorbasis. Het doel is om met het zeven te bepalen welke elementen van $\mathbb{Z}[\alpha]$ 'glad' zijn, maar wat betekent dat eigenlijk? Voordat hier een uitspraak over gedaan kan worden moeten we meer te weten komen ontbinding van elementen van de vorm $a - b\alpha$ in $\mathbb{Z}[\alpha]$. We beginnen met de definitie van de norm van een element, die vaak inzicht kan verschaffen in de factorisatie van het element.

6.3.3 Norm van een element

Om de norm van een element in $\mathbb{Z}(\alpha)$ te kunnen definiëren hebben we eerste de volgende stelling nodig.

Stelling 6.3.1. *Zij $\mathbb{Q}(\alpha)$ een getallenlichaam, f minimum polynoom voor α en $[\mathbb{Q}(\alpha) : \mathbb{Q}] = d$. Dan zijn er precies d verschillende inbeddingen $\sigma_i : \mathbb{Q}(\alpha) \to \mathbb{C}$, $i \in \{1 \dots d\}$, waarbij $\sigma_i(\alpha) = \beta_i$ dan precies de nulpunten van f in $\mathbb{C}$ zijn.*

Bewijs. Volgens [ST01] Gevolg 1.3, heeft f precies d verschillende nulpunten die we zullen aangeven met $\beta_1, \dots, \beta_d$. Voor elke β_i is $\sigma_i : \mathbb{Q}(\alpha) \to \mathbb{Q}(\beta_i) \subset \mathbb{C}$ een isomorfisme geïnduceerd door $\sigma_i(1) = 1, \sigma_i(\alpha) = \beta_i$. We hebben dus minstens d inbeddingen van $\mathbb{Q}(\alpha)$ in $\mathbb{C}$.
Stel nu dat we nog een inbedding $\sigma : \mathbb{Q}(\alpha) \to \mathbb{C}$ hebben waarvoor geldt $\sigma(\alpha) = \beta \neq \beta_i\ \forall i$. Als $f(x) = s_0 + s_1 x + \cdots + s_d x^d$, dan hebben we:

$$\begin{aligned} f(\beta) &= s_0 + s_1\beta + \cdots + s_d\beta^d \\ &= s_0 + s_1\sigma(\alpha) + \cdots + s_d\sigma(\alpha)^d \\ &= \sigma(s_0 + s_1\alpha + \cdots + s_d\alpha^d) \\ &= \sigma(0) \\ &= 0 \end{aligned}$$

En zien we dat β toch een nulpunt van f was, wat in tegenspraak is met de aanname $\beta \neq \beta_i\ \forall i$.

Definitie 6.3.2. Laat $\alpha_1, \ldots, \alpha_d \in \mathbb{C}$ de nulpunten van f zijn, met $\alpha = \alpha_1$. De *norm* van een element $\beta = t_0 + t_1\alpha + \cdots + t_{d-1}\alpha^{d-1} \in \mathbb{Q}(\alpha)$ is dan [CP01]:

$$N(\beta) = \prod_{j\in\{1\ldots d\}} \sigma_j(\beta) = \prod_{j\in\{1\ldots d\}} t_0 + t_1\alpha_j + \cdots + t_{d-1}\alpha_j^{d-1} \in \mathbb{Q}$$

Het is duidelijk dat de norm multiplicatief is, wat i.h.b interessant is wanneer we naar kwadraten kijken. Stel $\gamma^2 \in \mathbb{Z}[\alpha]$, dan $N(\gamma^2) = N(\gamma)^2$. We zien dat een kwadraat in $\mathbb{Z}[\alpha]$ in ieder geval de eigenschap heeft dat de norm ervan ook een kwadraat is. Als we nu eens kijken naar de norm van elementen van de vorm $a - b\alpha, a, b \in \mathbb{Z}$:

$$N(a - b\alpha) = \prod_{j\in\{1\ldots d\}} a - b\alpha_j = (a - b\alpha_1)\cdots(a - b\alpha_d)$$

Als we dit verder gaan uitschrijven wordt duidelijk waarom het handig is om elementen van deze vorm te beschouwen:

$$\begin{aligned} N(a - b\alpha) &= (a - b\alpha_1)\cdots(a - b\alpha_d) \\ &= (b(a/b - \alpha_1))\cdots(b(a/b - \alpha_d)) \\ &= b^d(a/b - \alpha_1)\cdots(a/b - \alpha_d) \end{aligned}$$

Omdat $f(x) = (x - \alpha_1)\cdots(x - \alpha_d) \in \mathbb{C}[x]$, geldt:

$$N(a - b\alpha) = b^d f(a/b)$$

Door verder herschrijven wordt duidelijk dat deze norm in $\mathbb{Z}$ ligt (bedenk dat alle coëfficienten $c_i \in \mathbb{Z}$):

$$N(a - b\alpha) = a^d + c_{d-1}a^{d-1}b + \cdots + c_0 b^d = b^d f(a/b)$$

Aan de hand van deze norm definiëren we de functie $F(a, b) = a^d + c_{d-1}a^{d-1}b + \cdots + c_0 b^d$. We willen dus waarden $a - b\alpha$ samenstellen tot een kwadraat en we weten dat de norm van een kwadraat weer een kwadraat is én dat deze norm in $\mathbb{Z}$ zal liggen. Helaas is eerstgenoemde conditie niet voldoende om een kwadraat te vinden, anders kon uiteraard dezelfde zeeftechniek als in $\mathbb{Z}$ toegepast worden. Wanneer de norm een kwadraat is, hoeft het element zelf namelijk geen kwadraat te zijn zoals in het volgende voorbeeld te zien is.

Voorbeeld 6.3.3. Neem als polynoom $f(x) = x^2 + 1$ en bekijk het getallenlichaam $\mathbb{Q}(i)$. We weten dat $i, -i$ de nulpunten van f zijn dus we krijgen norm: $N(\beta) = \sigma_1(\beta)\sigma_2(\beta)$ voor $\sigma_j : \mathbb{Q}[i] \to \mathbb{C}, \sigma_1(i) = i$ en $\sigma_2(i) = -i$. Laten we nu kijken naar de elementen $8 + 6i$ en 10;

$$N(10) = 10^2 = 8^2 + 6^2 = N(8 + 6i)$$

Hier geldt dat $8+6i = (3+i)^2$ inderdaad een kwadraat is, maar $5 = (3-i)(3+i)$ niet, hoewel van beide elementen de norm een kwadraat is.

Het probleem is dat verschillende factoren van een element in $\mathbb{Z}[\alpha]$ zorgen voor gelijke priemfactoren in de norm van het element. Laten we eerst dieper ingaan op factorisatie in $\mathbb{Z}[\alpha]$.

6.3.4 Factoriseren in $\mathbb{Z}[\alpha]$

We zullen verderop zien dat priemgetallen in $\mathbb{Z}$ te relateren zijn aan de factoren van elementen in $\mathbb{Z}[\alpha]$, wat erg gunstig is voor de implementatie van het algoritme. Om dit te kunnen bekijken, herinneren we ons de volgende definities:

Definitie 6.3.4. Een ideaal van een ring R is een deelring S van R zodat $\forall s \in S$ en $\forall r \in R$ geldt: $rs, sr \in S$.

Definitie 6.3.5. Zij R een ring. We noemen een ideaal $\mathfrak{p} \neq R$ een priemideaal wanneer voor idealen $\mathfrak{b}, \mathfrak{c}$ van R met $\mathfrak{bc} \subseteq \mathfrak{p}$ geldt: $\mathfrak{b} \subseteq \mathfrak{p}$ of $\mathfrak{c} \subseteq \mathfrak{p}$.

We zeggen dat een priemideaal $\mathfrak{p}$ een element $r \in R$ deelt wanneer $r \in \mathfrak{p}$. Een priemideaal ligt over een uniek priemideaal van $\mathbb{Z}$: $\mathfrak{p} \cap \mathbb{Z} = p\mathbb{Z}$ voor zekere p priem [Ste05]. We willen te weten komen voor welke priemidealen $\mathfrak{p}_1, \ldots, \mathfrak{p}_\ell$ geldt $a - b\alpha \in \mathfrak{p}_i$. Om in te zien waarom factoriseren in $\mathbb{Z}[\alpha]$ niet zo makkelijk is als in bijvoorbeeld $\mathbb{Z}$ bekijken we het volgende:

Definitie 6.3.6. Een *integriteitsdomein* R is een commutatieve ring met identiteit zonder nuldelers.

Definitie 6.3.7. Zij R een integriteitsdomein. Dan is $r \in R$ een *eenheid* als $\exists b \in R$ zodat $rb = br = 1$. Verder is $p \in R$, ongelijk aan 0, *irreducibel* als het geen eenheid is en er geldt: als $p = xy \Rightarrow x$ of y is een eenheid.

Definitie 6.3.8. Zij R een integriteitsdomein. Dan is R een *uniek factorisatie domein* als geldt:

- elk element ongelijk aan een eenheid of nul is te ontbinden in irreducibele elementen.
- als $p_1 \cdots p_m = q_1 \cdots q_n$ waar p_i en q_j irreducibel zijn, dan geldt $m = n$ en zijn p_i en q_i gelijk na mogelijke herordening en op eventuele vermenigvuldiging met een eenheid na.

Er geldt dat $\mathbb{Z}$ een uniek factorisatie domein is, terwijl $\mathbb{Z}[\alpha]$ dat niet hoeft te zijn. Beschouw bijvoorbeeld de ring $\mathbb{Z}[\sqrt{-5}]$. Hier geldt $6 = 2 \cdot 3 = (1 + \sqrt{-5})(1 - \sqrt{-5})$. Al deze factoren zijn irreducibel en ook vermenigvuldiging met eenheden maakt weinig uit, aangezien de eenheden 1 en -1 zijn.

Elk priemideaal is irreducibel en aangezien we een verband gaan leggen tussen priemgetallen en priemidealen zou unieke factorisatie in priemidealen een gewenste eigenschap zijn van de ring waarin we gaan werken. Uit de volgende stelling volgt dat de ring van alle algebraïsche gehele getallen (zie definitie 6.2.1) in $\mathbb{Q}(\alpha)$ genaamd $\mathfrak{O}$ deze geschikte eigenschap heeft.

Stelling 6.3.9. *Elk ideaal ongelijk aan nul van $\mathfrak{O}$ kan op unieke wijze geschreven worden als product van priemidealen, op volgorde na.*

Bewijs. Zie [ST01] p.107 – 110.

Helaas hoeft het niet zo te zijn dat $\mathbb{Z}[\alpha] = \mathfrak{O}$, zoals we kunnen zien aan het volgende voorbeeld.

Voorbeeld 6.3.10. Zij $f(x) = x^2 - 13$. Dit polynoom is monisch en irreducibel waardoor we de lichaamsuitbreiding $\mathbb{Q}(\sqrt{13})$ krijgen. Deze bevat onder andere het element $\beta = \frac{1+\sqrt{13}}{2}$. Het is duidelijk dat $\beta \notin \mathbb{Z}[\sqrt{13}]$. Verder is β een nulpunt van het irreducibele polynoom $g(x) = x^2 - x - 3$ en is dus een algebraïsch geheel getal. Dus in dit geval geldt: $\mathbb{Z}[\sqrt{13}] \neq \mathfrak{O}$.

Laten we voor nu echter even aannemen dat dit wél het geval is zodat we gebruik kunnen maken van de eigenschap die genoemd wordt in stelling 6.3.9; unieke factorisatie van idealen in $\mathfrak{O}$. Later zullen we kijken wat er moet gebeuren wanneer deze aanname niet waar is.

6.3.5 Factoriseren in $\mathfrak{O}$

We nemen vanaf nu dus aan dat $\mathbb{Z}[\alpha] = \mathfrak{O}$. Het idee is om ons te concentreren op de ontbinding van het ideaal voortgebracht door $a - b\alpha$: $\langle a - b\alpha \rangle \subset \mathfrak{O}$. Als geldt $\langle a - b\alpha \rangle = \mathfrak{p}_1 \cdots \mathfrak{p}_k$, dan hebben we $a - b\alpha \in \mathfrak{p}_i \ \forall i \in \{1 \dots k\}$. Om meer te weten te komen over de factorisatie van $\langle a - b\alpha \rangle$ wordt er weer gebruik gemaakt van de norm, ditmaal die gedefinieerd voor idealen. De norm van een ideaal in $\mathfrak{O}$ kan gedefinieerd worden aan de hand van de volgende stelling.

Stelling 6.3.11. *Voor elk ideaal $\mathfrak{I} \subset \mathfrak{O}$, ongelijk aan nul, is $|\mathfrak{O}/\mathfrak{I}|$ eindig.*

Bewijs. Dit bewijs is te vinden in [ST01] Stelling 5.3. Een belangrijke stap is concluderen dat $\mathfrak{O}/\mathfrak{I} \subset \mathfrak{O}/\langle N(\beta) \rangle$ voor $\beta \in \mathfrak{I}$ ongelijk aan nul. Er geldt namelijk:

$$N(\beta) = \sigma_1(\beta) \cdots \sigma_d(\beta) = \beta_1 \cdots \beta_d$$

Dit element $N(\beta)$ ligt in $\mathfrak{I}$ omdat voor een $i \in \{1 \dots d\} : \beta = \beta_i$ en $\mathfrak{I}$ een ideaal is. Maar dan geldt ook $\langle N(\beta) \rangle \subset \mathfrak{I}$ waardoor de inclusie volgt.

Definitie 6.3.12. Zij $\mathfrak{I}$ een ideaal ongelijk aan nul in $\mathfrak{O}$. Dan is de *norm* van dit ideaal gelijk aan: $\mathcal{N}(\mathfrak{I}) = |\mathfrak{O}/\mathfrak{I}|$. Verder is $\mathcal{N}(\langle 0 \rangle) = 0$.

Ook de norm voor idealen in $\mathfrak{O}$ is multiplicatief [ST01]. Er is nu een verband te leggen tussen priemgetallen in $\mathbb{Z}$ en priemidealen in $\mathfrak{O}$, met behulp van de volgende stelling:

Stelling 6.3.13. *Zij $\mathfrak{p}$ een niet nul ideaal van $\mathfrak{O}$, dan geldt:*

1. *Als $\mathcal{N}(\mathfrak{p}) = p$ voor een priemgetal $p \in \mathbb{Z}$, dan is $\mathfrak{p}$ een priemideaal.*
2. *Als $\mathfrak{p}$ een priemideaal is, bevat het precies één priemgetal $p \in \mathbb{Z}$ en geldt $\mathcal{N}(\mathfrak{p}) = p^e$ voor $e \leq d$ geheel getal. Dan noemen we $\mathfrak{p}$ een e-de graads priemideaal.*

Bewijs: Wegens multipliciteit van de norm volgt deel 1. Voor deel 2 gebruiken we dat geldt $\mathcal{N}(\mathfrak{p}) \in \mathfrak{p}$ uit [ST01] 5.14(b) en $\mathcal{N}(\mathfrak{p}) = p_1^{f_1} \cdots p_m^{f_m}$. Stel dat twee verschillende priemgetallen p en q bevat zijn in $\mathfrak{p}$, dan geldt ook voor zekere $a, b \in \mathbb{Z}$: $ap + bq = 1 \in \mathfrak{p}$ en dus $\mathfrak{p} = \mathfrak{O}$. Dus moet gelden dat $\mathfrak{p} = p_i^{f_i}$ voor zekere p_i priem.

Een ideaal voortgebracht door een enkel element noemen we een *hoofdideaal*. Er bestaat een handige correspondentie tussen de norm van een element $\beta \in \mathfrak{O}$ en de norm van het hoofdideaal $\langle\beta\rangle$ dat erdoor voortgebracht wordt.

Propositie 6.3.14. Voor $\beta \in \mathfrak{O}$ geldt $|N(\beta)| = \mathcal{N}(\langle\beta\rangle)$.

Bewijs: [ST01] p.116.

Dit verband komt uitstekend van pas omdat we zagen dat de norm van een element van de vorm $a - b\alpha$ te bepalen is door $F(a, b)$ uit te rekenen. We krijgen dus:

$$|F(a,b)| = |N(a - b\alpha)| = \mathcal{N}(\langle a - b\alpha\rangle)$$

We weten door stelling 6.3.9 dat voor zekere priemidealen $\mathfrak{p}_1, \ldots, \mathfrak{p}_m \subset \mathfrak{O}$ en $f_1, \ldots, f_m \in \mathbb{Z}$ geldt:

$$\langle a - b\alpha\rangle = \mathfrak{p}_1^{f_1} \cdots \mathfrak{p}_m^{f_m}$$

Dit kunnen we samenvoegen tot:

$$p_1^{\mathrm{ord_{p_1}(N(a-b\alpha))}} \cdots p_k^{\mathrm{ord_{p_k}(N(a-b\alpha))}} = |N(a-b\alpha)| = \mathcal{N}(\langle a-b\alpha\rangle) = \mathcal{N}(\mathfrak{p}_1^{f_1} \cdots \mathfrak{p}_m^{f_m})$$

Hierbij zijn p_i priemgetallen in $\mathbb{Z}$ en is $\mathrm{ord_p(N(\beta))}$ gelijk aan het aantal factoren van p in β. Vanwege unieke factorisatie in $\mathfrak{O}$ is $\langle a - b\alpha\rangle$ een kwadraat $\Leftrightarrow f_i \equiv 0 \bmod 2\ \forall i \in \{1 \ldots m\}$. Is nu uit $|N(a - b\alpha)|$ af te leiden wanneer dit het geval is? Helaas is de conditie $\mathrm{ord_{p_i}(N(a - b\alpha))} \equiv 0 \bmod 2\ \forall \mathrm{i} \in \{1 \ldots \mathrm{k}\}$ niet voldoende omdat er kan gelden:

1. priem $p \in \mathbb{Z}$ is bevat in meerdere priemidealen $\mathfrak{p}$
2. $\mathfrak{p}$ hoeft geen eerstegraads priemideaal te zijn, waardoor $\mathcal{N}(\mathfrak{p}) = p^s, s > 1$.

Gelukkig hebben de elementen van de vorm $a - b\alpha$ een handige eigenschap; zij worden alleen gedeeld door eerstegraads idealen. Stel namelijk dat $\mathfrak{p}$ een priemideaal is dat ligt over p, dus $\mathfrak{p} \cap \mathbb{Z} = p\mathbb{Z}$. Stel $b \in \mathfrak{p}$, dan zou ook gelden $a \in \mathfrak{p}$. Dat kan niet omdat we paren hebben waarbij $\mathrm{ggd}(a, b) = 1$. Nu kunnen we een paar definiëren (p, r) wat correspondeert met een eerstegraads priemideaal, wat gelijk is aan $\mathfrak{p}$, op de volgende wijze:

Stelling 6.3.15. *Zij $p \in \mathbb{Z}$ een priemgetal en laat*

$$R(p) = \{r \in \mathbb{Z}/p\mathbb{Z} : f(r) \equiv 0 \bmod p\}$$

Dan bestaat er een bijectie tussen paren (p, r) met $r \in R(p)$ en eerstegraads priemidealen in $\mathfrak{O}$.

Bewijs. Zie voor dit bewijs ook [JBP93] p.59 en [Ste05] p.90. We nemen nog steeds aan dat $\mathfrak{O} = \mathbb{Z}[\alpha]$. We maken gebruik van een homomorfisme $\tau : \mathbb{Z}[\alpha] \to \mathbb{Z}/p\mathbb{Z} = F_p$. Een eerstegraads priemideaal $\mathfrak{p}$ is dan precies de kern van τ voor een zekere priem p. Er geldt dat τ wordt voortgebracht door $\tau(1) = 1, \tau(\alpha) = r$ voor een zekere r. Deze combinatie (p, r) legt τ dan vast. Er geldt voor deze r:

$$\begin{aligned} f(r) &= c_0 + c_1 r + \cdots + c_{d-1} r^{d-1} + r^d \\ &= c_0 + c_1 \tau(\alpha) + \cdots + c_{d-1} \tau(\alpha)^{d-1} + \tau(\alpha)^d \\ &= \tau(c_0 + c_1 \alpha + \cdots + c_{d-1} \alpha^{d-1} + \alpha^d) \\ &= \tau(f(\alpha)) = \tau(0) \equiv 0 \bmod p \end{aligned}$$

Wat laat zien dat $r \in R(p)$.

Zoals in bovenstaand bewijs gaan we een homomorfisme $\tau : \mathbb{Z}[\alpha] \to F_p$ definiëren waarvan de kern $\mathfrak{p}$ moet zijn. Dan hebben we namelijk laten zien dat $\mathfrak{p}$ een eerstegraads priemideaal is. We hebben dus $a - b\alpha \in \mathfrak{p}$ en de correspondentie van $\mathfrak{p}$ en $p\mathbb{Z}$ voor een p. Er geldt dus $\overline{a} = \overline{b\alpha} \in \mathbb{Z}[\alpha]/\mathfrak{p}$ waardoor we kunnen zien dat $\overline{\alpha} = ab^{-1} \bmod p$. Definieer nu: $\tau(\alpha) = r = ab^{-1} \bmod p$, er geldt voor r: $f(r) \equiv 0 \bmod p$. Verder geldt dat $\ker(\tau) = p\mathbb{Z}[\alpha] + (a - b\alpha)\mathbb{Z}[\alpha] = \mathfrak{p}$ waardoor we een correspondentie hebben gevonden tussen het paar (p, r) en in te zien is dat de elementen $a - b\alpha$ enkel gedeeld worden door eerstegraads priemidealen. Hier gaan we ons nu op focussen. Voor beide genoemde problemen is nu dus een oplossing gevonden door de unieke identificatie van een priemideaal. De exponent van een priemideaal kunnen we met behulp van de volgende definitie bepalen.

Definitie 6.3.16. Zij $p \in \mathbb{Z}$ een priemgetal en laat $r \in R(p)$ zoals hierboven. Definieer de exponent $e_{p,r}(a - b\alpha) = \begin{cases} \mathrm{ord_p(N(a - b\alpha))} & \text{als } a \equiv br \bmod p \\ 0 & \text{anders} \end{cases}$

Als $\mathfrak{p}$ correspondeert met (p, r) dan is $e_{p,r}(a - b\alpha)$ dus gelijk aan de exponent van $\mathfrak{p}$ in $\langle a - b\alpha \rangle$. We kunnen de exponent opvatten als een homomorfisme $e_{p,r} : \mathfrak{O} \to \mathbb{Z}$. Dan zien we dat de volgende eigenschappen gelden $\forall \beta \in \mathfrak{O}$:

1. $e_{p,r}(\beta) \geq 0$
2. Als $\mathfrak{p}$ correspondeert met (p, r) dan geldt: $e_{p,r}(\beta) > 0 \Leftrightarrow \mathfrak{p} | \langle \beta \rangle$
3. $e_{p,r}(\beta) \neq 0$ voor eindig veel (p, r). Laat alle $\mathfrak{p}$ corresponderen met (p, r) dan geldt:
$$|N(\beta)| = \prod_{\mathfrak{p}} \mathcal{N}(\mathfrak{p})^{e_{p,r}(\beta)}$$

Nu hebben we een precieze correspondentie tussen de norm van een element in $\mathfrak{O}$ en de factorisatie van het ideaal voortgebracht door dit element in priemidealen in $\mathfrak{O}$. Laat elke $\mathfrak{p}$ in bijectie staan met een (p, r) dan geldt:

$$\prod_p p^{e_{p,r}(a-b\alpha)} = |N(a-b\alpha)| = \mathcal{N}(\langle a-b\alpha\rangle) = \prod_{\mathfrak{p}} \mathcal{N}(\mathfrak{p}^f)$$

Nu kan er gedefinieerd worden wanneer een element in $\mathfrak{O}$ glad is, zoals er ook een definitie voor bestaat in $\mathbb{Z}$, en kan er gekeken worden hoe er bepaald wordt wanneer een element glad is.

6.3.6 Gladde waarden

Voor gladde waarden hanteren we de volgende definitie:

Definitie 6.3.17. $\beta \in \mathfrak{O}$ noemen we *glad* over de algebraïsche factorbasis $\mathfrak{F} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_s\}$ als $\langle\beta\rangle$ uiteenvalt over de priemidealen in $\mathfrak{F}$.

Dit kan nu omgeschreven worden naar een meer werkbare situatie voor elementen van de vorm $a - b\alpha$. We weten immers dat elke $\mathfrak{p} \in \mathfrak{O}$ correspondeert met een uniek paar (p, r). Neem nu als nieuwe algebraïsche factorbasis F_2 deze paren (p, r). Stel dat er een bijectie bestaat tussen het specifieke priemideaal $\mathfrak{p}$ en het paar (p, r), dan geldt voor $a - b\alpha$:

$$\mathfrak{p} | \langle a - b\alpha\rangle \Leftrightarrow a \equiv br \bmod p$$

Als $a-b\alpha$ glad is over $\mathfrak{F} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_s\}$ dan geldt $\forall (p, r) \in F_2 : a \equiv br \bmod p$. Hier zien we dat de norm alle informatie geeft die nodig is, mits er gebruik gemaakt wordt van de 'nieuw' gedefinieerde exponent. Dit wordt samengevat in de volgende definitie.

Definitie 6.3.18. $a - b\alpha \in \mathfrak{O}$ noemen we *glad* over de algebraïsche factorbasis F_2 als $N(a - b\alpha)$ B-glad is, waarbij B het grootste priemgetal in F_2 is en voor de ontbinding van de norm gebruik wordt gemaakt van de exponent zoals gedefinieerd in definitie 6.3.16.

Het is nu eenvoudig de zeeftechniek die gebruikt zal worden te omschrijven. Laat de algebraïsche factorbasis $F_2 = \{(p, r) : p \leq B, r \in R(p)\}$, waarbij B dezelfde grens is als bij de priemfactorbasis F_1 zoals eerder beschreven. Het doel is om waarden van de vorm $a - b\alpha$ te vinden die glad zijn over de factorbasis F_2. Tijdens het zeven zal b vastgehouden worden en laten we a over een gekozen interval $[-M, M]$ lopen. Als er dan voor een a geldt $a \equiv br \bmod p$, dan: $(a + qp) \equiv br \bmod p$ $\forall q \in \mathbb{Z}$ dus is direct duidelijk welke waarden 'deelbaar' zijn door (p, r).

Ondanks dat we nu de zeeftechniek hebben uitgerold, zijn we nog niet klaar. Er is namelijk uitgegaan van de aanname $\mathbb{Z}[\alpha] = \mathfrak{O}$, waarvan al was aangetoond dat dit niet altijd het geval is. Daarnaast is nog niet duidelijk hoe we nu een kwadraat in $\mathbb{Z}[\alpha]$ vinden.

6.3.7 Oplossingen voor $\mathbb{Z}[\alpha] \neq \mathfrak{O}$

Om de situatie te generaliseren naar het geval dat $\mathbb{Z}[\alpha] \neq \mathfrak{O}$ hebben we de volgende propositie nodig:

Propositie 6.3.19. Voor elk priemideaal $\mathfrak{p} \in \mathbb{Z}[\alpha]$ bestaat er een groepshomormorfisme $\mathfrak{l}_\mathfrak{p} : \mathbb{Q}(\alpha)^* \to \mathbb{Z}$ zodat geldt:

1. $\forall \beta \in \mathbb{Z}[\alpha], \beta \neq 0, : \mathfrak{l}_\mathfrak{p}(\beta) \geq 0$
2. $\forall \beta \in \mathbb{Z}[\alpha], \beta \neq 0$ geldt: $\mathfrak{l}_\mathfrak{p}(\beta) > 0 \Leftrightarrow \beta \in \mathfrak{p}$
3. $\forall \gamma \in \mathbb{Q}(\alpha)^* : \mathfrak{l}_\mathfrak{p}(\gamma) \neq 0$ voor eindig veel $\mathfrak{p}$ en er geldt voor alle priemidealen $\mathfrak{p}$ van $\mathbb{Z}[\alpha]$:
$$|N(\gamma)| = \prod_\mathfrak{p} \mathcal{N}(\mathfrak{p})^{\mathfrak{l}_\mathfrak{p}(\gamma)}$$

Bewijs. Voor het bewijs zie [JBP93] p.63 – 64. Omdat niet meer geldt $\mathfrak{O} = \mathbb{Z}[\alpha]$ kunnen we voor $\mathfrak{l}_\mathfrak{p}$ niet $e_{p,r}$ nemen. We kunnen dit homomorfisme wel interpreteren als iets soortgelijks; de hoogste exponent van $\mathfrak{p} \subset \mathbb{Z}[\alpha]$ die $\beta \in \mathfrak{O}$ deelt (als $\mathfrak{p}|\beta$ dan $\beta \in \mathfrak{p}$). Hiervoor wordt $\mathfrak{p}$ gerelateerd aan een priemideaal $\mathfrak{q} \subset \mathfrak{O}$ waarvoor $\mathfrak{q} = \mathfrak{O} \cap \mathfrak{p}$. Als $[\mathfrak{O}/\mathfrak{q} : \mathbb{Z}[\alpha]/\mathfrak{p}] = s$ dan wordt voor elke factor $\mathfrak{q}$ van een element in $\mathfrak{O}$ door $\mathfrak{l}_\mathfrak{p}$ s maal de factor $\mathfrak{p}$ geteld.

We kunnen deze stelling toepassen op elementen $a - b\alpha$ waarvan we weten dat ze enkel gedeeld worden door eerstegraads idealen. Daarnaast geldt voor een priemideaal dat correspondeert met een paar (p, r) dat dit het unieke priemideaal is dat p en $a - b\alpha$ bevat [JBP93], waardoor we krijgen:

Gevolg 6.3.20. Zij $\mathfrak{p}$ een priemideaal in $\mathbb{Z}[\alpha]$

(a) Als $\mathfrak{p}$ een eerstegraads priemideaal is en correspondeert met (p, r), dan geldt: $\mathfrak{l}_\mathfrak{p}(a - b\alpha) = e_{p,r}(a - b\alpha)$

(b) Als $\mathfrak{p}$ geen eerstegraads priemideaal is, dan geldt: $\mathfrak{l}_\mathfrak{p}(a - b\alpha) = 0$.

Wegens het homomorfisme $\mathfrak{l}_\mathfrak{p}$ zien we dat de nieuw gedefinieerde exponent ook zal werken voor de situatie $\mathbb{Z}[\alpha] \neq \mathfrak{O}$, als we alleen de priemidealen in $\mathbb{Z}[\alpha]$ beschouwen. Op dit punt zijn we dus in staat een verzameling $R = \{(a, b) : a, b \in \mathbb{Z} \text{ en } \text{ggd}(a, b) = 1\}$ te vinden waarvoor geldt $\forall (a, b) \in R : e_{p,r}(a - b\alpha) \equiv 0 \bmod 2$. We denken dus dat

$$\prod_{(a,b)\in R} \langle a - b\alpha \rangle$$

een kwadraat is (op basis van de factorisatie in eerstegraads priemidealen van $\mathbb{Z}[\alpha]$). Nu zouden we dit op een bepaalde manier willen relateren aan een element in $\mathbb{Z}[\alpha]$, de meest voorliggende situatie zou dan zijn:

$$\prod_{(a,b)\in R} a - b\alpha = \gamma^2$$

Het probleem is dat we niet weten:

1. Of dit element een kwadraat is
2. Of dit element in $\mathbb{Z}[\alpha]$ bevat is

Voor het tweede punt biedt de volgende propositie de oplossing:

Propositie 6.3.21. Zij f een monisch en irreducibel polynoom in $\mathbb{Z}[x]$ met nulpunt $\alpha \in \mathbb{C}$. Laat $\mathfrak{O}$ de ring van algebraïsche gehele getallen in $\mathbb{Q}(\alpha)$. Dan geldt: als $\gamma \in \mathfrak{O}$, dan $f'(\alpha)\gamma \in \mathbb{Z}[\alpha]$.

Bewijs. Laat $f(x)/(x-a) = \sum_{j\in\{0\ldots d-1\}} \beta_j x^j$ voor zekere coëfficienten $\beta_j \in \mathbb{Q}$. Er geldt dat $\beta_0/f'(\alpha), \ldots, \beta_{d-1}/f'(\alpha)$ een basis vormt voor $\mathbb{Q}(\alpha)$ over $\mathbb{Q}$. Daarnaast geldt dat $\forall j : \beta_j \in \mathbb{Z}[\alpha]$. Laat nu $\gamma \in \mathfrak{O} \subset \mathbb{Q}(\alpha)$, dan kunnen we deze dus schrijven als $\gamma = \sum_{j\in\{0\ldots d-1\}} s_j(\beta_j/f'(\alpha))$ voor zekere $s_0, \ldots, s_{d-1} \in \mathbb{Q}$. De trace van een element is te definiëren als $T(e) = \sigma_1(e) + \cdots + \sigma_d(e)$ waarbij $\sigma_1, \ldots, \sigma_d$ de d inbeddingen van $\mathbb{Q}(\alpha)$ in $\mathbb{C}$ zijn zoals gedefinieerd in stelling 6.3.1. Er geldt dat de trace van elementen uit $\mathfrak{O}$ in $\mathbb{Z}$ vallen. Dat komt er mooi uit omdat er geldt: $T(\alpha^k\beta) = s_k$ voor $k \in \{0, \ldots, d-1\}$ waardoor geldt $\forall k : s_k \in \mathbb{Z}$. Hierdoor krijgen we: $f'(\alpha)\beta = \sum_{j\in\{0\ldots d-1\}} s_j\beta_j \in \mathbb{Z}[\alpha]$. Zie ook [CP01] p.254.

Naar aanleiding van deze propositie gaan we dus nog steeds op zoek naar $R = \{(a,b) : a,b \in \mathbb{Z} \text{ en } \mathrm{ggd}(a,b) = 1\}$ waarvoor $\prod_{(a,b)\in R} a - b\alpha = \gamma^2, \gamma \in \mathfrak{O}$. En dan geldt $f'(\alpha)^2\gamma^2$ een kwadraat in $\mathbb{Z}$. Verandert dit dan niets aan de samenhang met het kwadraat in $\mathbb{Z}$? Als we deze op analoge wijze vermenigvuldigen met $f'(m)^2$, dan krijgen we:

$$f'(m)^2 \prod_{(a,b)\in R} a - bm = f'(m)^2c^2 \in \mathbb{Z}$$
$$f'(\alpha)^2 \prod_{(a,b)\in R} a - b\alpha = \gamma^2 \in \mathbb{Z}[\alpha]$$

Noem $\theta(\gamma) = d$, dan krijgen we:

$$f'(m)^2c^2 \equiv \theta(\gamma^2) \equiv \theta(\gamma)^2 \equiv d^2 \bmod n$$

Als er nu zou gelden $\mathrm{ggd}(f'(m)^2, n) = 1$, dan hebben we ook $c^2 \equiv d^2 \bmod n$ en in dat geval is het vermenigvuldigen met $f'(m)^2$ dus niet nodig. Wegens de eigenschappen van f zal gelden $1 < f'(m) < n$. Nu mogen we al aannemen dat $\mathrm{ggd}(f'(m), n) = 1$, want stel dat dit niet het geval is dan hebben we een deler van n gevonden! Dan geldt uiteraard ook $\mathrm{ggd}(f'(m)^2, n) = 1$; er hoeft dus niets aangepast te worden aan het kwadraat in $\mathbb{Z}$.

Het andere probleem was dat we niet weten of $\prod_{(a,b)\in R} a - b\alpha = \gamma^2$ daadwerkelijk een kwadraat is. Er moet een methode bepaald worden om er meer zeker van te zijn dat dit het geval is. Dit zal gedaan worden met een *probabilistische* oplossing; er wordt geprobeerd met meer zekerheid te kunnen vaststellen dat we met een kwadraat te maken hebben. Dit blijkt te kunnen met kwadraatresten.

6.3.8 Kwadraatresten

Het principe waarop dit berust is vrij eenvoudig en daarnaast is het ook makkelijk te implementeren. Laten we een voorbeeld bekijken in $\mathbb{Z}$ en deze wordt vervolgens gegeneraliseerd. Als onze factorbasis gelijk is aan $F = \{2, 3, 5\}$ en we bekijken een kwadraat $r = 16$. Dan geldt:

$$\left(\frac{16}{3}\right) = 1$$
$$\left(\frac{16}{5}\right) = 1$$

Waardoor we weten dat r een kwadraatrest is modulo $2, 3$ en 5. Maar ook als we kijken naar grotere priemgetallen zal gelden dat het Legendresymbool 1 is; als r een kwadraat is, is het een kwadraat modulo elke priem. Deze redenatie werkt ook andersom. Bekijk nu $r = 61$ en dezelfde factorbasis. Dan hebben we weer:

$$\left(\frac{61}{3}\right) = 1$$
$$\left(\frac{61}{5}\right) = 1$$

Weer is r een kwadraatrest modulo de priemgetallen in de factorbasis. Op grond van deze resultaten zou je met betrekking tot de factorbasis F kunnen vermoeden dat 61 een kwadraat is. Maar als we een stapje verder kijken, namelijk naar het priemgetal 7 dan zien we al dat $\left(\frac{r}{7}\right) = -1$ wat betekent dat r geen kwadraatrest is modulo 7. En inderdaad hebben we niet te maken met een kwadraat, 61 is namelijk een priemgetal. Het idee is om een soortgelijke tactiek in $\mathbb{Z}[\alpha]$ toe te passen. Nu blijkt ook dat het om een probabilistische methode zal gaan, aangezien we niet alle priemen tot in den treure langs willen gaan om te kijken of het gevonden getal een kwadraatrest is. Wat er in dit geval gebeurt is wanneer r gegeven is, k verschillende priemgetallen $q < |r|$ te kiezen. Als $\forall q : \left(\frac{r}{q}\right) = 1$ dan is de kans heuristisch gezien 2^{-k} dat r toch geen kwadraat is [CP01]. Het komt erg mooi uit dat we deze methode via de volgende stelling kunnen generaliseren naar $\mathbb{Z}[\alpha]$.

Stelling 6.3.22. *Laat f irreducibel en monisch in $\mathbb{Z}[\alpha]$, met nulpunt $\alpha \in \mathbb{C}$. Stel dat $f'(\alpha)^2 \prod_{(a,b)\in R} a - b\alpha$ een kwadraat is voor een element in $\mathbb{Z}[\alpha]$. Laat (q, s) zo zijn dat q een priemgetal is, $s \in R(q)$ en $f'(s) \neq 0 \bmod q$, waarbij q geen enkele waarde $a - bs$ met $(a, b) \in R$ deelt. Dan geldt:*

$$\prod_{(a,b)\in R} \left(\frac{a - bs}{q}\right) = 1$$

Bewijs. Volgt [CP01]. Bekijk hiervoor het homomorfisme $\tau : \mathbb{Z}[\alpha] \to q\mathbb{Z}$ waarbij $\tau(\alpha) = s \bmod q$. We hebben $f'(\alpha)^2 \prod_{(a,b)\in R} a - b\alpha = \gamma^2$ voor een $\gamma \in \mathbb{Z}[\alpha]$.

Wegens de aanname dat q geen van de $a - bs$ deelt, geldt $\tau(\gamma^2) \not\equiv 0 \bmod q$. Dan hebben we: $\left(\frac{\tau(\gamma^2)}{q}\right) = \left(\frac{\tau(\gamma)^2}{q}\right) = 1 = \left(\frac{f'(\alpha)^2}{q}\right)$ waardoor moet gelden $\left(\frac{\prod_{(a,b)\in R} a-bs}{q}\right) = 1$.

Op gelijke heuristische wijze kan nu voor k dergelijke paren (q, s) getest worden wat het Legendresymbool is. Wanneer het product 1 is kunnen we dan concluderen dat we (zeer waarschijnlijk) een kwadraat hebben gevonden. Hier is de grens $k = \lfloor 3\log(n) \rfloor$ voor gevonden [CP01], we zullen verder niet ingaan op hoe deze grens is bepaald. De tactiek wordt om een 'extra' basis te maken op gelijke wijze als de algebraïsche factorbasis die deze (q, s) bevat en hiervan de Legendresymbolen te bepalen.

Samenvattend: de algebraïsche factorbasis is $F_2 = \{(p, r) : p \leq B \text{ priem}, r \in R(p)\}$, waarbij B dezelfde grens is als bij de priemfactorbasis. Fixeer een b en laat a over een gekozen interval $[-M, M]$ lopen. Als voor a geldt $a \equiv br \bmod p$, dan: $(a + qp) \equiv br \bmod p \ \forall q \in \mathbb{Z}$. Op deze wijze kan bepaald worden welke paren (a, b) een gladde waarde $a - b\alpha$ geven. Daarnaast wordt er een extra basis gemaakt: $F_3 = \{(q, s) : q \leq B \text{ priem}, s \in R(q)\}$, waarbij $\#F_3 = k = \lfloor 3\log(n) \rfloor$. Voor elk paar (a, b) wordt $\forall (q, s) : \left(\frac{a-bs}{q}\right)$ bepaald.

6.4 Zeven samengevat

In de vorige paragraaf zagen we dat we te maken hebben met 3 bases:

$$F_1 = \{p : p \in \mathbb{Z} \text{ priem en } p \leq B\}$$
$$F_2 = \{(p, r) : p \in \mathbb{Z} \leq B \text{ priem}, r \in R(p)\}$$
$$F_3 = \{(q, s) : q \in \mathbb{Z} \leq B \text{ priem}, s \in R(q)\}$$

Een paar (a, b) zullen we geschikt bevinden om een kwadraat mee te maken wanneer geldt:

- $a - bm$ is glad over F_1
- $F(a, b)$ is glad over F_2, waarbij de 'speciale' exponent uit definitie 6.3.16 wordt gebruikt
- $\text{ggd}(a, b) = 1$

De basis F_3 zal van belang worden bij het samenstellen van een kwadraat, oftewel het combineren van meerdere waarden (a, b), maar deze basis heeft geen aandeel in het wel of niet geschikt bevinden van paren. Om te bepalen welke paren (a, b) gladde waarden geven over F_1 en F_2 worden zeeftechnieken gebruikt die nu toegelicht worden.

Bij het zeven over F_1 maken we gebruik van de volgende equivalentie:

$$\forall p \in F_1 : p | a - bm \Leftrightarrow a = b \cdot m + k \cdot p \text{ voor een } k \in \mathbb{Z}$$

Er geldt een soortgelijke uitspraak voor F_2:

$$\forall (p,r) \in F_2 : p | F(a,b) \Leftrightarrow a = b \cdot r + k \cdot p \text{ voor een } k \in \mathbb{Z}$$

De strategie is als volgt; kies $b \leq M$, laat $a \in [-M, M]$ en maak lijsten L_1, L_2 van respectievelijk de waarden $a - bm$ en $F(a,b)$. Neem de eerste priem $p \in F_1$ en bepaal nu voor welke eerste waarde $x = a - bm$ in L_1 geldt $a = b \cdot m + k \cdot p$ voor een $k \in \mathbb{Z}$. Op dit moment weten we dat alle waarden $x + q \cdot p \in [-M, M], q \in \mathbb{N}$ deelbaar zijn door p. Deel deze allemaal zo vaak mogelijk door p en herhaal dit voor alle $p \in F_1$. Deze methode kan gekopieerd worden voor F_2, met als enige aanpassing dat bepaald moet worden voor welke eerste waarde $y = F(a,b)$ in L_2 geldt $a = b \cdot r + k \cdot p$ voor een $k \in \mathbb{Z}$. Wanneer dit voor L_1 en L_2 gedaan is, wordt voor elk paar (a,b) geverifieerd of:

(a) Bijbehorende waarde in L_1 is 1

(b) Bijbehorende waarde in L_2 is 1

(c) $\text{ggd}(a,b) = 1$

Als dit het geval is bevinden we het paar geschikt en wordt het toegevoegd aan de lijst met geschikte paren. Als er genoeg paren gevonden zijn om een afhankelijk stelsel te maken, is deze stap klaar. Anders verhogen we b (met 1) zolang deze kleiner blijft dan M en wordt het hele verhaal herhaald. Als blijkt dat er niet genoeg paren te vinden zijn met de gekozen grens M, zal deze grens opgehoogd moeten worden. Ervan uitgaande dat er wel genoeg paren zijn verzameld, kunnen we doorgaan naar de volgende stap; het bepalen van exponentvectoren om kwadraten te kunnen vinden.

6.5 Kwadraat vinden met exponentvectoren

Zoals bij de kwadratische zeef maken we gebruik van exponentvectoren om kwadraten samen te stellen. Deze exponentvectoren worden bij elkaar gevoegd in een matrix. Hier zal beschreven worden hoe we dit doen aan de hand van de gevonden paren en bases F_1, F_2 en F_3. De tactiek wordt om de twee kwadraten in één keer te vinden. Hiervoor zal een grote matrix gevormd worden (de precieze invulling volgt direct) waarin de rijen bestaan uit de exponent vector van $a - bm$ over F_1 en van $F(a,b)$ over F_2 direct achter elkaar. Het laatste deel van elke rij zal gevuld worden aan de hand van F_3. In deze grote matrix wordt een afhankelijkheid gezocht waardoor de nulvector samengesteld kan worden. Belangrijk om op te merken is dat hier 2 kwadraten uit zullen volgen; één in $\mathbb{Z}$ en de andere in $\mathbb{Z}[\alpha]$.

De matrix die gevuld zal worden heeft $1 + \#F_1 + \#F_2 + \#F_3$ kolommen en 1 rij meer, om het stelsel afhankelijk te maken. Er zal nu een overzicht gegeven worden van hoe deze kolommen per rij (die correspondeert met één paar (a,b)) gevuld worden [CP01]:

(i) Kolom 1 $= \begin{cases} 1 & \text{als } a - bm < 0 \\ 0 & \text{anders} \end{cases}$

(ii) De volgende $\#F_1$ kolommen: loop over $p \in F_1$ en vul in

$$e \bmod 2 \text{ als } p^e | a - bm \text{ en } f > e : p^f \nmid a - bm$$

(iii) De volgende $\#F_2$ kolommen: loop over $(p, r) \in F_2$ en vul in

$$e_{p,r}(a - b\alpha) \bmod 2$$

(iv) De volgende $\#F_3$ kolommen: loop over $(q, s) \in F_3$
en vul in $\begin{cases} 1 & \text{als } \left(\frac{a-bs}{q}\right) = -1 \\ 0 & \text{anders} \end{cases}$

Net zoals bij de kwadratische zeef wordt er gezocht naar een niet triviale lineaire combinatie van de nulvector, waardoor duidelijk wordt welke paren gekozen moeten worden voor het maken van kwadraten. Het vinden van de wortel van het kwadraat in $\mathbb{Z}$ is vervolgens eenvoudig, aangezien we beschikken over de exponentvectoren van de waarden $a - bm$ die in het kwadraat zitten. Hiermee kan uitgerekend worden wat het getal is dat we zoeken. Merk op dat dit proces versneld kan worden door alle berekeningen modulo n te doen. Het vinden van de wortel in $\mathbb{Z}[\alpha]$ is een stuk moeilijker. Dit probleem zal in de volgende paragraaf aangepakt worden.

6.6 Algebraïsche wortel vinden

Laten we ons het volgende herinneren; stel dat $(c^2, \gamma^2) \in \mathbb{Z} \times \mathbb{Z}[\alpha]$ een congruent paar van kwadraten is. Noem $\theta(\gamma) = d \in \mathbb{Z}$. Dan geldt:

$$c^2 \equiv \theta(\gamma^2) \equiv \theta(\gamma)^2 \equiv d^2 \bmod n$$

Het is duidelijk dat we uit γ^2, γ te weten willen komen. Vervolgens passen we dan θ toe op de gevonden wortel en hebben we d gevonden. De methode om dit te doen die hier besproken wordt is afkomstig van Couveignes [Cou93], waarbij gebruik wordt gemaakt van de aanname dat $\deg f$ oneven is. Voor willekeurige graad kan bijvoorbeeld gebruik gemaakt worden van een algoritme van Montgomery, te vinden in [Mon94].

Het algoritme van Couveignes interpreteert $\gamma^2 \in \mathbb{Z}[\alpha]$ als element van meerdere eindige lichamen; in elk eindig lichaam wordt de wortel bepaald en deze kennis tezamen zal door de Chinese reststelling leiden tot γ. Laten we eerst bekijken hoe we de overgang van $\mathbb{Z}[\alpha]$ naar een eindig lichaam kunnen maken.

Laat p een priemgetal zijn zo dat f nog steeds irreducibel is over $(\mathbb{Z}/p\mathbb{Z})[x] \cong \mathbb{F}_p[x]$. Laat ϑ een nulpunt van f zijn in een uitbreidingslichaam van $\mathbb{F}_p$. Door toepassen van stelling 6.2.3 krijgen we dat $\mathbb{F}_p[x]/f\mathbb{F}_p[x] \cong \mathbb{F}_p(\vartheta)$. Hieruit volgt

ook dat dit nieuwe lichaam $p^{\deg f} = p^d$ elementen heeft. We kunnen een correspondentie tussen $\mathbb{Z}(\alpha)$ en $\mathbb{F}_p(\vartheta)$ als volgt weergeven:

$$\begin{aligned} \tau_p : \mathbb{Z}(\alpha) &\to \mathbb{F}_p(\vartheta) \\ \alpha &\mapsto \vartheta \\ s &\mapsto (s \bmod p) \end{aligned}$$

Noem $\gamma^2 = \beta$ en schrijf deze als:

$$\begin{aligned} \gamma &= s_0 + s_1\alpha + \cdots + s_{d-1}\alpha^{d-1} \\ \beta &= r_0 + r_1\alpha + \cdots + r_{d-1}\alpha^{d-1} \end{aligned}$$

Ter verkorting gebruiken we voor $\tau_p(x) \in \mathbb{F}_p(\vartheta)$ de notatie x_p. Nu geldt:

$$\begin{aligned} \tau_p(\gamma^2) = \tau_p(\gamma)^2 &= ((s_0 \bmod p) + (s_1 \bmod p)\vartheta + \cdots + (s_{d-1} \bmod p)\vartheta^{d-1})^2 = \gamma_p^2 \\ \tau_p(\gamma^2) = \tau_p(\beta) &= (r_0 \bmod p) + (r_1 \bmod p)\vartheta + \cdots + (r_{d-1} \bmod p)\vartheta^{d-1} = \beta_p \end{aligned}$$

We zien dus dat de wortel van β_p gelijk is aan $\pm\gamma_p$ in $\mathbb{F}_p(\vartheta)$. Wat we gaan doen is γ bepalen in verschillende eindige lichamen zoals $\mathbb{F}_p(\vartheta)$, om zo steeds de coëfficienten van γ op een andere gereduceerde wijze te krijgen. Zo krijgen we $\gamma_{p_1}, \ldots \gamma_{p_k}$ waarmee γ zelf geconstrueerd kan worden. We zijn uitsluitend geïnteresseerd in $\gamma \bmod n$, wat ervoor zorgt dat we nog efficiënter kunnen rekenen.

De situatie is nu als volgt; laat $p_1, \ldots, p_k$ priemgetallen en γ_{p_i} zodat $0 \leq \gamma_{p_i} < p_i$ en

$$\begin{aligned} \gamma &\equiv \gamma_{p_1} \bmod p_1 \\ &\vdots \\ \gamma &\equiv \gamma_{p_k} \bmod p_k \end{aligned}$$

Dan is γ uniek modulo $\prod_{1\ldots k} p_i$. Verder geldt, als we definiëren;

$$\begin{aligned} P &= \prod_{i=1}^{k} p_i \\ P_i &= \frac{P}{p_i} \\ a_i &= P_i^{-1} \bmod p_i, \end{aligned}$$

dat $z = \sum_{1,\ldots,k} a_i P_i \gamma_{p_i}$ congruent is aan γ modulo P [Cou93]. Merk op dat z groter zal zijn dan γ, sterker nog; z willen we liever niet uitrekenen omdat deze heel groot zal zijn. De uitdaging wordt nu om γ te bepalen, zonder z te berekenen. Laat $q = [z/P]$, het quotiënt van z en P afgerond naar het

dichtstbijzijnde gehele getal. Dan is $z - qP$ dus de restklasse van z modulo P. Omdat γ en z congruent zijn modulo P is nu te schrijven: $\gamma = z - qP$. Als we $(\gamma \bmod (n))$ willen berekenen, gebruiken we de volgende twee eigenschappen:

$$\gamma = z - qP = \left(\sum_{1,\ldots,k} a_i P_i \gamma_{p_i} \right) - qP$$

$$q = \frac{z}{P} = \frac{\sum_{1,\ldots,k} a_i P_i \gamma_{p_i}}{P} = \frac{\sum_{1,\ldots,k} a_i \frac{P}{p_i} \gamma_{p_i}}{P} = \frac{\sum_{1,\ldots,k} a_i \gamma_{p_i}}{p_i}$$

Om deze berekeningen uit te kunnen voeren moeten er nog twee openstaande punten aangepakt worden. Allereerst weten we nog niet hoe worteltrekken in een eindig lichaam gaat. Het andere punt is dat vastgesteld is dat de wortel van β_p gelijk is aan $\pm\gamma_p$ in $\mathbb{F}_p(\vartheta)$. We moeten wel zeker weten dat we bij elke congruentie modulo p_i dezelfde wortel bekijken, dus met hetzelfde teken. Hier komt de beperking van $\deg f$ de hoek om kijken; we laten zien dat het in dit geval goed te bepalen is met welke wortel we moeten werken. Hiervoor gebruiken we de volgende stelling:

Stelling 6.6.1. *Als* $\deg f$ *oneven is, geldt voor* $\beta \in \mathbb{Q}(\alpha)$*:* $N(-\beta) = -N(\beta)$.

Bewijs.

$$\begin{aligned} N(-\beta) &= \prod_{j \in \{1\ldots d\}} \sigma_j(-\beta) \\ &= \prod_{j \in \{1\ldots d\}} -\sigma_j(\beta) \\ &= (-1)^d \prod_{j \in \{1\ldots d\}} \sigma_j(\beta) \end{aligned}$$

Dus we zien in dat wanneer d oneven is, er geldt $N(-\beta) = -N(\beta)$.

Om bovenstaande stelling te kunnen gebruiken, bepalen we wat de norm van een element $\zeta \in \mathbb{F}_p(\vartheta) \cong \mathbb{F}_{p^d}$ is [R.L97]:

$$\mathcal{N}(\zeta) = \zeta \cdot \zeta^p \cdots \zeta^{p^{d-1}} = \zeta^{\frac{p^d-1}{p-1}} \in \mathbb{F}_p$$

Om te bepalen of we steeds dezelfde wortel van γ pakken doen we het volgende: bereken $N(\gamma)$ modulo p_i en ook $\mathcal{N}(\gamma_{p_i})$. Wanneer deze gelijk zijn, moeten we inderdaad γ_{p_i} hebben, anders wordt deze vervangen door $-\gamma_{p_i}$. Merk op dat we, ondanks dat γ nog onbekend is, we de norm $N(\gamma)$ wel kunnen berekenen, er geldt immers:

$$N(\gamma) = N(f'(\alpha)) \prod_{p \leq B} p^{f_p}.$$

Waarbij f_p bepaald is door de factorisatie van γ in priemidealen. Het laatste punt dat nu nog behandeld moet worden is het worteltrekken in een eindig lichaam.

6.6.1 Worteltrekken in een eindig lichaam

Als onderdeel van het algoritme van Couveignes is het nodig dat we weten hoe worteltrekken in een eindig lichaam gaat. Eerst hevelen we het begrip kwadraatrest uit definitie 5.2.1 over naar een eindig lichaam $\mathbb{F}_p(\vartheta)$, wat vanaf nu geschreven zal worden als $\mathbb{F}_q$ voor $q = p^d$.

Definitie 6.6.2. We noemen $\zeta \in \mathbb{F}_q^* = \mathbb{F}_q \backslash \{0\}$ een kwadraatrest als geldt $\zeta \neq 0$ en $\exists \kappa \in \mathbb{F}_q^*$ met $\kappa^2 = \zeta$.

Als we eenzelfde soort test als het Legendresymbool in definitie 5.2.2 willen formuleren, kunnen we gebruik maken van de multiplicatieve groep $\mathbb{F}_q^*$. Deze groep is cyclisch [CP01] en wordt dus voortgebracht door 1 element. Nu kunnen we het volgende criterium opstellen:

Propositie 6.6.3. $\zeta \in \mathbb{F}_q^*$ is een kwadraatrest $\Leftrightarrow \zeta^{\frac{q-1}{2}} = 1$.
Verder geldt ook: ζ is geen kwadraatrest $\Leftrightarrow \zeta^{\frac{q-1}{2}} = -1$.

Bewijs. Zij μ een voortbrenger voor $\mathbb{F}_q^*$. Er geldt $\zeta^{q-1} = 1$, dus moet gelden $\zeta^{\frac{q-1}{2}} \in \{1, -1\}$. Deze obeservatie in combinatie met de volgende twee redenaties geven bovenstaande uitdrukkingen:

1. Stel dat ζ geen kwadraatrest is, dan schrijven we $\zeta = \mu^{2r+1}$. Omdat de orde van μ gelijk is aan $q-1$ en er geldt μ^{q-1} wegens de orde van $\mathbb{F}_q^*$, hebben we $\mu^{\frac{q-1}{2}} = -1$. Nu geldt:

$$1 = \zeta^{\frac{q-1}{2}} = (\mu^{2r+1})^{\frac{q-1}{2}} = \mu^{r(q-1)} \cdot \mu^{\frac{q-1}{2}} = 1 \cdot -1 = -1$$

2. Stel ζ is een kwadraatrest, schrijf dan $\zeta = \mu^{2r}$. Nu geldt:

$$\zeta^{\frac{q-1}{2}} = (\mu^{2r})^{\frac{q-1}{2}} = \mu^{r(q-1)} = 1$$

Het Tonelli-Shanks algoritme gaf initieel een methode om worteltrekken in $\mathbb{F}_p$ voor een priemgetal p te kunnen doen, maar kan uitgebreid worden naar $\mathbb{F}_{p^d}$ door alle operaties daarin uit te voeren. Schrijf $q = 2^s r + 1$, waarbij r oneven is. We herinneren ons de volgende definitie:

Definitie 6.6.4. Zij G een groep van orde $p^s r$. Dan wordt een ondergroep met p^s elementen een *Sylow p-ondergroep* genoemd.

Noem in $\mathbb{F}_q^*$ de Sylow 2-ondergroep S_{2^s}. Deze is in het bijzonder cyclisch, omdat $\mathbb{F}_q^*$ dat ook is. Het is niet moeilijk om een voortbrenger voor S_{2^s} te vinden; neem $\mu \in \mathbb{F}_q^*$ géén kwadraatrest. Voor dit soort elementen geldt het volgende:

Propositie 6.6.5. $\zeta \in \mathbb{F}_q^*$ is geen kwadraatrest $\Leftrightarrow \zeta^{\frac{q-1}{2}} = -1$.

Bewijs. Omdat $|F_q^*| = q - 1$, geldt $\zeta^{q-1} = 1$, waardoor $\zeta^{\frac{q-1}{2}} = \pm 1$. Dan geldt er $\zeta \in \mathbb{F}_q^*$ is geen kwadraatrest $\Leftrightarrow \zeta^{\frac{q-1}{2}} = -1$ met propositie 6.6.3.

Nu zien we het volgende:

$$-1 = \mu^{\frac{q-1}{2}} = \mu^{\frac{2^s r+1-1}{2}} = \mu^{2^{s-1}r} = (\mu^r)^{2^{s-1}}$$

Dus orde$(\mu^r) = 2^s$. Dit element brengt een groep van orde 2^s voort, dus deze moet gelijk zijn aan de Sylow 2-ondergroep. Zo vinden we een voortbrenger van S_{2^s}. We kunnen nu opvolgend aan S_{2^s} een reeks ondergroepen als volgt te definiëren: $F_i = \{\zeta \in \mathbb{F}_q^* : \text{orde}(\zeta) | 2^i\}$. Dan hebben we:

$$\mathbb{F}_q^* \supset F_s \supset \cdots \supset F_2 \supset F_1 \supset F_0 = \{1\}$$

Waarbij $F_s = S_{2^s}$. Omdat elke ondergroep van een cyclische groep weer cyclisch is krijgen we $|F_i| = 2^i$ en $|\mathbb{F}_q^*/F_j| = 2^s r/2^j = 2^{s-j}r$. Gedurende het algoritme wordt $\sqrt{\zeta}$ benaderd door steeds een $\sqrt{\zeta} \in \mathbb{F}_q^*/F_p$ te zoeken voor een p die we laten dalen. Uiteindelijk zal p gelijk aan 0 worden en hebben we $\zeta \in \mathbb{F}_q^*/F_0 = \mathbb{F}_q^*$ want $F_0 = \{1\}$. Eerst zal het algoritme gegeven worden, en vervolgens wordt de correctheid ervan besproken.

Tonelli-Shanks algoritme
Input: $q = 2^s r - 1$, ζ een kwadraatrest en μ^r een voortbrenger van S_{2^r}.
Output: $x \in \mathbb{F}_q$ waarvoor $x^2 = \zeta$.
Opmerking: alle berekeningen worden gedaan in $\mathbb{F}_q$.

1. Initialisatie:
   - $y = \mu^r$
   - $p = s$
   - $x' = \zeta^{(r-1)/2}$
   - $b = \zeta(x')^2$
   - $x = \zeta(x')$
2. Geldt $b \equiv 1 \bmod q$?
   (a) Ja; klaar. Return x.
   (b) Nee; bepaal de kleinste m zodat $b^{2^m} \equiv 1 \bmod q$ en ga door naar stap 3.
3. Laat:
   - $t = y^{2^{p-m-1}}$
   - $y = t^2$
   - $p = m$
   - $x = xt$
   - $b = by$

   En ga weer naar stap 2.

Door een loopinvariant I kunnen we inzien waarom het algoritme correct is. We kunnen I zien als een verzameling eigenschappen van variabelen in het algoritme. Er moeten dan 3 condities aangetoond worden om correctheid van het algoritme te laten zien[**?**]:

1. Initialisatie: I is waar voorafgaand aan de eerste iteratie van de loop.
2. Instandhouding: als I geldt voorafgaand aan een iteratie van de loop, is deze ook nog waar aan het einde van de loop.
3. Terminatie: wanneer de loop eindigt, geeft I een eigenschap waardoor in te zien is dat het algoritme klopt.

We definiëren I nu als:

(i) $x^2 = \zeta b$

(ii) $y^{2^{p-1}} \equiv -1$

(iii) $b^{2^{p-1}} \equiv 1$

Er zal aangetoond worden dat het algoritme in combinatie met deze loopinvariant voldoet aan de 3 genoemde condities.

Initialisatie: Na de initialisatie worden de voorwaarden op volgende wijze vervuld:

(i) $x^2 = \zeta^2 (x')^2 = \zeta\zeta(x')^2 = \zeta b$

(ii) $y^{2^{p-1}} = (\mu^r)^{2^{s-1}} = -1$ want μ geen kwadraatrest

(iii) $b^{2^{p-1}} = (\zeta(x')^2)^{2^{s-1}} = (\zeta^r)^{2^{s-1}} = \zeta^{(q+1)/2} = 1$ want ζ kwadraatrest

Instandhouding: Stel dat voorafgaand aan de loop I geldig was voor x', y', b' en p'. Dan geldt:

(i) $x^2 = (x')^2 t^2 = (x')^2 y = \zeta b' y = \zeta b$

(ii) $y^{2^{p-1}} = t^{2^p} = t^{2^m} = (y')^{2^{p'-1}} = -1$

(iii) $b^{2^{p-1}} = (b')^{2^{m-1}}((y')^{2^{p'-m}})^{2^{m-1}} = -1 \cdot (y')^{2^{p'-1}} = -1 \cdot -1 = 1$

Terminatie: De loop zal eindigen wanneer geldt $b \equiv 1 \bmod q$. In dit geval hebben we $x^2 \equiv \zeta b \equiv \zeta \bmod q$ en wordt x teruggegeven; precies wat we nodig hadden!

Hiermee is ook het laatste openstaande punt vervuld en is het algoritme getallenlichamenzeef compleet. In de volgende paragraaf zal het algoritme in zijn geheel gegeven worden.

6.7 Algoritme

Getallenlichamenzeef
Input: samengestelde n
Output: een deler van n

1. Laat $d = 5$ en $m = \lfloor n^{1/d} \rfloor$.
2. Bepaal $c_0, \ldots, c_{d-1}$ waarvoor $n = c_0 + c_1 m + \cdots + c_{d-1} m^{d-1} + m^d$ en definieer polynoom $f(x) = c_0 + c_1 x + \cdots + c_{d-1} x^{d-1} + x^d \in \mathbb{Q}[x]$.
3. Maak de bases:
   (a) $F_1 = \{p : p \leq y \text{ priem}\}$
   (b) $F_2 = \{(p, r) : p \leq y \text{ priem en } r \in R(p) = \{r : f(r) \equiv 0 \bmod p\}\}$
   (c) $F_3 = \{(q, s) : q > y \text{ priem en } s \in R(q)\}$, waarbij $\#F_3 = \lfloor 3 \log n \rfloor$
4. Bepaal grens M, die bepaalt welke waarden a en b aan kunnen nemen. Laat $b = 1$ en maak lijst S aan.
5. Maak lijsten L_1, L_2 aan. Voor alle $a \in [-M, M]$ waarvoor $\text{ggd}(a, b) = 1$: voeg $(a - bm)$ toe aan L_1 en $F(a, b)$ aan L_2.
6. Voor alle $p \in F_1$
   (a) Bepaal $x = a - bm \in L_1$ waarvoor $a = b \cdot m + k \cdot p$ voor een $k \in \mathbb{Z}$.
   (b) Deel elke waarde $x + q \cdot p \in L_1, q \in \mathbb{Z}$ zo vaak mogelijk door p.
7. Voor alle $(p, r) \in F_2$:
   (a) Bepaal $y = F(a, b) \in L_2$ waarvoor $y = b \cdot r + k \cdot p$ voor een $k \in \mathbb{Z}$.
   (b) Deel vervolgens elke waarde $y + q \cdot p \in L_2, q \in \mathbb{Z}$ zo vaak mogelijk door p.
8. Voor elk paar (a, b): als de bijbehorende waarden in L_1 en L_2 gelijk aan 1 zijn: voeg paar toe aan S.
9. Geldt $\#S > (\#F_1 + \#F_2 + \#F_3 + 1)$?
   (a) Ja: er zijn genoeg paren gevonden, ga door naar stap 10.
   (b) Nee: als $b < M$ laat $b = b + 1$ en ga naar stap 5. Als $b \geq M$ ga opnieuw naar stap 4, maar kies een hogere grens M.
10. Maak een matrix die op de volgende manier per rij gevuld wordt:
    (a) Kolom 1: 1 als $a - bm < 0$, anders 0.
    (b) Volgende $\#F_1$ kolommen: $v(a - bm) \bmod 2$.

(c) Vervolgens voor elke $(p, r) \in F_2 : e_{p,r}(a - b\alpha) \bmod 2$.

(d) Tot slot voor elke $(q, s) \in F_3 : 1$ als $\left(\frac{a-bs}{q}\right) = -1$, anders 0.

11. Maak een niet triviale lineaire combinatie van de nulvector en voeg de bijbehorende paren (a, b) samen in lijst R.

12. Laat $c = \sqrt{\prod_{(a,b)\in R} a - bm}$ en $\gamma^2 = f'(\alpha)^2 \prod_{(a,b)\in R} a - b\alpha$.

13. Bepaal een verzameling priemgetallen $\{p_1, \ldots, p_k\}$.
    En laat: $\mathfrak{P} = \prod_{1\ldots k} p_i,\ \mathfrak{P}_\mathfrak{i} = \frac{P}{p_i},\ \mathfrak{a}_\mathfrak{i} = P_i^{-1} \bmod p_i$.

14. Voor elke p_i bepaal γ_{p_i} met het Tonelli-Shanks algoritme op pagina 34 en laat dan $q = \frac{\sum_{1,\ldots,k} \mathfrak{a}_\mathfrak{i}\gamma_{p_i}}{p_i}$.

15. Laat nu $\gamma = \left(\sum_{1,\ldots,k} a_i P_i \gamma_{p_i}\right) - qP$. Definieer $\theta(s_0 + s_1\alpha + \cdots + s_{d-1}\alpha^{d-1}) = s_0 + s_1 m + \cdots + s_{d-1}m^{d-1}$.

16. Return $\text{ggd}(c - \theta(\gamma), n)$.

Referenties

[ALL82] H.W. Lenstra A.K. Lenstra and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann. **261** (1982), 515–534.

[Cam08] P. J. Cameron, *Introduction to algebra*, Oxford mathematics, 2008.

[Cou93] J. Couveignes, *Computing a square root for the number field sieve*, 95–102.

[CP01] R. Crandall and C. Pomerance, *Prime numbers, a computational perspective*, Springer, 2001.

[Dix81] J. D. Dixon, *Asymptotically fast factorization of integers*, Mathematics of computation **36** (1981), 255–260.

[JBO81] M. Filaseta J. Brillhart and A. Odlyzko, *On an irreducibility theorem of a. cohn*, Canad. J. Math **33** (1981), 1055–1059.

[JBP93] H.W. Lenstra J.P. Buhler and C. Pomerance, *Factoring integers with the number field sieve*, The development of the number field sieve, Springer, 1993, pp. 50–94.

[Keu11] F. Keune, *Getallen*, Epsilon Uitgaven, 2011.

[Lab14] RSA Laboratories, *The rsa challenge numbers*, juni 2014.

[Mon94] P.L. Montgomery, *Square roots of products of algebraic numbers*, Mathematics of Computation 1943-1993: A Half-Century of Computational Mathematics (1994), 567–571.

[Pom96] C. Pomerance, *A tale of two sieves*, Notices of the AMS **43** (1996), 1473–1485.

[Pom08] ———, *Smooth numbers and the quadratic sieve*, Algorithmic Number Theory **44** (2008), 69–81.

[R.L97] R.Lidl, *Finite fields*, Cambridge University Press, 1997.

[RRA78] A. Shamir R.L. Rivest and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM **21** (1978), 120–126.

[SBE] R.P. Brent S. Bai and E.Thomé, *Root optimization of polynomials in the number field sieve*, Mathematics of computation.

[Sha99] A. Shamir, *Factoring integers with the twinkle device*, Cryptographic hardware and embedded systems, Springer, 1999, pp. 2–12.

[ST01] I. Stewart and D. Tall, *Algebraic number theory and fermat's last theorem*, Taylor and Francis Inc, 2001.

[Ste05] Peter Stevenhagen, *The number field sieve*, Surveys in Algorithmic Number Theory, edited by JP Buhler and P. Stevenhagen, Math. Sci. Res. Inst. Publ **44** (2005), 83–100.

[TKZ10] J. Franke A. K. Lenstra E. Thomé J. Bos P. Gaudry A. Kruppa P. L. Montgomery D. A. Osvik H. te Riele A. Timofeev T. Kleinjung, K. Aoki and P. Zimmermann, *Factorization of a 768-bit rsa modulus*, Advances in Cryptology - CRYPTO 2010, Springer, 2010, pp. 333–350.

A Kwadratische zeef (Magma code)

```
/*Gegeven: x, n
Return: x^n
*/

power := function (x, p)
if p eq 0 then
  return 1;
  elif p eq 1 then
    return x;
  elif IsEven(p) then
    return $$(x,p div 2)^2;
  else
    return x*$$(x, (p-1) div 2)^2;
  end if;
end function;

/*Gegeven: integers d en k
Return: de hoogste macht van k die 'past' in d
*/
getBiggestPower := function(d,k)
//Base case
  if d eq 0 then
  return 0;
  end if;
//Recursie
  count := 0;
  while IsIntegral(d) do
    d := d/k;
    count := count +1;
  end while;
  return count -1;
end function;

/*Gegeven: oneven priem p, a waarvoor legendre symbool (a,p)=1
Return: x waarvoor x^2=a mod p
Dit is Algoritme 2.3.8 uit Prime Numbers (Crandall en Pomerance)
*/

quadraticResidue := function(p, n)
//Check de simpele gevallen p eq 3,5,7 mod 8
  a := n mod p;
  if (p mod 8 eq 3) or (p mod 8 eq 7) then
    x := power(a,(p+1) div 4) mod p;
    return x;
  end if;

  if (p mod 8 eq 5) then
    x := power(a, (p+3) div 8) mod p;
    c := power(x,2) mod p;
    if c mod p ne a then
      x := x*power(2,(p-1) div 4) mod p;
    end if;
    return x;
  end if;
```

```
  //Er is maar 1 optie over: p eq 1 mod 8
  d := Random (2, p-1);
  while LegendreSymbol(d,p) ne -1 do
    d := Random (2, p-1);
  end while;

  s := getBiggestPower(p-1,2);
  t := (p-1) div power(2,s);
  A := power(a,t) mod p;
  D := power(d,t) mod p;
  m := 0;
  for i in [0 .. s-1] do
    if power((A*power(D,m)), power(2,s-1-i)) mod p eq -1 then
      m := m+ power(2,i);
    end if;
  end for;

  x := power(a, (t+1) div 2)*power(D, m div 2) mod p;
  return x;
end function;

/*Gegeven: lijst met paren (x,f(x)), priemfactorbasis P
Return: matrix die die exponenvectoren modulo 2 van elke f(x) bevat
*/

getExponentvectorMatrix := function(S, P);
  K := #P ;
  //Begin met een matrix van grootte (K+1) rijen van lengte K met
      allemaal nullen (ga ervan uit dat elke priemmacht 0 is en
      verhoog deze wanneer je hier 'bewijs' van ziet)
  matrix := ZeroMatrix(GF(2), K+1, K);
  //Ga alle priemgetallen uit P langs en voeg steeds de priemmacht
      toe aan de matrix
  //De functiewaarde is voor elke S[i] steeds de 2e positie.
  for i in [1 .. K+1] do
    for j in [1 .. K] do
      p := P[j];
      if S[i][2] mod p eq 0 then
        power := getBiggestPower(S[i][2], p);
        matrix[i,j] := power mod 2;
      end if;
    end for;
  end for;
  return matrix;
end function;

/*Gegeven: exponentvector v, lijst met paren (x,f(x))
Return: integer die precies het product is van alle x in S die
    overeenkomen met de indices die in v een 1 hebben
*/

getXY:= function (v, S)
  prodx := 1;
  prody := 1;
  for i in [1 .. #S] do
    if v[i] eq 1 or v[i] eq -1 then
      prodx := prodx * S[i][1];
```

```
      prody := prody * S[i][2];
    end if;
  end for;
  return prodx, prody;
end function;

/*Gegeven: priemgetal p, waarde r waarvoor r^2=n mod p en x, het
    startpunt van het interval
Return: de eerste waarde in het interval [x,x+1,...,x+k] waarvoor x
    =qp+r voor een q in Z
*/

determineStart := function (p,r,x)
  q := 0;
  temp := q*p+r;
  while temp lt x do
    q := q+1;
    temp := q*p+r;
  end while;
  return (temp - x+1);
end function;

determineStart2 := function (p,r,x)
  return determineStart (p, r+p, x);
end function;

/*Gegeven: matrix M
Return: lijst met vectoren die aangeven welke combinatie van rijen
    uit M opgeteld de nulvector mod 2 geven
Uit: Bressoud - Factorization and primality testing: Gaussian
    Elimination p.114
*/

GaussianEliminationMod2 := function (M)
  m := NumberOfRows(M);
  n := NumberOfColumns(M);
  //Houd een lijst bij waarin wordt opgeslagen welke combinatie van
      paren een kwadraat geeft
  //Dit wordt weergegeven in een vector bestaande uit enen (paar
      wel kiezen) en nullen (paar niet kiezen)
  Solutions := [];

  //Houd de acties bij in een identiteitsmatrix over GF(2)
  T := IdentityMatrix(GF(2), m);
  //Voeg kwadraten toe aan de lijst oplossingen. Check voor elke
      rij of het toevallig al de nulvector is.
  for i in [1 .. m] do
    if IsZero(M[i]) then
    //Als dit zo is, voegen we een vector met een 1 op de plek van
        dit paar toe aan de lijst met oplossingen
      vector := Zero(Parent(T[1]));
      vector[i] := 1;
      Append(~Solutions, vector);
    end if;
  end for;
```

```
  /*Loop over de kolommen. Per kolom bekijk de eerste keer dat een
      1 voorkomt.
  Tel deze kolom op bij alle volgende rijen waar een 1 in deze
      kolom voorkomt. Check steeds of deze optelling zorgt voor de
      nulvector
  Wanneer de nulvector is gevonden, return dan de corresponderende
      rij in T
  */
  for column in [1 .. n] do
    //Update m want de grootte van M kan aangepast zijn
    m := NumberOfRows(M);
    row := 0;
    rowFound := false;
    //Zoek de eerste 1 die voorkomt in die kolom
    while not rowFound and row lt NumberOfRows(M) do
      row := row+1;
      if M[row][column] eq 1 then
        rowFound := true;
      end if;
    end while;

    //Tel de gevonden rij op bij elke volgende rij met een 1 in de
        kolom waar we naar kijken
    if rowFound then
      for j in [row+1 .. m] do
        if M[j][column] eq 1 then
          AddRow(~M, 1, row, j);
          AddRow(~T, 1, row, j);
          //Check of deze rij in M nu de nulvector is geworden
          if IsZero(M[j]) then
            Append(~Solutions, T[j]);
          end if;
        end if;
      end for;
    //Verwijder de rij waar de eerste 1 in stond
      RemoveRow(~M, row);
      RemoveRow(~T, row);
    end if;

  end for;

  return Solutions;
end function;


/*Gegeven: samengestelde n en integer bs
Return: deler van n, als mogelijk geen triviale. Hierbij wordt het
    interval waarin standaard wordt gezocht uitgebreid met de
    waarde bs
*/
Kwadratischezeef := function(n, bs)
//Initialization
  e := Exp(1);;
  B := Ceiling(Sqrt(e^Sqrt((Log(e,n)*Log(e, Log(e,n))))));
  //B := Floor(Sqrt(n)) ;
  //Maak de factorbasis
  PP := PrimesUpTo(B);
```

```
  P := [2];
  for i in [2..#PP] do
  //Stop alle priemgetallen totaan B met legendre symbool 1 erin
    if LegendreSymbol(n,PP[i]) eq 1 then
      Append(~P, PP[i]);
    end if;
  end for;
  P;
  K := #P;
  // For all p in P: find r such that r^2 eq n mod p
  //For p = 2 take 1 (n mod 2 =1)
  R := [1];
  for j in [2.. K] do
    r := quadraticResidue(P[j], n);
    Append (~R, r);
  end for;
// End initialization

//Sieving
//f(x)= x^2-n x_0 = ceiling(sqrt(n)) x_i = x_(i-1) +1
//Zeef de functiewaarden voor B-smooth waarden tot er K+1 paren
    gevonden zijn
// Zeef over een interval van lengte u^u(K+1) met u = ln(n)/(2ln(B)
    )
  Z<x> := PolynomialRing(Integers());
  f := x^2-n;

  u := Log(n)/(2*Log(B));
  t := Ceiling((u^u)*(K+1)) +bs;
  x := Ceiling(Sqrt(n));
  found := false;
  //Zolang er niet genoeg paren zijn gevonden, vergroten we het
      interval om meer paren te vinden
  //Merk op dat je dit zelf ook kunt reguleren met invoer van bs
  while not found do
    interval := [x .. x+t];
    interval;

    FX := [Evaluate(f,i) : i in interval];
  // Je kent voor elke p een r zodat p|r^2-n, dus je weet dat p|f(r
      +q*p) voor alle q in Z
  // Bereken wat de kleinste en grootste q is waarvoor r+q*p dan
      binnen het interval zit en deel elke f(r+q*p) zo vaak
      mogelijk door p
  // Dit doe je dus voor elke r in R. Er geldt dat R[i] hoort bij P
      [i], want voor alle p in P is het legendresymbool 1
    for j in [1..#R] do
    r := R[j];
    p := P[j];
      //Interval[L,R] neem dan range van Floor((r-L)/r) tot Floor((
          R-r)/r) met stapjes van r.
      //Bepaal index van eerste waarde in het interval dat deelbaar
           is door p
      i := determineStart (p,r,x);
        //Deel elke f(r+qp) zo vaak mogelijk door p (in FX)
        while i le t+1 do
          pow := getBiggestPower(FX[i], p);
```

```
          FX[i] := FX[i]/(p^pow);
          i :=i +p;
        end while;
    end for;
  //Herhaal voor alle -r
    minR := [-r : r in R];
    for j in [1..#minR] do
    r := minR[j];
    p := P[j];
      //Interval[L,R] neem dan range van Floor((r-L)/r) tot Floor((
          R-r)/r) met stapjes van r.
      //Bepaal index van eerste waarde in het interval dat deelbaar
           is door p
      i := determineStart2 (p,r,x);
        //Deel elke f(r+qp) zo vaak mogelijk door p (in FX)
        while i le t+1 do
          pow := getBiggestPower(FX[i], p);
          FX[i] := FX[i]/(p^pow);
          i :=i +p;
        end while;
    end for;

  //Alle functiewaarden in FX die nu 1 zijn geworden zijn geschikt
      om tot kwadraat te schrijven!
  //Maak lijst S aan met alle geschikte paren (x,f(x))
    S := [];
      FF := [Evaluate(f,i) : i in interval];
      for i in [ 1 .. #interval] do
        if FX[i] eq 1 and #S lt K+1 then
          Append(~S, [interval[i], FF[i]]);
        end if;
      end for;


      print #S;
      print K+1;

    //Als er meer dan K paren gevonden zijn, kunnen we uit de loop
        breken.
    if #S ge K+1 then
      found := true;
    else
      t := t+10000000;
    end if;

  end while;
//End sieving

//Linear Algebra
// Maak een matrix met alle gevonden exponentvectoren (K+1 rijen, K
     kolommen)
  matrix := getExponentvectorMatrix (S, P);
  matrix;

  //Sol geeft een lijstje met vectoren waar de combinaties in staan
       die je kunt uitproberen
  Sol := GaussianEliminationMod2(matrix);
```

```
  print "solutions found with GE", Sol;
  S;
//End linear algebra

//Factorization
//In Sol zijn de vectoren te vinden die met 1 en 0 aangeven of een
    exponentvector er wel of niet inzit
//Deze corresponderen weer met de paren in S, dus we kunnen we x-
    jes er makkelijk uitvissen

  //We proberen steeds een nieuwe mogelijkheid die in Sol
      opgeslagen staat, te beginnen bij de eerste
  indexSol := 1;
  found := false;

  //Zolang er nog oplossingen zijn en geen geschikte deler, gaan we
       door met proberen
  while not found and indexSol le #Sol do
    x,Y := getXY(Sol[indexSol], S);
    y := Integers()!Sqrt(Y);
    print "try";
    x,y;
    d := Gcd(x-y, n);
    if d ne 1 and d ne n then
      found := true;
      break;
    else
      indexSol := indexSol +1;
    end if;
  end while;
//End factorization

  return d;

end function;

//FACTORED 100003*100019 met bs = 120.000 time 1.280
//FACTORED 1000003*1000033 met bs = 170.000 time 3.140
//FACTORED 10000019*10000079 met bs = 4500000 time 66.690
//FACTORED 100000007*100000037 met bs = 4500000 time 163.110
```

B Getallenlichamenzeef (Magma code)

```
\small
//GNFS

/*Gegeven: n,m,d
Return: coefficienten [c_0, .., c_(d-1), c_d = 1] van polynoom f
    zodat
f(x) = c_0 + .. + c_d-1*x^d-1 + x^d als n = c_0 + .. + c_d-1*m^d-1
    + m^d
*/

getCoeff := function(n,m,d)
```

```
  C := [];

  while d ge 0 do
    c := Floor(n/(m^d));
    Append(~C,c);
    n := n -(c*m^d);
    d := d-1;
  end while;
  return Reverse(C);
end function;

/*Gegeven: variabele x en lijst coefficienten
Return: polynoom dat je maakt met de coefficienten en x.
Als coeff = [c_0, .., c_(d-1), c_d] dan return c_0 + .. + c_d-1*x^d
    -1 + c_d*x^d
*/

makePolynomial := function (x, coeff)
  //Graad van polynoom wordt d-1
  d := #coeff;
  temp := 0;

  for i in [1..d] do
    temp := temp + x^(i-1)*coeff[i];
  end for;

  return temp;
end function;

/*Gegeven: variabelen x,y en lijst coefficienten
Return: multivariabel polynoom dat je maakt met de coefficienten en
     variabelen x en y.
Als coeff = [c_0, .., c_(d-1), c_d] dan return c_0*y^d + .. + c_d
    -1*x^d-1*y + c_d*x^d
*/

makeMultiPolynomial := function (x,y,coeff)
  //Graad van polynoom wordt d-1
  d := #coeff;
  temp := 0;

  for i in [1..d] do
    temp := temp + x^(i-1)*y^(d-i)*coeff[i];
  end for;

  return temp;
end function;

/*Gegeven: functie f en priem p
Return: [r in [0, .. , p-1] : f(r) eq 0 mod p]
*/

setOfZeros := function(f,p)
  R := [];
  for i in [0 .. p-1] do
    if (Evaluate(f,i) mod p) eq 0 then
      Append(~R, 1);
```

```
    else Append(~R,0);
    end if;
  end for;
  return R;
end function;

/*Gegeven: lijst
Return: som van alle elementen in de lijst
*/

sumList := function(list)
  sum := 0;
  for i in [1..#list] do
    sum := sum + list[i];
  end for;
  return sum;
end function;

/*Gegeven: functie f en priemfactorbasis RationalBase
Return: lijst met paren (p,r) waarvoor p een priemgetal <= B is en
    f(r) eq 0 mod p
*/

getAlgebraicBase := function(f, B)
RationalBase := PrimesUpTo(B);
    base := [];
    for i in [1 .. #RationalBase] do
      p := RationalBase[i];
      R := setOfZeros(f, p);
      if (sumList(R) ne 0) then
        for j in [1..#R] do
          if R[j] eq 1 then
          //De lijst R geeft per getal [0 .. p-1] aan of f(r) equiv
               0 (dan staat er een 1),
          //of niet (dan staat er een 0). Corrigeer de index j dus
              met 1
            Append(~base, [p,j-1]);
          end if;
        end for;
      end if;
    end for;
    return base;
end function;

/*Gegeven: aantal k, ondergrens B en functie f
Return: lijst met k paren (q,s) waarvoor q>B en f(s) eq 0 mod p
*/

getExtraBase := function(k,B,f)
  count := 0;
  pairs := [];
  q := NextPrime(B);
  ff:= Derivative(f);

  while count lt k do
    Rq := setOfZeros(f,q);
    if (sumList(Rq) ne 0) then
```

```
      i := 0;
        while count lt k and i lt #Rq-1 do
          found := false;
          if Rq[i+1] eq 1 then
            y :=  Evaluate(ff,i);
            if (y mod q ne 0) then
              found:= true;
              Append(~pairs, [q,i]);
            end if;
          end if;
          i := i+1;
          if found then
            count := count+1;
          end if;
        end while;
    end if;
    q := NextPrime(q);
  end while;

  return pairs;
end function;

/*Gegeven: integers d en k
Return: de hoogste macht van k die 'past' in d
*/
getBiggestPower := function(d,k)
//Base case
  if d eq 0 then
  return 0;
  end if;
//Recursie
  count := 0;
  while IsIntegral(d) do
    d := d/k;
    count := count +1;
  end while;
  return count -1;
end function;


/*Gegeven: positieve integers b en m, priem p en grens M (interval
    = [-M,M])
Return: eerste index waarvoor geldt s= b*m -k*p voor een k in Z
*/

determineStart := function (b,m,p,M)
  q := 1;
  temp := -M -b*m;

  while (temp mod p ne 0) do
    temp := temp + 1;
    q := q+1;
  end while;
  return q;
end function;

determineStartF := function (b,r,p,M)
```

```
  q := 1;
  temp := -M;

  while (temp mod p) ne (b*r mod p) do
    temp := temp + 1;
    q := q+1;
  end while;
  return q;
end function;

/*Gegeven: Rationale basis RationalB, algebraische basis AlgebraicB
    , functie f, root m van f, grens M voor zeef interval en aantal
     V (= benodigd aantal paren)
Return: Een lijst met paren (a,b) die glad zijn over beide bases

Methode: line sieving
Fix b en loop over a, waarbij a in [-M,M]. (Laat vervolgens b
    oplopen zolang er nog niet genoeg paren zijn gevonden)
*/

getPairs:= function (RationalB, AlgebraicB, m, M, V, F)
  b := 1;
  G_ab := [];
  F_ab := [];
  pairs := [];
  while #pairs lt V and b lt M do
  //Voor deze b, haal a uit [-M,M] en bepaal voor deze paren of G(a
      ,b) en F(a,b) smooth zijn over beide bases

  //Bepaal de waarden G(a,b)=a-b*m voor de paren (a,b)
    for a in [-M .. M] do
      //indices mogen niet negatief zijn, dus voer een correctie
          uit
      G_ab[a+M+1] := (a - b*m);
    end for;

  //Zeef de lijst G_ab mbt de rationale basis RationalB
    for j in [1 .. #RationalB] do
      //Deel elke priem p zo vaak mogelijk uit de waarden in G_ab
      p := RationalB[j];
      //p|a-b*m desda a = b*m + k*p voor een k in Z. Deze m is
          meegegeven aan de functie.
      //Bepaal de index van de eerste waarde waarvoor dit geldt
      index := determineStart(b,m,p,M);
      while index le 2*M+1 do
        power := getBiggestPower(AbsoluteValue(G_ab[index]), p);
        G_ab[index] := G_ab[index]/(p^power);
        //Als p|a-b*m, dan a = b*m + k*p en dus a+p = b*m + (k+1)*p
            oftewel p|(a+p)-b*m
        //Schuif dus p stappen op.
        index := index+p;
      end while;
    end for;

  //Bepaal de waarden F(a,b) = b^d * f(a/b) voor de paren (a,b)
    for a in [-M .. M] do
```

```
      //indices mogen niet negatief zijn, dus voer een correctie
          uit
      F_ab[a + M+1] := (Evaluate(F,[a,b]));
    end for;

  //Zeef de lijst F_ab mbt de algebraische basis AlgebraicB
    for j in [1 .. #AlgebraicB] do
      //AlgebraicB bestaat uit paren (p,r)
      p := AlgebraicB[j][1];
      r := AlgebraicB[j][2];
      //p|F_ab desda a = b*r + k*p voor een k in Z en r in R(p), is
          meegegeven in de algebraische basis bij een p.
      //Bepaal de index van de eerste waarde waarvoor dit geldt
      indexAl := determineStartF(b,r,p,M);

      while indexAl le 2*M+1 do
        power := getBiggestPower(AbsoluteValue(F_ab[indexAl]), p);
        F_ab[indexAl] := F_ab[indexAl]/(p^power);
        //Met dezelfde redenatie als eerder, schuif p stappen op.
        indexAl := indexAl+p;
      end while;

    end for;

  /*Voor paren die smooth zijn over beide bases, staat er op de
      corresponderende plek in de
  lijsten F_ab en G_ab een 1 (ze zijn helemaal uitgedeeld). Verder
      willen we dat gcd(a,b) =1.
  */
    for l in [-M .. M] do
      if AbsoluteValue(G_ab[l+M+1]) eq 1 and AbsoluteValue(F_ab[l+M
          +1]) eq 1 then
        if Gcd(l, b) eq 1 then
          Append(~pairs, [l, b]);
        end if;
      end if;
    end for;
    b := b+1;
  end while;

  if #pairs gt V then
    pairs := [pairs[i] : i in [1.. V]];
  end if;

  return pairs;
end function;

/*Gegeven: lengte V, paren pairs, functies G en F, rationale basis
    RFB, algebraische basis AFB en extra basis ExtraB
Return: matrix waarin de V exponentvectoren mod 2 van de paren in
    pairs staat (tov de 3 meegegeven bases)
*/

makeMatrix := function(V, pairs, G, F, RFB, AFB, ExtraB)
  //De matrix heeft V+1 kolommen en V rijen. Hij leeft in GF(2)
      zodat er mod 2 gerekend wordt.
  matrix := ZeroMatrix(GF(2), V, V+1);
```

```
  //pair = (a,b)
  for j in [1 .. #pairs] do
    a := pairs[j][1];
    b := pairs[j][2];
    //1e bit
    G_ab := Evaluate(G, [a,b]);
    if G_ab lt 0 then
      matrix[j][1] := 1;
    else
      matrix[j][1] := 1;
    end if;

    //bits afhankelijk van RFB
    for i in [1..#RFB] do
      p := RFB[i];
      if G_ab mod p eq 0 then
        power := getBiggestPower(AbsoluteValue(G_ab), p);
        matrix[j][i+1] := power mod 2;
      else
        matrix[j][i+1] := 0;
      end if;
    end for;

    l := #RFB +1;
    //bits afhankelijk van AFB
    F_ab := Evaluate(F, [a,b]);
    for i in [1 .. #AFB] do
      p := AFB[i][1];
      if F_ab mod p eq 0 then
        power := getBiggestPower(AbsoluteValue(F_ab), p);
        matrix[j][i+l] := power mod 2;
      else
        matrix[j][i+l] := 0;
      end if;
    end for;

    l :=  l + #AFB;
    //bits afhankelijk van de extra meegegeven basis. Dit is om
        meer zekerheid te krijgen dat we straks daadwerkelijk een
        kwadraat vinden.
    //Hoe groter deze extra basis, hoe meer zekerheid.
    for i in [1 .. #ExtraB] do
      q := ExtraB[i][1];
      s := ExtraB[i][2];
      if LegendreSymbol((a-b*s),q) eq -1 then
        matrix[j][i+l] := 1;
      else
        matrix[j][i+l] := 0;
      end if;
    end for;
  end for;
      return matrix;

end function;

/*Gegeven: grondtal x en macht p
```

```
Return: x^p
Uit Prime numbers (Crandall en Pomerance): Algorithm 2.1.5
    Recursive powering
*/

powerRec := function (x, p)
  if p eq 1 then
    return x;
  elif IsEven(p) then
    return $$(x,p div 2)^2;
  else
    return x*$$(x, (p-1) div 2)^2;
  end if;
end function;

/*Gegeven: exponentvector w, integer n en rationale basis RationalB
Return: de integer die hoort bij de gegeven exponentvector tov de
    rationale basis, mod n
*/

multResiduesModN := function(w, n, RationalB);
  F := ResidueClassRing(n);
  v := F!1;
  for i in [1 .. #RationalB] do
    p := F!RationalB[i];
    exp := w[i];
    if (exp ne 0) then
    v := v*powerRec(p,exp);
    end if;
  end for;
  return v;
end function;

/*Gegeven: set paren Try, rationale basis RationalB en functie G
Return: de squareroot van het product van G(a,b) over alle (a,b) in
     Try

Methode: Bepaal de exponentvectoren van deze paren en tel de
    vectoren bij elkaar op.
Deel elke positie in de vector door 2 en bepaal vervolgens de
    integer die hoort bij deze exponentvector tov RationalB
*/

getIntegerSquareroot := function (Try, RationalB, G,n)
  matrix := ZeroMatrix(Integers(), #Try, #RationalB+1);
  //In Try pairs (a,b) are stored
  for j in [1 .. #Try] do
  a := Try[j][1];
  b := Try[j][2];

  G_ab := AbsoluteValue(Evaluate(G, [a,b]));
  //bits depending on RationalB
    for i in [1..#RationalB] do
      p := RationalB[i];
      if G_ab mod p eq 0 then
        power := getBiggestPower(G_ab, p);
        matrix[j][i] := power;
```

```
      else
        matrix[j][i] := 0;
      end if;
    end for;
  end for;
//Tel alle overige rijen bij de eerste rij op
  for i in [2 .. #Try] do
    AddRow(~matrix, 1, i, 1);
  end for;

  w := matrix[1];
//Deel alle entries door 2 en je krijgt de representatie van v in
    exponent vector over RationalB
  for j in [1.. #RationalB+1] do
    w[j] := w[j] div 2;
  end for;
//Bepaal wat v is aan de hand van de exponent vector w
//Gebruik hiervoor algoritme 2.15: bepaal voor elke p in RationalB
    bereken residue mod n en vermenigvuldig deze met elkaar
  v := multResiduesModN (w, n, RationalB);
  return v;
end function;

/*Gegeven: matrix M
Return: lijst met vectoren die aangeven welke combinatie van rijen
    uit M opgeteld de nulvector mod 2 geven
Uit: Bressoud - Factorization and primality testing: Gaussian
    Elimination p.114
*/

//Gaussian Elimination Bressoud - Factorization and primality
    testing
GaussianEliminationMod2 := function (M)
  m := NumberOfRows(M);
  n := NumberOfColumns(M);
  //Houd een lijst bij waarin wordt opgeslagen welke combinatie van
      paren een kwadraat geeft
  //Dit wordt weergegeven in een vector bestaande uit enen (paar
      wel kiezen) en nullen (paar niet kiezen)
  Solutions := [];

  //Houd de acties bij in een identiteitsmatrix over GF(2)
  T := IdentityMatrix(GF(2), m);
  //Voeg kwadraten toe aan de lijst oplossingen. Check voor elke
      rij of het toevallig al de nulvector is.
  for i in [1 .. m] do
    if IsZero(M[i]) then
    //Als dit zo is, voegen we een vector met een 1 op de plek van
        dit paar toe aan de lijst met oplossingen
      vector := Zero(Parent(T[1]));
      vector[i] := 1;
      Append(~Solutions, vector);
    end if;
  end for;

  /*Loop over de kolommen. Per kolom bekijk de eerste keer dat een
      1 voorkomt.
```

```
  Tel deze kolom op bij alle volgende rijen waar een 1 in deze
      kolom voorkomt. Check steeds of deze optelling zorgt voor de
      nulvector
  Wanneer de nulvector is gevonden, return dan de corresponderende
      rij in T
  */
  for column in [1 .. n] do
    //Update m want de grootte van M kan aangepast zijn
    m := NumberOfRows(M);
    row := 0;
    rowFound := false;
    //Zoek de eerste 1 die voorkomt in die kolom
    while not rowFound and row lt NumberOfRows(M) do
      row := row+1;
      if M[row][column] eq 1 then
        rowFound := true;
      end if;
    end while;

    //Tel de gevonden rij op bij elke volgende rij met een 1 in de
        kolom waar we naar kijken
    if rowFound then
      for j in [row+1 .. m] do
        if M[j][column] eq 1 then
          AddRow(~M, 1, row, j);
          AddRow(~T, 1, row, j);
          //Check of deze rij in M nu de nulvector is geworden
          if IsZero(M[j]) then
            Append(~Solutions, T[j]);
          end if;
        end if;
      end for;
    //Verwijder de rij waar de eerste 1 in stond
      RemoveRow(~M, row);
      RemoveRow(~T, row);
    end if;

  end for;

  return Solutions;
end function;

/*Gegeven: samengestelde n
Return: deler van n, als mogelijk geen triviale.
*/

//GNFS
GNFS := function(n)
  //Setup
  //Log is de natuurlijke logaritme
  //Voor getallen rond de 130 cijfers is de keuze voor d meestal 5
  //d := Floor(((3* Log(n))/(Log(Log(n))))^(1/3)) volgens Prime
      numbers, Crandall en Pomerance. Ook de grens B komt daarvan
  d:=3;
  //B := Floor(Exp((8/9)^(1/3)*Log(n)^(1/3)*Log(Log(n))^(2/3)));
  B:=29;
  //m := Floor (n^(1/d));
```

```
  m := 31;

print "info",d,B,m;
   X<xx> := PolynomialRing(Integers());
    coeff := getCoeff (n, m,d);
    print "coeff", coeff;
    f := makePolynomial(xx, coeff);

    //f moet irreducibel zijn, en anders hebben we een deler
        gevonden:
    //Als f=gh dan f(m)=n=g(m)h(m) en g(m) een factor dus return
        deze.
   if (not IsIrreducible(f)) then
     list := Factorization(f);
     g := list[1][1];
     g;
     return Evaluate(g,m);
   end if;
   print "f",f;

   S<x,y> := PolynomialRing(Integers(), 2);
   F := makeMultiPolynomial (x,y,coeff);
    print "F",F;
   G := x - m*y;
    print "G",G;
   //End Setup

//Setup bases
   //k := Floor(3*Log(n));
   k:=5;
   RationalB := PrimesUpTo(B);
   AlgebraicB := getAlgebraicBase (f, 104);
   ExtraB := getExtraBase (k, 103, f);

//Om een afhankelijk stelsel te vinden zullen V vectoren gevonden
    moeten worden
   V := 1+#RationalB+#AlgebraicB+k+50;
   print "v", V;
   //M := B+1000;
   M:=100000;
//End setup bases
   print "bases", RationalB, AlgebraicB, ExtraB;

   //Sieve
   // Find B-smooth values F(a,b)G(a,b) waarbij a en b coprime zijn.
        Doe dit met een zeef.
   pairs := getPairs (RationalB, AlgebraicB, m, M, V,F);
   print "pairs", #(pairs);
   pairs;
   //pairs;
   //Als je niet genoeg paren hebt, verleg dan de grens. Doe nu maar
        even +500 maar dit komt totaal uit de lucht vallen
   //print "V", V;

   while #pairs lt V do
     M := M +1000;
     pairs := getPairs (RationalB, AlgebraicB, m, M, V,F);
```

```
      print "pairs", #(pairs);
  end while;

  //End Sieve

  //The matrix
  matrix := makeMatrix(V, pairs, G, F, RationalB, AlgebraicB,
      ExtraB);
  //print "matrix", matrix;
  //Linear Algebra
  //Sol is a list of vectors that indicate the correct combinations
      . You have #Sol tries.
  Sol := GaussianEliminationMod2(matrix);
  print "Sol", #(Sol), "x";
  Sol;
  //End linear algebra

  //Try the combinations we've found until a factor has been found
      or there are no more solutions
  found := false;
  countTry :=1 ;
  //Construeer getallenlichaam Q(Alpha)
  NumberFieldQ<alpha> := NumberField(f);
  algebraicProduct := NumberFieldQ!1;
  //Dit homomorfisme gaan we nodig hebben om de algebraische wortel
      af te beelden op de restklassen van n
  phi:=hom< NumberFieldQ -> IntegerRing(n) | m >;
  deler:=1;
  while not found and countTry le #(Sol) do
  print "Counttry", countTry;
  //Pak de paren die corresponderen met deze poging
  print "try number: ", countTry;
  Sol[countTry];
  Try := [pairs[i] : i in [1 .. NumberOfColumns(Sol[countTry])] |
      Sol[countTry][i] eq 1];

  //Vind v mod n met prod{(a,b) in Try}(a-bm) eq v^2 mod n
  //Bekijk de originele exponentvectoren van (a,b) (dus niet mod
      2). Tel deze allemaal bij elkaar op en deel dan elke positie
      door 2. Dan heb je de algebraische wortel gevonden.
  v:= getIntegerSquareroot (Try, RationalB, G, n);
  print "integer root", v;

  //Vind y in Z[Alpha] waarvoor f'(Alpha)^2*prod{(a,b) in Sol}(a-
      bAlpha)=y^2
      for index in [1..#Try] do
        algebraicProduct := algebraicProduct * (Try[index][1] - Try[
            index][2]*alpha);
      end for;
      sqrt := true;
      //Probeer de wortel van het gevonden product te krijgen
      try
        algebraicRoot := Sqrt(algebraicProduct);
      catch e
        sqrt:= false;
      end try;
```

```
    //Als er een wortel gevonden is, wordt deze afgebeeld op de
        restklassen van n
    if sqrt then
      ff := Derivative(f);
      gamma := Evaluate(ff, alpha) *algebraicRoot;
      gamma;
      try
        c := phi(gamma);
      catch e
        c:=0;
      end try;

      print "algebraic root", c;

      deler := Gcd(v-c, n);
      deler;
      if d ne 1 and d ne n then
        found := true;
        break;
      else
        countTry := countTry +1;
      end if;
    else
      countTry := countTry +1;
    end if;

  end while;
  return deler;
end function;

time GNFS(544152);
//Factored 34152
//Factored 44152
//Factored 544152
```