RSA: Gekraakt door kwantumcomputers?

Hector de Windt en William Yang

30 januari 2026

Wat is een cryptosysteem?

Om helder uit te leggen wat een cryptosysteem is, moet eerst de definitie van cryptografie belicht worden. In haar breedste zin is cryptografie de kunst van informatie zodanig verhusselen dat alleen mensen die daartoe gemachtigd zijn de verhusseling ongedaan kunnen maken en die informatie kunnen aflezen. Cryptografie is dus onmisbaar in ons dagelijks leven: zij beveiligt privéberichten tussen vrienden, wachtwoorden in databanken en communicatie tussen overheidsinstanties, onder andere. IBM vatte de algemene principes samen in vier punten die in deelvraag 3 nader uitgewerkt zullen worden bij de belichting van specifieke cryptografische algoritmen die weer verwerkt worden in cryptosystemen, protocollen waarmee men veilig informatie deelt (IBM, z. j.).

Concreet en enigszins wiskundig verwoord is een cryptografische algoritme als het ware een familie van functies $f(m, n_1) : (M, \mathbb{Z}) \to C$ en $g(c, n_2) : (C, \mathbb{Z}) \to M$ waarbij M de verzameling van alle berichten is en C die van alle mogelijke codes. n_1 en n_2 stellen respectievelijk een slot en een bijbehorende sleutel voor: een vijand (iemand die zonder toestemming vertrouwelijke informatie probeert te lezen) moet ze beide achterhalen om berichten te kunnen decoderen.

Echter, die vijand kan een grote databank van gecodeerde berichten samenstellen en die analyseren om patronen te vinden, waarna hij nieuwe berichten (gedeeltelijk) zou kunnen achterhalen zonder n_1 en n_2 te verkrijgen. Claude Shannon, beroemde informaticus uit de 20^{e} eeuw, stelde bovendien twee belangrijke eisen aan cryptografische algoritmes om tegen statistische aanvallen te verdedigen. Ten eerste, dergelijke systemen moeten voldoen aan *confusion*: de relatie tussen parameter en uitkomst mag niet gemakkelijk te doorgronden zijn. Daarnaast moet voldaan worden aan *diffusion*: één bit van de gecodeerde bericht moet afhangen van zo veel mogelijk bits van de originele bericht. Als vuistregel wordt gehanteerd dat één bit van de gecodeerde bericht moet afhangen van de helft van de bits in de origineel (Shannon, 1949).

Het belang van de principes van Shannon wordt geïllustreerd door simpelere cryptografische algoritmes die niet voldoen aan de principes te breken. Neem bijvoorbeeld het Caesarcijfer. Hiermee zou Julius Caesar volgens Suetonis, secretaris van keizer Hadrianus rond het begin van de 2^e eeuw n. Chr., zijn vertrouwelijke bevelen hebben doorgegeven aan zijn leger (Suetonius, 2025). Het algoritme was simpelweg de letters in een bericht een aantal stappen opschuiven in het alfabet, waarbij Z doorschuift naar A. Caesar schoof zelf alles drie letters door, dus de gecodeerde versie van "ROMA CECIDIT" is bijvoorbeeld "URPD FHFLGLW".

Het is gelijk duidelijk dat deze algoritme niet voldoet aan de eisen van Shannon. Een letter in de codetekst hangt namelijk alleen af van één letter in het bericht en de relatie is vrij simpel om te zien door een oplettende aanvaller, zelfs als die niet afwist van de werking van dit algoritme. De frequentie van letters in een taal is namelijk per letter verschillend (zowel in het Latijn als in het Nederlands): "E" komt bijvoorbeeld veel vaker voor dan "Q". Door simpelweg te tellen hoe vaak letters voorkomen in een lang genoeg bericht en de verdeling te vergelijken met de gebruikelijke verdeling van letters kan aardig veel van het bericht goed gegokt worden.

Het bericht bestaat natuurlijk uit meer letters dan alleen "E" of "A", maar omdat de relatie tussen codetekst en bericht zo simpel is, kan een aanvaller door maar één letter in de codetekst en diens vertaling alle andere letters ook vertalen, want hij hoeft alleen de andere letters met dezelfde verschil in plaats te verschuiven. Een aanval op basis van informatie die uit statistische methodes verkregen kan worden kan in de meeste gevallen niet leiden tot een volledige breuk in de veiligheid van een cryptografische algoritme zoals bij het Caesarcijfer, maar kan de benodigde informatie over het bericht lekken om nadere technieken toe te passen. Cryptoloog Lars Knudsen heeft aanvallen op codes die worden toegepast op berichten met een vaste lengte (blokvercijferingen) geordend op volgorde van hoeveel informatie beschikbaar is voor de aanvaller. Een statistische analyse kan zogenaamde "known-plaintext attacks" mogelijk maken, waarbij de aanvaller van tevoren van een (aantal) codetekst(en) de/het bijbehorende bericht(en) kent (Knudsen, 1999).

Uiteraard hangt de veiligheid van een algoritme af ook van hoe makkelijk f te inverteren is (bij systemen zoals RSA is n_1 namelijk algemeen bekend), maar een onbetwistbaar bewijs voor de 'moeilijkheid' van het inverteren van f is niet beschikbaar in de meeste gevallen en in het algemeen bestaan bewijzen voor de 'moeilijkheid' van rekenproblemen zoals een getal factoriseren alleen in zeer uitzonderlijke gevallen. Hoe dat komt en wat de notie van 'moeilijkheid' inhoudt, vergt nadere uitleg.

Algoritmen zijn in feite een reeks aan instructies. De lengte van die reeks hangt (in de meeste gevallen) af van de grootte n van een of ander input. Hoe het precies daarvan afhangt wordt uitgedrukt in 'grote-O-notatie': een algoritme heeft complexiteit $O(L(n))$ als de looptijd ongeveer evenredig is met $L(n)$ (en in de limiet alleen afhangt van $L(n)$). Bijvoorbeeld, een algoritme die het n-de Fibonacci-getal F_n berekent door steeds F_{k-2} en

F_{k-1} tot F_k op te tellen en F_{k-2} te vervangen met F_k totdat $k = n$ moet ongeveer n keer de recursieve formule voor de Fibonacci-reeks toepassen, waardoor $T(n)$ verdubbelt als we n verdubbelen (zonder moeilijkheden met opslag en verwerking van grote getallen in beschouwing te nemen). Er geldt dus dat deze algoritme een looptijd van $O(n)$ heeft, aangezien de looptijd rechtlijnig evenredig is met n. De coëfficiënt van n of een constante term in de uitdrukking voor de looptijd doen er niet toe in de limiet, dus die worden buiten beschouwing gelaten. De notie van het asymptotische gedrag van een functie $f(x)$ kan gedefinieerd worden als

$$\lim_{x\to\infty} \frac{f(x)}{O(f(x))} = k$$

met k een constante.

Problemen worden 'moeilijk' genoemd wanneer er geen algoritme bestaat die dit probleem voor alle mogelijke parameters ervan kan oplossen met complexiteit kleiner of gelijk aan $O(n^\alpha)$. Binnen de informatica worden zulke problemen onder de collectie NP-problemen geordend, waaronder een deel NP-compleet zijn. Van een NP-compleet probleem is bewezen dat, indien er een oplossing (een oplossend algoritme) met complexiteit $O(n^\alpha)$ bestaat, dat er ook 'snelle' oplossingen voor alle NP-problemen bestaan. Echter, ondanks dat na vele decennia onderzoek geen snelle oplossing is gevonden voor een NP-probleem kunnen onderzoekers niet bewijzen dat snelle oplossingen niet bestaan. Toch wijst de gigantische tevergeefse inspanning van informatici erop dat NP-complete problemen een cryptografische algoritme praktisch gezien zeer veilig maakt (Knudsen, 1999).

Voorbeelden van NP-complete problemen die in dit werkstuk behandeld zullen worden zijn het factoriseren van getallen en het berekenen van discrete logaritmen. Ze komen uit de getaltheorie, een aftakking binnen de wiskunde die gaat over gehele getallen. Echter, met de opkomst van kwantumcomputers dreigen deze problemen door algoritmen met complexiteit $O(n^\alpha)$ (of met vergelijkbare complexiteit) opgelost te worden. Hoe ze toegepast worden om cryptografische algoritmen te ontwerpen en hoe kwantum daarbij komt kijken, volgt in de volgende hoofdstukken.

Hoe werkt RSA en waarom is het veilig?

Tegenwoordig is het beveiligen van digitale berichten zoals WhatsApp-gesprekken wellicht de belangrijkste toepassing van cryptografie. Hoe kan gebruikgemaakt worden van de moeilijkheid van factoriseren om een bericht te beveiligen? Dat is precies de werking van RSA, maar om die en andere getaltheoretische algoritmen grondig te ontplooien, moet eerst een fundament gelegd worden in de getaltheorie.

In dit werkstuk wordt uitgegaan van voorkennis over de basisprincipes van modulorekenen en rekenregels omtrent de grootste gemene deler en kleinste gemene veelvoud.

Enkele stellingen tonen aan dat het mogelijk is om een machtsverheffing modulo een arbitrair getal ongedaan te maken, waardoor machtsverheffing als cryptografische algoritme gebruikt kan worden.

Stelling 1.1 (De stelling van Bézout)**.** Bij elk tweetal verschillende positieve gehele getallen (a, b) zijn er gehele getallen x en y zodat $\text{ggd}(a, b) = ax + by$.

Toelichting. Deze stelling volgt uit het bestaan van de algoritme van Euclides, waarmee de grootste gemene deler van twee getallen berekend kan worden. Daarbij zijn alle tussenstappen vergelijkingen van de vorm

$$\begin{aligned} a_0 &= k_0 a_1 + a_2, \\ a_1 &= k_1 a_2 + a_3, \\ &\vdots \\ a_{n-1} &= k_{n-1} a_n + 0 \Rightarrow a_n = \text{ggd}(a_0, a_1), \end{aligned}$$

waarbij de rij $a_0, a_1, \ldots, a_n$ strikt dalend is. Door te herleiden kan de grootste gemene deler van a_0 en a_1 uitgedrukt worden in de coëfficiënten $k_0, k_1, \ldots, k_{n-1}$:

$$\begin{aligned} a_{n-2} &= k_{n-2} a_{n-1} + a_n \Rightarrow a_n = a_{n-2} - k_{n-2} a_{n-1}, \\ a_{n-3} &= k_{n-3} a_{n-2} + a_{n-1} \Rightarrow a_{n-1} = a_{n-3} - k_{n-3} a_{n-2}, \end{aligned}$$

dus $a_n = a_{n-2} - kn - 2(a_{n-3} - k_{n-3} a_{n-2})$. Zo kan steeds gesubstitueerd worden totdat $a_n = \text{ggd}(a_0, a_1)$ uitgedrukt wordt in een lineaire combinatie van a_0 en a_1.

Gevolg 1.2. Stel we bekijken een modulus m en geheel getal a met $\text{ggd}(a, m) = 1$, dan kan 1 uitgedrukt worden als $xm + ya$, wat congruent is aan ya modulo m. y is dus precies het getal modulo m waarvoor het vermenigvuldigen met y overeenkomt met delen door a. y wordt ook wel de multiplicatieve inverse van a modulo m genoemd en bestaat vanwege de stelling van Bézout dan en slechts dan als $\text{ggd}(a, m) = 1$.

Pierre de Fermat, Franse advocaat en wiskundige, bemerkte dat $a^{p-1} \equiv 1 \mod p$ voor p priem en a geheel zodat $\text{ggd}(a, p) = 1$. Deze relatie heet tegenwoordig de kleine stelling van

Fermat. Leonhard Euler, een latere wiskundige, ontdekte bovendien dat een vergelijkbare macht $\varphi(m)$ bestaat voor alle moduli m, ook samengestelde, waarvoor $a^{\varphi}(m) \equiv 1 \mod m$ als $\text{ggd}(a, m) = 1$ en equivalent geformuleerd $a^{\varphi(m)+1} \equiv a \mod m$. Wat $\varphi(m)$ is, wordt in 1.4 uitgelegd.

Definitie 1.3. Bekijk $\{1, 2, \ldots, n-1\}$ voor een natuurlijk getal n. $\varphi(n)$ is aantal elementen hieruit die copriem zijn (dus geen gemene delers behalve 1 hebben) met n. φ wordt ook wel de Euler-phi-functie genoemd.

Stelling 1.4 (De stelling van Euler-Fermat)**.** Zij gegeven natuurlijke getallen a, n met $\text{ggd}(a, n) = 1$. Dan geldt dat $a^{\varphi(n)} \equiv 1 \mod n$.

Bewijs. Bekijk de elementen $x_1, x_2, \ldots, x_{\varphi(n)}$ van de verzameling $\{1, 2, \ldots, n-1\}$ die allemaal copriem met n zijn. $ax_1, ax_2, \ldots, ax_{\varphi(n)}$ zijn allemaal verschillend modulo n, want stel dat er verschillende $k, l \leq \varphi(n)$ zijn met $ax_k \equiv ax_l \mod n$, dan kan a van beide kanten uitgedeeld worden wegens 1.2, waaruit zou volgen dat $x_k \equiv x_l \mod n$ terwijl $x_k \neq x_l$ en beiden kleiner dan n zijn, tegenspraak. Bovendien geldt voor alle $1 \leq t \leq \varphi(n)$ dat ax_t copriem is met n aangezien a en x_t dat zijn. Omdat er precies $\varphi(n)$ getallen kleiner dan n zijn die copriem zijn met n, volgt hieruit dat de rij $ax_1, ax_2, \ldots, ax_{\varphi(n)}$ modulo n simpelweg de rij $x_1, x_2, \ldots, x_{\varphi(n)}$ is, eventueel op een andere volgorde. Dus,

$$x_1 x_2 \cdots x_{\varphi(n)} \equiv (ax_1)(ax_2) \cdots (ax_{\varphi(n)}) = a^{\varphi(n)}(x_1 x_2 \cdots x_{\varphi(n)}) \mod n$$

aangezien de volgorde er niet toe doet bij vermenigvuldiging. $x_1 x_2 \cdots x_{\varphi(n)}$ is copriem met n en kan wegens 1.2 uit beide kanten gedeeld worden, waaruit de relatie

$$a^{\varphi(n)} \equiv 1 \mod n$$

volgt.

Voor een willekeurige $k \in \mathbb{N}$ en een gehele a copriem met een getal N geldt dus dat

$$a^{k\varphi(N)} = (a^{\varphi(N)})^k \equiv 1^k = 1 \mod N,$$

dus $a^{k\varphi(N)+1} \equiv a \mod N$. Met deze relatie hebben Ron Rivest, Adi Shamir en Leonard Adleman in 1977 hun beroemde RSA cryptosysteem ontworpen, het hoofdonderwerp van dit werkstuk. $\varphi(N)$ is bovendien voor een aantal getallen zeer makkelijk berekenbaar, wat nu wordt aangetoond.

Stelling 1.5. Zij gegeven gehele $n \geq 2$, gehele getallen x, y_1, y_2 en natuurlijke getallen m_1, m_2. Dan heeft het stelsel vergelijkingen

$$\begin{cases} x \equiv y_1 \mod m_1 \\ x \equiv y_2 \mod m_2 \end{cases}$$

precies één oplossing voor x modulo $m_1 m_2$ als m_1 en m_2 copriem zijn (een grootste gemene deler van 1 hebben).

Bewijs. Omdat $\text{ggd}(m_1, m_2) = 1$, geldt dat de rij $0, m_1, 2m_1, \ldots, (m_2 - 1)m_1$ allemaal verschillend zijn modulo m_2. Stel namelijk voor verschillende $k, l < m_2$ geldt dat $km_1 \equiv lm_1 \mod m_2$, dan $(k - l)m_1 \equiv 0 \mod m_2$ terwijl $\text{ggd}(m_1, m_2) = 1$ en $|k - l| < m_2$, dus dan moet gelden dat $|k - l| = 0 \Leftrightarrow k = l$, tegenspraak. Merk nu op dat die rij alle restklassen modulo m_2 bevat, waardoor één element te vinden is die congrueert met $y_1 - y_2$, noem die element tm_1. Nu is de oplossing $x \equiv y_1 + tm_1 \mod m_1 m_2$. (Merk tevens op dat $y_1 + tm_1 \leq (m_1 - 1) + (m_2 - 1)m_1 = m_1 m_2 - 1 < m_1 m_2$, dus $y_1 + tm_1$ is ook een restklasse modulo $m_1 m_2$.) □

Stelling 1.6. Voor coprieme natuurlijke getallen m en n geldt dat $\varphi(mn) = \varphi(m)\varphi(n)$.

Bewijs. Laat $a_1, a_2, \ldots, a_{\varphi(m)}$ de getallen kleiner dan m zijn die copriem zijn met m en laat $b_1, b_2, \ldots, b_{\varphi(n)}$ de getallen kleiner dan n zijn die copriem zijn met n. Voor willekeurige $1 \leq k \leq \varphi(m)$ en $1 \leq l \leq \varphi(n)$ bestaat volgens 1.5 een oplossing $x < mn$ voor

$$\begin{cases} x \equiv a_k \mod m \\ x \equiv b_l \mod n. \end{cases}$$

Deze oplossing x kan uitgedrukt worden als $sm + a_k$ of $tn + b_l$ voor gegeven s, t, waardoor $\text{ggd}(x, m) = \text{ggd}(sm + a_k, m) = \text{ggd}(a_k, m) = 1$ en met analoge redenering geldt $\text{ggd}(x, n) = 1$. Elk paar (k, l) levert dus telkens een ander getal kleiner dan mn op dat copriem is met mn en het aantal mogelijke paren is precies $\varphi(m)\varphi(n)$. Dus, $\varphi(mn) = \varphi(m)\varphi(n)$. □

Merk op dat voor een priemgetal p geldt dat $\varphi(p) = p - 1$. Dit volgt uit definitie 1.3 en het feit dat priemgetal p per definitie geen delers (behalve 1) gemeen heeft met de getallen kleiner dan p, waarvan er in totaal $p - 1$ zijn. Met stelling 1.6 kan $\varphi(n)$ nu berekend worden voor alle n die bestaan uit het product van *verschillende* priemgetallen.

Met al het essentiële voorkennis wordt hieronder de uitleg van het verloop van RSA uitgelegd. Om wille van de duidelijkheid nemen we aan dat fictieve personages, Alina en Boas, vertrouwelijk een bericht proberen te sturen aan elkaar, terwijl Casper dit bericht probeert te lezen. De kern van het protocol verloopt dan, getrouw aan moderne conventies (PKCS#1 versie 2.2), als volgt:

1. Alina wil iets vertrouwelijk kunnen ontvangen van Boas. Daartoe genereert zij willekeurig twee verschillende grote priemgetallen p en q, waarvan het product N genoemd zal worden. Stelling 1.6 geeft dat $\varphi(N) = \varphi(pq) = \varphi(p)\varphi(q) = (p - 1)(q - 1)$.

2. Aan de hand van $\varphi(N)$ construeert Alina nu twee getallen a en b die voldoen aan $ab \equiv 1 \mod \varphi(N)$. Hiervoor kiest zij eerst willekeurig een (voldoende groot) a die geen delers gemeen heeft met $\varphi(N)$ (kleiner dan $\varphi(N)$, want ze rekent modulo $\varphi(N)$) en gebruikt ze de algoritme van Euclides om de multiplicatieve inverse van a modulo $\varphi(N)$ te vinden. Die multiplicatieve inverse is b.

3. Alina publiceert nu het tweetal getallen (N, a) als de zogeheten *public key* en houdt b geheim. Casper onderschept (N, a) ook.

4. Boas wil een bericht M versturen. Op M past hij eerst een *paddingprotocol* toe om M_{pad} te krijgen, zodat willekeur geïntroduceerd wordt en de codetekst van M niet te onderscheiden is van andere codeteksten. Later zal hier dieper op worden ingegaan. (Opdat M te onderscheiden is van $M - N$, moet M strikt kleiner dan N zijn.)

5. Boas berekent $M_{pad}^a \mod N$: dat wil zeggen, bij de herhaalde vermenigvuldigingen die zijn computer uitvoert mag hij per vermenigvuldiging de uitkomst vereenvoudigen tot het kleinste getal dat met de uitkomst congrueert modulo N. Boas stuurt $M_{pad}^a \mod N$ naar Alina, waardoor Casper $M_{pad}^a \mod N$ ook leert.

6. Alina ontcijfert het bericht van Boas door $(M_{pad}^a)^b \mod N$ te berekenen. Stelling 1.4 geeft nu dat dit gelijk moet zijn aan $M_{pad} \mod N$.

7. Alina maakt de padding ongedaan om M te verkrijgen van M_{pad}. Nu kan het bericht gelezen worden (Moriarty et al., 2016).

In dit proces zal Casper enkel N, a en M_{pad}^a te weten krijgen. Hieruit weer M_{pad} en dan M afleiden heet het RSA-probleem: bereken de a-demachtswortel van een willekeurig getal modulo N. Voor grote a en N is op dit moment geen efficiënte algoritme bekend om deze berekening uit te voeren. Uiteraard kan de RSA-functie (die M_{pad} naar $M_{pad}^a \mod N$ stuurt gegeven M_{pad}, N en a) zeer gemakkelijk ongedaan gemaakt worden als b bekend is. Stel nu dat Casper de priemfactorisatie van N wist, dan berekent hij $\varphi(N)$ en kan hij, evenals Alina, de algoritme van Euclides gebruiken om b te krijgen met a. Het RSA-probleem is dus ten hoogste even moeilijk voor computers als het factoriseren van grote getallen N (waarvan we weten dat ze uit twee priemfactoren bestaan) (Katzenbeisser, 2001).

Waarom is een zogenaamde paddingprotocol nodig? De RSA-functie heeft geen stochastisch element, waardoor zogenaamde *chosen plaintext attacks* mogelijk zijn. Als Casper van tevoren ongeveer wist waar Alina en Boas over communiceren, kan Casper steeds mogelijke berichten uitproberen door ze met N en a te coderen, net zoals Boas zou doen met M. Uiteindelijk kan Casper per toeval op de codetekst van M komen en weet hij wat het inhoud van M is; immers, hij heeft net zelf M gecodeerd (Goldwasser & Micali, 1982). Praktische implementaties van RSA bevatten dus een paddingprotocol: een manier om een vast aantal willekeurig gekozen extra bits bij een bericht toe te voegen, zodat zelfs twee dezelfde berichten verschillende gepadde versies worden en in codetekst verschillen na

encryptie met eenzelfde exponent en modulus. Volgens de huidige normen wordt het *Optimal Asymmetric Encryption Padding*-protocol (OAEP) gebruikt (Moriarty et al., 2016). De exacte werking van dit protocol valt buiten de omvang van dit werkstuk, maar die en het bewijs voor diens effectiviteit kunnen in een artikel door Bellare en Rogaway (1995) gevonden worden (Bellare & Rogaway, 1995).

Voor de volledigheid is het belangrijk te vermelden dat RSA meestal gebruikt wordt om een key te delen voor een symmetrische cryptosysteem (waarin beide partijen een gedeelde key hebben i.p.v. een private en public key), waarna alle communicatie daarmee verloopt, omdat RSA operatief langzamer is dan bijvoorbeeld het *Advanced Encryption Standard* (AES), het standaardsysteem tegenwoordig. Voor meer informatie over AES en Rijndael (de matrijs waarop AES gebaseerd is), zie hoofdstuk 3 van (Daemen & Rijmen, 2001).

Ter opsomming: de veiligheid van RSA berust op de aanname dat er geen "snelle" algoritme (zoals gedefinieerd in de vorige paragraaf) bestaat om p en q te berekenen met N gegeven. Door een aantal algoritmen te implementeren in Python zal nu getoetst worden wanneer en in hoeverre deze aanname opgaat.

Hoe moeilijk is het factoriseren van N?

Het doel van dit onderzoek is om de relatie tussen looptijd en de grootte van N te benaderen door p en q van steeds verschillende ordes van grootte te genereren en hun product in enkele factorisatie-algoritmen in te voeren om een ijklijn op te stellen. Alle programma's zullen in Python (CPython variant) 3.14.2 geïmplementeerd worden en uitgevoerd worden op een laptop met de volgende specificaties:

- Modelnaam: Framework Laptop 16
- CPU: AMD Ryzen 7 7840HS
- GPU*: AMD Radeon RX 7700S
- Totaal beschikbare RAM: 32 GB (DDR5-SODIMM format)

*Kan bij bepaalde bererkeningen (bijv. vermenigvuldigingen) uitmaken wegens optimalisaties binnen Python zelf.
De module die gebruikt wordt om de tijd bij te houden, heet `timeit`.

Geteste factorisatie-algoritmen
Ga uit van de definities $N = pq$, $k\varphi(N) + 1 = ab$ en $n = \lceil \log_2(N) \rceil$. Kortheidshalve wordt voor alle logaritmen uitgegaan van een grondtal van 2 tenzij anders vermeld. Voor iedere algoritme wordt een *asymptotische analyse* gemaakt om wiskundig te voorspellen wat de looptijd van de algoritme is in grote-O-notatie.

1. **Delertest**: doorloopt alle getallen d groter dan 1 en hoogstens $\sqrt{N}$. Als $N \equiv 0 \mod d$ geldt voor een d, is d een deler van N. Dit is de directste manier om te N te factoriseren en dient als een controlegroep.

   Asymptotische analyse
   De modulo-operatie `x % y` heeft een looptijd van $O(\log(x)\log(y))$, omdat het neerkomt op herhaaldelijk `x - y` uitvoeren. In dit geval zal de looptijd per d van de orde $O(\log(N)\log(d)) \approx O(n\log(d))$. Er zijn telkens $2^{x+1} - 2^x = 2^x$ getallen tussen 2^{x+1} en 2^x, dus er zijn telkens 2^x mogelijkheden voor d van x bits. x loopt maximaal op tot $\log(\sqrt{N}) = \frac{1}{2}\log(N) \approx \frac{n}{2}$. De looptijd is dus in het geheel van de orde

   $$n \sum_{x=1}^{\lceil \frac{n}{2} \rceil} 2 \cdot 2^x \cdot \log(d) = n \sum_{x=1}^{\lceil \frac{n}{2} \rceil} 2^{x+1} \cdot x.$$

   Laat

   $$S = \sum_{x=1}^{\lceil \frac{n}{2} \rceil} 2^{x+1} \cdot x,$$

dan geldt

$$2S = \sum_{x=1}^{\lceil \frac{n}{2} \rceil} 2^{x+2} \cdot x = \sum_{x=2}^{\lceil \frac{n}{2}+1 \rceil} 2^{x+1} \cdot (x-1).$$

Nu kunnen de twee uitdrukkingen van elkaar afgetrokken worden:

$$\begin{aligned}
2S - S = S &= \sum_{x=2}^{\lceil \frac{n}{2}+1 \rceil} 2^{x+1} \cdot (x-1) - \sum_{x=1}^{\lceil \frac{n}{2} \rceil} 2^{x+1} \cdot x \\
&= \lceil \frac{n}{2} \rceil \cdot 2^{\lceil \frac{n}{2} \rceil + 2} + \sum_{x=2}^{\lceil \frac{n}{2} \rceil} 2^{x+1} \cdot (x-1) - \sum_{x=2}^{\lceil \frac{n}{2} \rceil} 2^{x+1} \cdot x - 4 \\
&= \lceil \frac{n}{2} \rceil \cdot 2^{\lceil \frac{n}{2} \rceil + 2} - 4 - 2 \cdot \sum_{x=2}^{\lceil \frac{n}{2} \rceil} 2^{x} \\
&= \lceil \frac{n}{2} \rceil \cdot 2^{\lceil \frac{n}{2} \rceil + 2} - 4 - 2(2^{\lceil \frac{n}{2} \rceil + 1} - 1) \\
&= (\lceil \frac{n}{2} \rceil - 1) \cdot 2^{\lceil \frac{n}{2} \rceil + 2} - 6.
\end{aligned}$$

Dus, de asymptotische looptijd van deze algoritme kan afgeschat worden op $O(n^2 \cdot 2^{\frac{n}{2}})$.

2. **Zeef van Atkin**: soortgelijk met de bekendere *Zeef van Eratosthenes*, maar heeft een betere asymptotische looptijd. De algoritme gebruikt dat, gegeven dat een natuurlijk getal m niet deelbaar door een kwadraat is, er een oneven aantal oplossingen (x, y) is voor de kwadratische vergelijking $ax^2 + by^2 = m$ als m priem is en a, b handig gekozen zijn. De algoritme gaat als volgt:

   Maak een lijst van getallen vanaf 2 tot aan een zoeklimiet L en bij ieder getal een 'marker' voor of het priem is (positief) of niet (negatief). Aan het begin zijn alle markers negatief, behalve die van 2 en 3. Nu worden alle combinaties (x, y) afgegaan met $x^2 < L$ en $y^2 < L$ (deze worden van tevoren berekend om tijd te besparen), waarbij gecontroleerd worden op de volgende criteria:

   a. Als $4x^2 + y^2 > L$, sla deze stap dan over. Draai de marker van $4x^2 + y^2$ om indien $4x^2 + y^2$ congrueert met 1 modulo 12 of 5 modulo 12. Omdraaien wil zeggen: als het negatief was, is het nu positief, en vice versa.

   b. Als $3x^2 + y^2 > L$, sla deze stap dan over. Draai de marker van $3x^2 + y^2$ om indien $3x^2 + y^2$ congrueert met 7 modulo 12.

   c. Als $3x^2 - y^2 > L$, sla deze stap dan over. Draai de marker van $3x^2 - y^2$ om indien $3x^2 - y^2$ congrueert met 11 modulo 12.

   Aan het einde doorlopen we alle getallen m waarvoor $m^2 < L$. Als de marker van m positief is (dus als het priem is), maak dan de markers van de veelvouden van m^2 allemaal negatief. Geef als output een lijst van alle getallen wiens markers positief

zijn. De outputs zijn precies alle priemgetallen tot en met L (Atkin & Bernstein, 2004).

In het bijzonder kan deze algoritme gebruikt worden om $p < \sqrt{N}$ te vinden door $L \geq \sqrt{N}$ te kiezen en voor iedere output te kijken of het N deelt.

Asymptotische analyse

De Zeef van Atkin heeft drie hoofdzakelijke rekenstappen: het berekenen van de getallen $x^2, y^2, 4x^2 + y^2, 3x^2 + y^2$ en $3x^2 - y^2$, de modulo-operaties, en de delingen door de priemgetallen.

(a) x^2 en y^2 worden voor alle (x, y) zodat $x^2, y^2 < L$ berekend, dus het aantal berekeningen schaalt met $(\sqrt{\sqrt{N}})^2 = \sqrt{N}$. $4x^2 + y^2, 3x^2 + y^2, 3x^2 - y^2$ worden voor bijna alle paren (x, y) berekend, dus in totaal bedraagt dat ook ongeveer $3\sqrt{N}$ berekeningen. Al met al lijkt dit gedeelte $4\sqrt{N}$ berekeningen te kosten. Hierbij is gelet op welke soorten berekeningen uitgevoerd moeten worden en daarop geoptimaliseerd. Vermenigvuldigingen met 2 en additions zijn bijvoorbeeld veel sneller dan algemene vermenigvuldigingen en delingen. Machtsverheffen wordt vermeden door x^2 en y^2 als aparte variabelen bij te houden en steeds met $2x+1$ of $2y + 1$ op te tellen om $(x + 1)^2$ en $(y + 1)^2$ te krijgen.

(b) Voor bijna alle paren (x, y) wordt gecontroleerd of $4x^2 + y^2 \equiv 1 \vee 4x^2 + y^2 \equiv 5 \mod 12$; het tweede criterium wordt niet meer gecontroleerd als $4x^2 + y^2$ al aan het eerste voldoet. Dat bespaart gemiddeld $\frac{1}{12}\sqrt{N}$ modulo-operaties, wat het totaal naar $\frac{23}{12}\sqrt{N}$ modulo-operaties brengt. Ook wordt voor bijna alle paren (x, y) gecontroleerd of $3x^2 + y^2 \equiv 7 \mod 12$, wat $\sqrt{N}$ modulo's bedraagt. Voor bijna alle (x, y) waarvoor geldt $x > y$ wordt gecontroleerd of $3x^2 - y^2 \equiv 11 \mod 12$: dat zijn $\frac{1}{2}N$. In totaal worden ongeveer $\frac{41}{12}\sqrt{N}$ modulo-operaties gebruikt. De priemgetallen tot aan $\sqrt{N}$ vinden heeft daarom looptijd $O(\sqrt{N}) = O(2^{\frac{n}{2}})$.

(c) Noem $\pi(x)$ het aantal priemgetallen kleiner dan of gelijk aan x. De priemgetalstelling geeft $O(\pi(x)) = \frac{x}{\ln(x)}$ (Zagier, 1997). Na de zeef worden dus rond $\frac{\alpha L}{\ln(L)}$ delingen uitgevoerd. De factors van N vinden tussen alle priemgetallen heeft dus looptijd $O\left(\frac{2^{\frac{n}{2}}}{n}\right)$.

Hoe kunnen kwantumcomputers RSA in de toekomst verbreken?

Uit de resultaten van de vorige paragraaf blijkt dat de beste factorisatiealgoritmen exponentiële tijdscomplexiteit hebben. Echter, Peter Shor heeft een algoritme bedacht dat de eigenschappen van kwantumcomputers handig benut om (in theorie) een getal te factoriseren binnen een tijdsduur gelijk aan een polynoom van het aantal bits die nodig zijn om dat getal uit te drukken (dus bijvoorbeeld $n^3 - 2n^2 + 5n - 1$). Deze heeft als grondslag de algoritme van Euclides (zie Stelling 1.1 voor nadere toelichting): als $N = pq$ en een veelvoud van p of q - zeg kp - zijn gegeven, dan zal de algoritme van Euclides p geven binnen zeer korte tijd. (p zal bijna altijd de output van Euclides zijn, omdat k bijna nooit een factor p of q bevat; p en q zijn immers gigantisch grote priemgetallen.) De algoritme verloopt als volgt:

1. Reken modulo N. Laat x een willekeurig geheel getal zijn. (Deze is bijna altijd copriem met N, omdat N het product is van twee zeer grote priemgetallen.)

2. Vind de kleinste positief gehele r zodat $x^r \equiv 1$. Merk op: dit is niet per se $\varphi(N)$, maar kan toevallig voor de gekozen x een kleiner getal zijn. Als de gevonden r oneven is, begin weer opnieuw met een andere x. Zo niet, ga door met stap 3.

3. Omdat r even is, kan r geschreven worden als $2s$ met s geheel. Omdat $x^{2s} \equiv 1$, geldt $x^{2s} - 1 = kN$ voor een gehele k. Merk op dat de linkerlid een verschil van kwadraten is, dus $(x^s - 1)(x^s + 1) = kN = kpq$. Aangezien p en q priemgetallen zijn, moet ieder van de twee in precies één van de factoren in de linkerlid zitten. Ze kunnen niet beide in $x^s - 1$ zitten, want anders is $x^s - 1$ deelbaar door N en $x^s \equiv 1$ terwijl $s < r$, en dat is in tegenspraak met de aanname dat r de kleinste exponent is waarvoor deze congruentie geldt. Het is dus bekend dat $x^s + 1$ minstens één van de priemfactoren van N bevat.

4. Voer de algoritme van Euclides uit op $x^s + 1$ en N om p, q of N te krijgen (afhankelijk van welk(e) van de twee $x^s + 1$ deelt). Als hieruit N komt, begin dan opnieuw met een andere x. Anders komt een priemfactor van N hieruit en kan N door die factor gedeeld worden om de andere te krijgen. N is nu gefactoriseerd.

Voor deze stappen is een kwantumcomputer nog niet strikt nodig, maar een klassiek computer kan r niet efficiënt vinden (de beste manier is om domweg alle getallen vanaf 1 af te gaan tot de eerste waarvoor $x^r \equiv 1$), terwijl er een vernufte operatie voor bestaat in kwantumcomputers, namelijk de Quantum Fourier Transform (QFT of kwantumfouriertransformatie) (Shor, 1997). Om deze uit te leggen, is echter een basis in de terminologie en principes van kwantumcomputatie nodig.

Een klassieke computer slaat informatie op in de vorm van *bits*: nullen en enen. Als een nul afgelezen wordt, komt er ook altijd een nul uit. Een kwantumcomputer daarentegen draait op *qubits*: voor te stellen als samenstellingen, oftewel *superposities*, van nullen en

enen. Praktisch worden qubits als volgt gedefinieerd:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle ,$$

waarbij $|0\rangle$ en $|1\rangle$ de nul- en één-toestanden voorstellen. Bij aflezing (of vaker *meting*) van $|\psi\rangle$ zal niet altijd dezelfde toestand verschijnen: de kans dat een $|0\rangle$ toestand waargenomen wordt, is precies $|\alpha_0|^2$ conform conventies in de kwantummechanica. (Merk hierbij op dat α_0 wellicht complex kan zijn. De absolutus-strepen stellen dus de norm voor.) Doordat de coëfficiënten van $|0\rangle$ en $|1\rangle$ een kansverdeling voorstellen, moet gelden dat $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Analoog kan een *systeem* van t qubits gedefinieerd worden als

$$|\Psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle + \ldots + \alpha_{2^t-1} |2^t - 1\rangle ,$$

aangezien $2^t - 1$ het grootste getal is dat met t nullen en enen kan worden uitgedrukt in het binaire telsysteem. Ook hier geldt dat $|\alpha_0|^2 + |\alpha_1|^2 + \ldots + |\alpha_{2^t-1}|^2 = 1$. Een systeem van t qubits kan intuitief voorgesteld worden als een vector in 2^t-dimensionaal ruimte op de eenheids-2^t-sfeer (de equivalent van de eenheidscirkel in 2^t-dimensionaal ruimte). Bewerkingen op een systeem van qubits kunnen dus als matrices worden voorgesteld waarmee het systeemsvector vermenigvuldigd wordt. Deze bewerkingen heten vaak logische poorten, maar kortheidshalve wordt de term bewerking gehanteerd in dit werkstuk.

Een voorbeeld van een bewerking is de $CNOT$-bewerking, die er als volgt uit ziet:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Deze matrix is 2^2-dimensionaal, dus het is een bewerking op een systeem van 2 qubits. Als deze bewerking op klassieke bits zou worden toegepast, dan werkt het als volgt: "Verander de tweede bit van 0 tot 1 (of andersom) als de eerste bit een 1 is."Echter, op een stelsel van qubits die zich in superpositie bevinden is het effect van $CNOT$ minder simpel te omschrijven. Laat het systeem van 2 qubits weergegeven worden door $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$, dan

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_2 \end{pmatrix}$$

Van boven naar beneden stellen de coördinaten van zo'n vector de coëfficiënten van $|0\rangle , |1\rangle , |2\rangle$ en $|3\rangle$ voor, oftewel $|00\rangle , |01\rangle , |10\rangle , |11\rangle$ in binair. Hierbij wordt de meest linkse bit als eerste bit beschouwd. Bij een meting blijft de kans om een nul of één in de eerste qubit aan te treffen gelijk: voorheen gold

$$P(0) = |\alpha_0|^2 + |\alpha_1|^2, P(1) = |\alpha_2|^2 + |\alpha_3|^2$$

en na $CNOT$ geldt

$$P(0) = |\alpha_0|^2 + |\alpha_1|^2, P(1) = |\alpha_3|^2 + |\alpha_2|^2.$$

Echter, de kansverdeling van de tweede qubit is wel veranderd. Voor $CNOT$ gold

$$P(0) = |\alpha_0|^2 + |\alpha_2|^2, P(1) = |\alpha_1|^2 + |\alpha_3|^2$$

en daarna geldt

$$P(0) = |\alpha_0|^2 + |\alpha_3|^2, P(1) = |\alpha_1|^2 + |\alpha_2|^2.$$

Hieruit is het effect van $CNOT$ op de beide qubits af te lezen. Stel bijvoorbeeld dat de eerste qubit gemeten wordt en zijn superpositie *vervalt* tot de $|0\rangle$-toestand. Omdat α_3 en α_2 de kansen op $|2\rangle = |10\rangle$ respectievelijk $|3\rangle = |11\rangle$ voorstellen, volgt uit de meting dat ze allebei 0 zijn. Immers, de eerste qubit is $|0\rangle$ en niet $|1\rangle$. De (complementaire) kansen $|\alpha_0|^2$ en $|\alpha_1|^2$ schalen dan zodanig dat hun som 1 wordt. Als de eerste qubit tot $|1\rangle$ zou zijn vervallen, zouden de overgebleven kansen $|\alpha_2|^2$ en $|\alpha_3|^2$ zijn. Een $CNOT$ *verstrengelt* dus de kansverdelingen van de twee qubits in het systeem, waardoor de ene meting afhankelijk wordt van de andere. In het algemeen verstrengelen alle CU-bewerkingen, die bewerking U op een systeem uitvoert alleen als een gekozen controle-qubit $|1\rangle$ is, de controle-qubit en het systeem waar U al dan niet op opereert. De verstrengeling in het geval van het $CNOT$-voorbeeld kan het beste weergeven worden door de vector te schrijven als een lineaire combinatie:

$$\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_2 \end{pmatrix} = \alpha_0 |0\rangle |0\rangle + \alpha_1 |0\rangle |1\rangle + \alpha_2 |1\rangle |0\rangle + \alpha_3 |1\rangle |1\rangle .$$

Zodra $|0\rangle$ gemeten wordt bij de eerste qubit, komen alle termen met $|0\rangle$ als eerste qubit te vervallen.

De algoritme van Shor gebruikt op geraffineerde wijze de bijzondere eigenschap van verstrengeling. Het idee is om een t-qubit register (het telregister)

$$\frac{1}{\sqrt{T}} \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} = \frac{1}{\sqrt{T}}(|0\rangle + |1\rangle + \ldots + |T-1\rangle)$$

met gelijke kans op alle toestanden van 0 t/m $T-1$ te verstrengelen met een ander register van oplopende x-machten modulo N (vanaf nu het machtregister). Het machtregister bestaat dus uit $\lceil \log_2(N) \rceil = n$ qubits. Specifiek is de bedoeling om toestand $|y\rangle$ te verstrengelen met toestand $|x^y \mod N\rangle$. Hoe gaat dit proces in de praktijk? Eerst moet opgemerkt worden dat y in binaire vorm geschreven kan worden als $y_0 2^0 + y_1 2^1 + \ldots + y_{t-1} 2^{t-1}$ en dat $x^y \mod N$ niets anders is dan

$$x^y = \underbrace{x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdots x}_{y_0 2^0 + y_1 2^1 + \ldots + y_{t-1} 2^{t-1} \text{ keer}} = x^{y_0 2^0} \cdot x^{y_1 2^1} \cdots x^{y_{t-1} 2^{t-1}}.$$

Omdat $ab \mod N$ gelijk is aan $(a \mod N) \cdot (b \mod N)$ - waarbij $ab \mod N$ staat voor de restklasse van ab modulo N, geldt dus

$$x^y \mod N = (x^{y_0 2^0} \mod N) \cdot (x^{y_1 2^1} \mod N) \cdots (x^{y_{t-1} 2^{t-1}} \mod N).$$

Noem CU_k de bewerking die $|x\rangle$ stuurt naar $|x^{2^k} \mod N\rangle$ als een controle-qubit $|1\rangle$ is. Door steeds de qubit uit het telregister die overeenkomt met de k-de plaats in de binaire weergave van y (dus y_k) als controle te kiezen, zullen de twee registers na alle t de bewerkingen (CU_0, CU_1, …, CU_{t-1}) verstrengeld zijn zoals gewenst, namelijk

$$\frac{1}{\sqrt{T}}(|0\rangle |1\rangle + |1\rangle |x\rangle + |2\rangle |x^2 \mod N\rangle + \ldots + |T-1\rangle |x^{T-1} \mod N\rangle).$$

Vermenigvuldiging van twee getallen is niets anders dan herhaaldelijk optellen (en vermenigvuldiging modulo N is herhaaldelijk optellen modulo N), dus de bewerkingen voor vermenigvuldiging kunnen op analoge wijze worden geconstrueerd uit bewerkingen voor optellingen. Voor een gedetailleerde constructie, zie (Vedral, Barenco & Ekert, 1996).

Merk op dat de gezochte r voor de gekozen x ervoor zorgt dat het machtregister maar r verschillende toestanden kan aannemen, namelijk $|1\rangle, |x\rangle, |x^2 \mod N\rangle, \ldots, |x^{r-1} \mod N\rangle$. Immers, $|x^r \mod N\rangle = |1\rangle$ en daarmee $|x^{r+k} \mod N\rangle = |x^k \mod N\rangle$. De toestand $|x^k \mod N\rangle$ is dus verbonden met $|k\rangle, |r+k\rangle, |2r+k\rangle$, enzovoort in het telregister. Als het machtregister nu gemeten wordt, komt daar een toestand $|x^k \mod N\rangle$ uit. De lineaire combinatie vervalt dan tot

$$\sqrt{\frac{r}{T}}(|k\rangle |x^k \mod N\rangle + |r+k\rangle |x^k \mod N\rangle + |2r+k\rangle |x^k \mod N\rangle + \ldots).$$

Als de kans om een bepaald getal a te meten bij meting van de telregister uitgezet wordt tegen a, komt daar een periodieke functie uit met periode r met pieken ter hoogte van $\frac{r}{T}$ bij $k, r+k, 2r+k, \ldots$. Deze periodieke kansdichtheid kan met behulp van de eerdergenoemde QFT omgezet worden tot een wederom periodieke kansdichtheid, maar deze keer met een periode van $\frac{T}{r}$, oftewel (ongeveer) het aantal pieken in de oorspronkelijke kansdichtheid.

Beschouw de QFT-matrix voor t qubits (het aantal qubits in het telregister), dus met T rijen en kolommen:

$$\mathbf{F} = \frac{1}{\sqrt{T}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & z & z^2 & \cdots & z^{T-1} \\ 1 & z^{2\cdot 1} & z^{2\cdot 2} & \cdots & z^{2(T-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z^{(T-1)} & z^{(T-1)\cdot 2} & \ldots & z^{(T-1)(T-1)} \end{pmatrix},$$

waarbij $z = e^{\frac{2\pi i}{T}}$, oftewel $z^T = 1$ en $\arg(z) = \frac{2\pi i}{T}$. De coëfficiënten $(\alpha_0, \alpha_1, \ldots, \alpha_{T-1})$ worden dus als volgt gemanipuleerd:

$$\alpha_a \mapsto \frac{1}{\sqrt{T}}(\alpha_0 \cdot 1 + \alpha_1 z^a + \alpha_2 z^{2a} + \ldots + \alpha_{T-1} z^{(T-1)a}).$$

Merk eerst op dat de QFT-matrix de lengte van de vector $(\alpha_0, \alpha_1, \ldots, \alpha_{T-1})$ behoudt, waardoor de resulterende vector nog steeds lengte 1 heeft (*genormaliseerd* is). Dit zal hieronder bewezen worden.

Stelling 2.1. De QFT-matrix $\mathbf{F}$ is een unitaire transformatie (behoudt de lengtes van vectoren).

Bewijs. Om te bewijzen dat een $n \times n$ matrix unitair is, moet enkel bewezen worden dat $\mathbf{A}\mathbf{A}^\dagger = I_n$, waarbij I_n de n-dimensionale eenheidsmatrix is en de elementen van $\mathbf{A}^\dagger$ als volgt worden gedefinieerd (Axler, 2023):
Laat $a_{p,q}$ het element in rij p kolom q van $\mathbf{A}^\dagger$ zijn en definieer $b_{p,q}$ als het element in rij p kolom q van $\mathbf{A}$. Dan geldt

$$a_{p,q} = \overline{b_{q,p}}.$$

(Hier betekent $\overline{z}$ de complexe geconjugeerde van z, dus als $z = a + bi = re^{i\theta}$, dan $\overline{z} = a - bi = re^{-i\theta}$.)

Merk op dat $\mathbf{F}$ lijnsymmetrisch is in de hoofddiagonaal: immers, het element in rij p kolom q is $z^{(p-1)(q-1)}$ en het element in rij q kolom p is $z^{(q-1)(p-1)} = z^{(p-1)(q-1)}$. $z^{(p-1)(q-1)}$ is te schrijven als $e^{\frac{2\pi i}{T}\cdot(p-1)(q-1)}$, waarvan de geconjugeerde $e^{-\frac{2\pi i}{T}\cdot(p-1)(q-1)} = z^{-(p-1)(q-1)}$ is. Het element in rij p kolom q van $\mathbf{F}^\dagger$ is dus $z^{-(p-1)(q-1)}$. Hiermee kunnen alle elementen van $\mathbf{F}\mathbf{F}^\dagger$ uitgerekend worden.

Bekijk het element in rij p kolom q van het product van de matrices zonder de coëfficiënt $\frac{1}{\sqrt{T}}$ in beschouwing te nemen om wille van de leesbaarheid. Die factor kan aan het einde weer erbij vermenigvuldigd worden. Het element op dat plek is precies het inproduct van de vectoren die gevormd worden door kolom p van $\mathbf{F}^\dagger$ en rij q van $\mathbf{F}$, en die zijn respectievelijk $(1, z^{-(q-1)}, z^{-2(q-1)}, \ldots, z^{-(T-1)(q-1)})$ en $(1, z^{p-1}, z^{2(p-1)}, \ldots, z^{(T-1)(p-1)})$. Het inproduct is dan

$$1 + z^{p-1-(q-1)} + z^{2(p-1)-2(q-1)} + \ldots + z^{(T-1)(p-1)-(T-1)(q-1)} =$$
$$z^{0(p-q)} + z^{1(p-q)} + z^{2(p-q)} + \ldots + z^{(T-1)(p-q)}.$$

Merk op dat

$$z^{T(p-q)} - 1 = (z^{p-q} - 1)(z^{0(p-q)} + z^{1(p-q)} + z^{2(p-q)} + \ldots + z^{(T-1)(p-q)})$$

geschreven als het verschil van T-machten, waardoor de som van oplopende z^{p-q}-machten gelijk is aan

$$\frac{z^{T(p-q)} - 1}{z^{p-q} - 1}$$

voor $p - q \neq 0$ en $z^{p-q} - 1 \neq 0$. Echter, $z^T = e^{T\cdot\frac{2\pi i}{T}} = e^{2\pi i} = 1$, dus de teller is gelijk aan 0. Hieruit volgt dat de oplopende som van z^{p-q}-machten ook 0 is wanneer $p \neq q$, maar

wanneer $p = q$ wordt de som

$$\underbrace{z^0 + z^0 + z^0 + \ldots + z^0}_{T \text{ keer}},$$

waardoor het element T is. $p = q$ geldt precies voor alle elementen op de hoofddiagonaal. Dus,

$$\mathbf{F}\mathbf{F}^\dagger = \left(\frac{1}{\sqrt{T}}\right)^2 \begin{pmatrix} T & 0 & 0 & \cdots & 0 \\ 0 & T & 0 & \cdots & 0 \\ 0 & 0 & T & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & T \end{pmatrix} = I_T.$$

□

2.1 bevestigt dat de QFT een geldige bewerking op een systeem van qubits is, maar waarom kan $\mathbf{F}$ het gewenste effect bereiken? Herinner dat

$$\alpha_a \mapsto \frac{1}{\sqrt{T}}(\alpha_0 \cdot 1 + \alpha_1 z^a + \alpha_2 z^{2a} + \ldots + \alpha_{T-1} z^{(T-1)a}).$$

In het geval van het telregister is opmerkelijk dat de meeste α_a (die overeenkomen met de kans op $|a\rangle$ bij meting) gelijk zijn aan 0, behalve als a toevallig gelijk is aan één van $k, r+k, 2r+k, \ldots$; in dat geval is α_a gelijk aan $\sqrt{\frac{r}{T}}$. De uitdrukking waar α_a naartoe wordt gestuurd, kan dus herschreven worden als

$$\frac{1}{\sqrt{T}}\left(\sqrt{\frac{r}{T}} \cdot z^{ka} + \sqrt{\frac{r}{T}} z^{(r+k)a} + \sqrt{\frac{r}{T}} z^{(2r+k)a} + \ldots\right) = \frac{\sqrt{r}}{T} z^{ka}(z^0 + z^{ra} + z^{2ra} + \ldots).$$

De kans om $|a\rangle$ te meten na de QFT is dus gelijk aan

$$\begin{aligned} |\alpha_a|^2 &= \left|\frac{\sqrt{r}}{T} z^{ka}(z^0 + z^{ra} + z^{2ra} + \ldots)\right|^2 \\ &= \left|\frac{\sqrt{r}}{T}\right|^2 \cdot |z^{ka}|^2 \cdot |z^0 + z^{ra} + z^{2ra} + \ldots|^2 \\ &= \frac{r}{T^2} \cdot 1 \cdot |z^0 + z^{ra} + z^{2ra} + \ldots|^2. \end{aligned}$$

De norm van een eenheidswortel (herinner dat z de T-demachts complexe eenheidswortel is) is immers altijd 1, omdat het op de complexe eenheidscirkel ligt. De kans om $|a\rangle$ te meten hangt dus af van hoe groot de norm van de som is. Een intuitief beeld is dat de som het grootst is wanneer de machten van z^{ra} dicht bij elkaar liggen, omdat het sommeren van complexe getallen analoog gaat aan het sommeren van vectoren. De eenheidswortels heffen elkaar al snel op wanneer de machten $0, ra, 2ra, \ldots$ niet veelvouden van T zijn, dus de som is maximaal wanneer a een veelvoud van $\frac{T}{r}$ is en zeer klein elders.

Stel nu dat het telregister gemeten wordt. Hieruit zal bijna altijd[1] een getal van de vorm $\frac{mT}{r}$ voor een bepaalde m komen, of een getal dicht in de buurt daarvan. Door een grote T te kiezen kan de afstand $\frac{T}{r}$ tussen twee pieken vergroot worden en het aantal termen in de som van machten van z^{ra}, waardoor de kans dat een meting dicht in de buurt van een piek (of daarop) ligt, vergroot wordt. Normaal wordt uitgegaan van $T = N^2$ om het aantal qubits in het telregister te balanceren met de nauwkeurigheid van metingen (Shor, 1997). De kans om een getal dat hooguit 3 verschilt van $\frac{mT}{r}$ te observeren is namelijk > 0.97 voor $T = N^2$ (Ekerå, 2024). Verder in dit werkstuk zal ook verondersteld worden dat $T = N^2$.

Laat j het geobserveerde getal zijn. Als j heel klein is (dicht bij 0), moet in ieder geval een nieuwe meting worden gedaan. Na j gedeeld te hebben door N^2 kan r geschat worden met een relatief simpele klassieke (niet kwantum) algoritme:

1. Gebruik de algoritme van Euclides om $\text{ggd}(j, N^2)$ te vinden. $\frac{N^2}{\text{ggd}(j,N^2)}$ is dan de kleinste noemer waarmee $\frac{j}{N^2}$ geschreven kan worden. Nu bestaan er twee gevallen: ofwel $\frac{N^2}{\text{ggd}(j,N^2)} \geq N$, dan wel $\frac{N^2}{\text{ggd}(j,N^2)} < N$. In het eerste geval is zeker dat die noemer niet gelijk kan zijn aan r, want $r < N$. Ga dan door naar stap 3. Ga in het tweede geval naar stap 2.

2. In dit geval moet getest worden of $x^{\frac{N^2}{\text{ggd}(j,N^2)}} \equiv 1 \mod N$. Zo wel, dan is met uitermate hoge zekerheid[2] te zeggen dat $\frac{N^2}{\text{ggd}(j,N^2)} = r$. Zo niet, sla alsnog het net geteste getal op en stop dan de algoritme (hieronder meer hierover).

3. Herhaal stappen 1 en 2, maar met het volgende getal in de rij $(j, j+1, j-1, j+2, j-2, j+3, j-3)$ in plaats van het getal dat net gebruikt werd.

Als r hieruit niet gevonden is, herhaal dan de berekening op de kwantumcomputer om een nieuwe meting te verkrijgen die wezenlijk verschilt van de net gebruikte meting. Praktisch gezien is de kans dat de algoritme bij stap 2 minstens één keer een getal kleiner dan N krijgt zeer groot (namelijk > 0.97), dus een nieuwe meting zal meestal alleen nodig zijn als m en r gemene delers hebben, waarbij m komt van de uitdrukking voor de dichtstbijzijnde piek $\frac{mN^2}{r}$ t.o.v. j. In dit geval zal de opgeslagen resultaat een deler van r zijn. Noem dat resultaat d_1 (en volgende opgeslagen waardes worden $d_2, d_3, \ldots$ genoemd). Bij de k-de keer dat stap 2 wordt uitgevoerd zijn de delers $d_1, d_2, \ldots, d_{k-1}$ al bekend, dus kan $\text{kgv}\left(\frac{N^2}{\text{ggd}(j,N^2)}, d_1, d_2, \ldots, d_{k-1}\right)$ genomen worden als de macht van x, wetende dat het kleinst gemene veelvoud uiteindelijk r is zodra alle delers van r aanwezig zijn.

Ervan uitgaande dat de mogelijkheden voor m uniform verdeeld zijn over $[0, r-1]$, geldt dat de kans dat $\text{ggd}(m, r) > 1$ gelijk aan $\frac{\varphi(r)}{r}$. Als r even is (want dat moet het zijn

[1] Shor heeft in zijn eigen analyse de kans minstens $\frac{4}{\pi^2}$ is wanneer gekozen wordt voor $T = N^2$ (Shor, 1997).

[2] De kans dat de kleinste noemer per toeval een veelvoud van r is, is verwaarloosbaar klein (Ekerå, 2024).

zodat de algoritme van Shor afgemaakt kan worden), zal deze breuk net iets groter dan 0.5 zijn. In het overgrote deel van de gevallen zijn hooguit 3 metingen voldoende om r te reconstrueren uit haar delers (Shor, 1997).

De looptijd van het kwantum-gedeelte van de algoritme van Shor is op de orde van $O(n^3)$ (herinner dat $\lceil \log_2(N) \rceil = n$) vanwege machtsverheffing modulo N als uitgegaan wordt van de constructie van Vedral, Barenco en Ekert (Vedral et al., 1996). Alternatieven bestaan waarbij optelling gedaan wordt met QFTs die een asymptotische looptijd hebben van $O(n^2)$, maar ze zijn niettemin slomer voor de kleine aantallen qubits die in dit werkstuk voorkomen vanwege de grote coëfficiënt van n^2 (Pavlidis & Gizopoulos, 2014). De klassieke gedeeltes zijn allemaal zeer snel: de algoritme van Euclides heeft voor grote inputs (die van ons zijn vrij groot) een looptijd van $O((2n)^2) = O(4n^2)$, omdat het effectief bestaat uit meermalig de modulo-operatie uitvoeren op getallen hooguit op de orde van N^2. De rest van de klassieke bewerkingen nemen een verwaarloosbaar klein gedeelte van de tijd in. Al met al is de looptijd van de algoritme van Shor dus ongeveer $O(n^3)$, veel beter dan klassieke algoritmen voor grote n.

Praktische implementatie

Hier kwamen wij tegen een aantal obstakels. Ten eerste: Quantum Inspire, de service die we van plan waren te gebruiken om de algoritme mee te testen, heeft geen voldoende grote computers. Zelfs de simulatie kan maar maximaal 24 qubits aan, terwijl er veel meer nodig zijn om een zinvolle vergelijking te maken met een klassieke computer.

Op dit moment zijn wij een alternatief (IBM Quantum) aan het testen, maar dat zal geld kosten en is iets ingewikkelder om te gebruiken, dus dit stuk zal later nog aangevuld worden.

Het is binnen de tijd die gegeven is voor dit werkstuk en de beperkingen van beschikbare materialen onmogelijk om de algoritme van Shor daadwerkelijk te gebruiken om getallen van voldoende aantallen bits te factoriseren. Hierdoor moet een analyse van het uitgeschreven computerprogramma volstaan. Bij dit analyse worden een aantal aannames gemaakt:

- De bewerkingen die beschikbaar zijn verschillen per kwantumtoestel. Hier wordt uitgegaan van de beste toestellen die beschikbaar gesteld worden door IBM Quantum, namelijk `ibm_fez`. Voor zover de auteurs kunnen vinden, biedt IBM Quantum de krachtigste toestellen aan aan het algemene publiek. De beschikbare bewerkingen worden *basisbewerkingen* genoemd. De basisbewerkingen relevant voor de algoritme

van Shor zijn (AbuGhanem, 2025):

$$NOT = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$R_x\left(\frac{\pi}{2}\right) = \frac{\sqrt{2}}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix},$$

$$R_z\left(\frac{\pi}{2}\right) = \frac{\sqrt{2}}{2}\begin{pmatrix} 1-i & 0 \\ 0 & 1+i \end{pmatrix}$$

$$T = R_z\left(\frac{\pi}{4}\right) = \begin{pmatrix} \frac{2+\sqrt{2}}{4} - i\frac{2-\sqrt{2}}{4} & 0 \\ 0 & \frac{2+\sqrt{2}}{4} + i\frac{2-\sqrt{2}}{4} \end{pmatrix}$$

$$T^\dagger = R_z\left(-\frac{\pi}{4}\right) = \begin{pmatrix} \frac{2+\sqrt{2}}{4} + i\frac{2-\sqrt{2}}{4} & 0 \\ 0 & \frac{2+\sqrt{2}}{4} - i\frac{2-\sqrt{2}}{4} \end{pmatrix}$$

$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Zie de documentatie van IBM Quantum zelf: (IBM Quantum Platform, z. j.).

- De gemiddelde tijd die één basisbewerking in beslag neemt is ongeveer 68 nanoseconden (AbuGhanem, 2025).
- Bewerkingen op twee qubits zijn alleen mogelijk als er een fysieke verbinding is tussen de twee qubits (Kaye, Laflamme & Mosca, 2006). Echter, een willekeurig paar qubits zijn bijna nooit per toeval verbonden, waardoor fysieke qubits op een zeer specifieke manier moeten overeenkomen met de qubits zoals omschreven staan in het programma zodat het aantal benodigde SWAP-operaties om de toestanden van twee qubits te verwisselen minimaal is (AbuGhanem, 2025). De optimalisatie hiervan is zeer ingewikkeld en valt buiten de omvang van dit werkstuk, dus hier zal het invloed van verbondenheid verwaarloosd worden.
- Op (een onbeperkt aantal) verschillende qubits kunnen tegelijkertijd bewerkingen worden uitgevoerd. Tijd wordt gemeten in eenheden van hoe lang een basisbewerking duurt, dus eenheden van 68 nanoseconden. De tijd wordt hier ook wel *circuitlengte* genoemd.

Het volledige circuit voor de algoritme van Shor bevat berekeningen op een klassieke computer. Deze zullen apart behandeld worden na het kwantumgedeelte.

Constructie van $H, CNOT$ en $CCNOT$

H heet de *Hadamard gate* en is onmisbaar voor de meeste algoritmen, waaronder die van Shor. Het wordt gedefinieerd door

$$H = \frac{\sqrt{2}}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$$

en stuurt in het bijzonder de $|0\rangle$- en $|1\rangle$-toestanden naar $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, zodat $P(0) = P(1)$ bij meting. H wordt als volgt geconstrueerd:

$$\begin{aligned}
R_z\left(\frac{\pi}{2}\right) R_x\left(\frac{\pi}{2}\right) R_z\left(\frac{\pi}{2}\right) &= \frac{2\sqrt{2}}{8}\begin{pmatrix}1-i & 0\\ 0 & 1+i\end{pmatrix}\begin{pmatrix}1 & -i\\ -i & 1\end{pmatrix}\begin{pmatrix}1-i & 0\\ 0 & 1+i\end{pmatrix}\\
&= \frac{\sqrt{2}}{4}\begin{pmatrix}1-i & 0\\ 0 & 1+i\end{pmatrix}\begin{pmatrix}1-i & 1-i\\ -1-i & 1+i\end{pmatrix}\\
&= \frac{\sqrt{2}}{4}\begin{pmatrix}-2i & -2i\\ -2i & 2i\end{pmatrix}\\
&= -i\cdot\frac{\sqrt{2}}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} = -iH.
\end{aligned}$$

Het is niet mogelijk om de factor $-i$ weg te halen met alleen de beschikbare basisbewerkingen, maar dat is ook niet nodig. $-i$ ligt namelijk op de complexe eenheidscirkel en veroorzaakt daarom een zogenaamde *globale faseverschil*[3]. $-iH$ kan dus simpelweg als H beschouwd worden, net als dat $e^{i\theta}\mathbf{A}$ simpelweg als $\mathbf{A}$ voor alle mogelijke bewerkingen $\mathbf{A}$ beschouwd kan worden. Men schrijft ook wel $e^{i\theta}\mathbf{A} \cong \mathbf{A}$. H heeft een circuitlengte van 3, omdat het bestaat uit 3 basisbewerkingen.

Herinner dat

$$CNOT = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}.$$

Om $CNOT$ te construeren zijn zowel H als CZ nodig. Echter, CZ is een bewerking op twee qubits, dus H moet ook geschreven worden als een bewerking op twee qubits: eentje blijft onveranderd, en de andere begaat de effecten van H. Wederom wordt bij de toestanden $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ de meest linkse qubit de eerste genoemd. Stel dat H op de tweede

[3]Een factor $e^{i\theta}$ nooit van belang wanneer het voor alle qubits aanwezig is, want $|e^{i\theta}\alpha|^2 = 1\cdot|\alpha|^2$. In het bijzonder kan H geschreven worden als een bewerking op twee qubits waarbij de ene simpelweg onveranderd blijft, dus $H \otimes I_2$ waarbij $\otimes$ het tensorproduct is. Het tensorproduct is het wiskundige fundament waarop redenaties met meerdere qubits berusten, maar de exacte werking daarvan valt helaas buiten de omvang van dit werkstuk. Hier is van belang dat $(e^{i\theta}H)\otimes\mathbf{A}$ *isomorf* is aan $e^{i\theta}(H\otimes\mathbf{A})$ voor alle mogelijke bewerkingen $\mathbf{A}$, wat betekent dat één qubit met $e^{i\theta}$ vermenigvuldigen effectief equivalent is aan alle qubits met $e^{i\theta}$ vermenigvuldigen (Kaye et al., 2006).

qubit wordt toegepast, dan noteert men formeel

$$I_2 \otimes H = \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 & 0 \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 & 0 \\ 0 & 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ 0 & 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix} = \frac{\sqrt{2}}{2} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix}.$$

Nu kan $CNOT$ geconstrueerd worden als volgt:

$$\begin{aligned}(I_2 \otimes H)(CZ)(I_2 \otimes H) &= \frac{2}{4} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \\ &= \frac{1}{2} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix} \\ &= \frac{1}{2} \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = CNOT.\end{aligned}$$

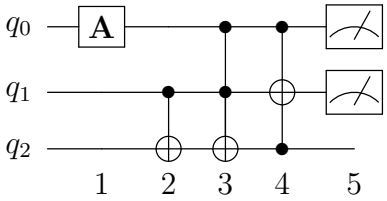
Omdat de tweede qubit twee keer door H heen gaat, is de circuitlengte van $CNOT$ in totaal 7.

De $CCNOT$ heeft een vergelijkbare functie als $CNOT$, maar in plaats van één controle-qubit heeft $CCNOT$ twee. $CCNOT$ is dus een bewerking op drie qubits. Als matrix wordt het zo geschreven:

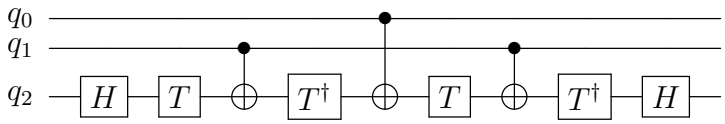
$$CCNOT = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Effectief betekent $CCNOT$ "Draai $P(0)$ en $P(1)$ voor de derde qubit om als de eerste en tweede qubits in de $|1\rangle$ toestand verkeren". De constructie van $CCNOT$ op basis van

matrices is te ingewikkeld om volledig uit te schrijven, maar de bewerkingen waar het uit bestaat kunnen in een *circuitdiagram* worden weergeven. Dit is hoe een circuitdiagram eruit ziet:



Hierbij betekent de rechthoek met **A** dat bewerking **A** op qubit q_0 toegepast wordt. $\oplus$ is de NOT-bewerking, waarbij $P(0)$ en $P(1)$ gewisseld worden voor een qubit. De zwarte stippen duiden controle-qubits aan, dus "voer de verbonden bewerking uit als de qubits met zwarte stippen in de $|1\rangle$ toestand verkeren", dus in kolom 2 staat een $CNOT$ en in kolommen 3 en 4 staan $CCNOT$s. In kolom 5 staat dat q_0 en q_1 gemeten worden. De $CCNOT$-bewerking is als volgt te construeren (Al-Baytay & Perkowski, 2024):



Per kolom is de benodigde tijd gelijk aan de (langzaamste) bewerking in die kolom. Alles bij elkaar genomen heeft $CCNOT$ dus een circuitlengte van 31.

Analyse van circuit voor machtsverheffing modulo N

In dit werkstuk wordt de constructie van Vedral, Barenco en Ekert (1996) aangehouden. Herinner dat het doel is om $CU_0, CU_1, \ldots, CU_{t-1}$ te construeren. Bekijk de logica van CU_k: dit circuit vermenigvuldigt een input m met $x^{2^k} \mod N$ als de controle-qubit y_k (namelijk het k-e cijfer in de binaire ontbinding van y) in de $|1\rangle$-toestand zit. Laat

$$m = m_0 2^0 + m_1 2^1 + \ldots + m_{n-1} 2^{n-1}$$

m geschreven in binair zijn. Merk op dat m maximaal uit n bits bestaat gezien het een getal modulo N is. Vermenigvuldigen is niets anders dan herhaaldelijk optellen, namelijk:

$$\begin{aligned} Mx^{2^k} \mod N &= (m_0 2^0 + m_1 2^1 + \ldots + m_{n-1} 2^{n-1}) x^{2^k} \mod N \\ &= (m_0 2^0 x^{2^k} \mod N) + (m_1 2^1 x^{2^k} \mod N) + \ldots + (m_{n-1} 2^{n-1} x^{2^k} \mod N) \mod N, \end{aligned}$$

dus elke CU_k kan ontbonden worden in de circuits $CV_{k,0}, CV_{k,1}, \ldots, CV_{k,n-1}$, waarbij $CV_{k,l}$ een constante $2^l x^{2^k} \mod N$ bij een input optelt wanneer de controle-bit m_l in de $|1\rangle$-toestand zit. Deze constantes $2^l x^{2^k} \mod N$ kunnen redelijk snel van tevoren berekend worden op een klassieke computer, maar daar later meer over. De CV-circuits zijn dus in essentie een reeks optellingen modulo N en die moeten gerealiseerd worden met algemene optelling- en aftrekkingscircuits.

Een zeer nuttige eigenschap van kwantumcomputers is dat bewerkingen allemaal een inverse hebben, omdat ze equivalent zijn aan meetkundige manipulaties in een hoger-dimensionaal ruimte, wat blijkt uit dat qubits en bewerkingen voorgesteld worden als respectievelijk vectoren en unitaire (hermetische) matrices (zie het bewijs van stelling 2.1). Dus, stel dat een optellingscircuit bestond met inputs (a, b) en outputs $(a, a + b)$, dan is het circuit dat (a, b) stuurt naar $(a, a - b)$ precies hetzelfde als het optellingscircuit, maar dan van achteren naar voren. Met andere woorden, de laatste bewerking is nu de eerste, de een-na-laatste wordt de tweede, enzovoort. Zo kunnen optellingscircuits gebruikt worden als aftrekkingscircuits.

Noem SUM het circuit dat a en b bij elkaar optelt, waarbij zowel a als b bestaan uit hooguit n qubits. Schrijf a en b uit als binaire getallen met cijfers $a_{n-1}a_{n-2}\ldots a_1a_0$ respectievelijk $b_{n-1}b_{n-2}\ldots b_1b_0$. Bedenk hoe optellen werkt in het tientallig stelsel:

$$\begin{array}{cccc} & {\scriptstyle 1} & {\scriptstyle 1} & \\ & 6 & 3 & 7 \\ + & 1 & 8 & 9 \\ \hline & 9 & 2 & 6 \end{array}$$

Wanneer twee cijfers een grotere som dan 10 hebben, wordt een extra 1 bij het volgende cijfer opgeteld als een soort overdracht. Zo werkt optellingen van binaire getallen ook: wanneer $a_{k-1} + b_{k-1} \geq 2$, dus wanneer $a_{k-1} = b_{k-1} = 1$, dan moet 1 bij $a_k + b_k$ opgeteld worden:

$$\begin{array}{cccccccc} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & \\ & 0 & 0 & 1 & 1 & 0 & 1 \\ + & 0 & 0 & 0 & 1 & 1 & 1 \\ \hline & 0 & 1 & 0 & 1 & 0 & 0 \end{array}$$

Om bij te houden of een overdracht nodig is, zijn n overdracht-qubits nodig. Omdat de qubits van b uiteindelijk naar die van $a+b$ veranderd worden, moeten de berekeningen voor wanneer overdracht nodig is, gedaan worden voordat de qubits van b vervangen worden. Ook moeten de overdracht-qubits aan het einde allemaal hersteld worden tot 0. Vedral en collega's hebben in deze instantie gekozen voor een ontwerp waarbij zo weinig mogelijk *ancilla-qubits* (qubits die tijdelijk informatie opslaan) te pas komen om zo ruimte te besparen, een principe geboren uit het feit dat het totale beschikbare aantal qubits momenteel de beperkende factor is voor berekeningen. Echter, hiervoor moeten langere circuits worden geaccepteerd. Hun ontwerp gaat als volgt:

1. Laat de overdracht-qubits $c_1, \ldots, c_{n-1}$ heten, waarbij c_k bij a_k en b_k opgeteld moet worden. Definieer ook een teken-qubit t als de $n + 1$-de qubit van b die bijhoudt wanneer $a + b > 2^n$ dan wel $a - b < 0$. (Immers, door met alleen n bits te rekenen, wordt impliciet modulo 2^n gerekend. Echter, dit is vaak onwenselijk en verstoort berekeningen, waardoor er bijgehouden moet worden wanneer een overschrijding van

2^n of 0 plaatsvindt.) De overdracht-qubits beginnen allemaal als $|0\rangle$. t kan overgenomen worden van eerdere sommeringen (of aftrekkingen) zodat ze omkeerbaar zijn, maar begint anders ook als $|0\rangle$.

2. Bekijk de qubits c_k, a_k, b_k en c_{k+1}. Wetende dat $c_{k+1} \mapsto |1\rangle$ als minstens twee van c_k, a_k, b_k in het $|1\rangle$-toestand zijn, kan het volgende logica worden geïmplementeerd:

   $CCNOT(a_k, b_k, c_{k+1})$ (a_k en b_k zijn hier controle)
   $CNOT(a_k, b_k)$ (a_k is hier controle)
   $CCNOT(c_k, b_k, c_{k+1})$.

   De $CNOT$-bewerking in het midden voorkomt dat de twee $CCNOT$s elkaar opheffen wanneer c_k, a_k, b_k allemaal $|1\rangle$ zijn. Uiteraard, er zal nooit een overdracht zijn bij $a_0 + b_0$, dus bij $k = 0$ kan de laatste $CCNOT$ weggelaten worden gezien c_0 niet bestaat. Door deze circuits uit te voeren voor alle k van 0 t/m $n - 2$ zullen alle overdracht-qubits berekend worden.

3. Met

   $CCNOT(a_{n-1}, b_{n-1}, t)$
   $CNOT(a_{n-1}, b_{n-1})$
   $CCNOT(c_{n-1}, b_{n-1}, t)$

   kan gecontroleerd worden of de grenzen van 2^n en 0 al dan niet overschreden worden.

4. Nu moet de optelling gecompleteerd worden terwijl alle overdracht-qubits teruggebracht worden naar hun oorspronkelijke toestand van $|0\rangle$. Aangezien de overdrachtbits en t nu berekend zijn, moeten alleen nog de qubits van b bewerkt worden. Wanneer $c_k = |1\rangle$, moeten de coëfficiënten van $|0\rangle$ en $|1\rangle$ in b_k omgewisseld worden, en idem als $a_k = |1\rangle$. Dus,

   $CNOT(a_k, b_k)$
   $CNOT(c_k, b_k)$.

   Echter, b_k werden bij het berekenen van c_{k+1} al bewerkt, dus voordat b_k de sommering kan ondergaan, moet c_{k+1} eerst hersteld worden. Dus, telkens worden de instructies

   $CCNOT(c_k, b_k, c_{k+1})$
   $CNOT(a_k, b_k)$
   $CCNOT(a_k, b_k, c_{k+1})$
   $CNOT(a_k, b_k)$
   $CNOT(c_k, b_k)$

   uitgevoerd: eerst de berekening van c_{k+1} maar in omgekeerde volgorde, dan de berekening van b_k. Dit geldt voor de k van $n - 2$ t/m 1 (zelfs deze volgorde moet omgekeerd zijn als hiervoor; c_0 bestaat niet en b_0 is al goed), maar voor $k = n - 1$ is iets anders van toepassing. Hierbij moet alleen nog $CNOT(c_{n-1}, b_{n-1})$ gebeuren (voorafgaand aan het herstellen van c_{n-1}), zodat de sommering van a en b compleet is (Vedral et al., 1996).

Het SUM-circuit is helaas heel lang: ieder van de $CNOT$- en $CCNOT$-bewerkingen moeten achter elkaar plaatsvinden. Voor het berekenen van de overdracht-qubits zijn n $CNOT$s en $2n-1$ $CCNOT$s gebruikt. Voor het herstellen en completeren van de sommering zijn $3n-7$ $CNOT$s en $2n-4$ $CCNOT$s gebruikt. In totaal zijn dat $4n-7$ $CNOT$s en $4n-5$ $CCNOT$s, wat uitkomt op een circuitlengte van $152n-204$. Omdat SUM enkel bestaat uit $CNOT$ en $CCNOT$, is $(SUM)^\dagger$ - het omgekeerde van SUM - precies alle instructies van SUM van achteren naar voren.

Volgens het ontwerp hierboven is SUM een circuit met (a,b) als input en $(a,a+b)$ als output. Nu kan de volgende logica gebruikt worden om van de output $(a,a+b \mod N)$ te maken: "Bereken $a+b-N$. Als dat kleiner is dan 0 (te zien aan of t in het $|1\rangle$-toestand komt), kan geconcludeerd worden dat $a+b<N$ en $a+b=(a+b \mod N)$. Echter, als het positief is, geldt zeker $a+b-N=(a+b \mod N)$. Immers, a en b zijn allebei kleiner dan N; ze komen namelijk altijd uit eerdere berekeningen modulo N, dus $a+b<2N$." Geschreven als algoritmische instructies ziet dat er zo uit:

1. Definieer een register `reg_k` van n qubits om $|N\rangle$ erin op te slaan als qubits die volledig in ofwel het $|0\rangle$-, dan wel het $|1\rangle$-toestand zitten. Hier staat '`k`' voor klassiek, want de qubits in dit register zitten nooit netto in superpositie; ze worden altijd teruggebracht naar een toestand van alleen nul of één na een circuit. Het teken-qubit is doorheen dit circuit in hetzelfde positie t en wordt steeds overgenomen van eerdere SUMs of $(SUM)^\dagger$s. Definieer daarnaast nog een controle-qubit c met $|c\rangle=|1\rangle$ aan het begin.

2. Noem `reg_a` en `reg_b` de n-qubit registers met toestanden respectievelijk $|a\rangle$ en $|b\rangle$. Voer SUM uit op `reg_a` en `reg_b`. Nu bevat `reg_b` $|a+b\rangle$.

3. Voer $(SUM)^\dagger$ uit op `reg_k` en `reg_b`, waarbij t overgenomen wordt van de vorige SUM, waardoor `reg_b` $|a+b-N\rangle$ (zonder verlies van informatie vanwege overschrijding van 2^n) wordt. $|t\rangle$ is nu $|0\rangle$ dan en slechts dan als $a+b-N<0$.

4. Voer $CNOT(t,c)$ uit, waardoor $|c\rangle=|1\rangle$ als $|t\rangle=|0\rangle$. c is zelf een controle voor een aantal $CNOT$s die de (qu)bits van N allemaal gelijk aan nul maken wanneer $|c\rangle=|1\rangle$. Hierdoor kan c effectief bepalen of `reg_k` $|N\rangle$ bevat, of dat het 0 bevat.

5. Voer SUM uit op `reg_k` en `reg_b`. Als $a+b-N<0$, wordt de aftrekking simpelweg ongedaan gemaakt en keert `reg_b` terug naar $a+b$, omdat dan geldt dat $a+b=(a+b \mod N)$. Als $a+b-N\geq 0$, dan $|c\rangle=|1\rangle$ en bevat `reg_k` $|0\rangle$, dus na optelling blijft `reg_b` onveranderd bij $|a+b-N\rangle$. In beide gevallen bevat `reg_b` nu de waarde van $|a+b \mod N\rangle$.

6. Herstel met dezelfde $CNOT$s als in stap 4 de waarde van `reg_k` tot N. (Dit gebeurt dan alleen als $|c\rangle$ toen $|1\rangle$ was.)

7. Voer nu $(SUM)^\dagger$ uit op `reg_a` en `reg_b`, zodat de waarde van `reg_b` ofwel $|b\rangle$, dan wel $|b-N\rangle$ is. Voer achtereenvolgens $NOT(t),CNOT(t,c),NOT(t)$ uit om c te

herstellen tot $|1\rangle$: in het geval dat `reg_b` $|b\rangle$ bevat was $a+b-N$ groter dan 0 en is $|t\rangle$ op het moment van de $CNOT$ gelijk aan $|1\rangle$, waardoor de reeds omgekeerde c teruggaat naar $|1\rangle$; in het geval dat `reg_b` $|b-N\rangle$ bevat was $a+b-N$ kleiner dan 0 en is $|t\rangle$ op het moment van de $CNOT$ gelijk aan $|0\rangle$, waardoor niks gebeurt met de onaangetaste c. In beide gevallen is de toestand van c aan het einde $|1\rangle$.

8. Voer een laatste keer SUM uit op `reg_a` en `reg_b`. Nu geldt dat `reg_a` nog steeds $|a\rangle$ bevat, maar `reg_b` bevat $|a+b \mod N\rangle$, zoals gewenst.

Noem dit nieuwe circuit SMD. De lengte van SMD is nog groter. Het heeft gebruikgemaakt van 5 SUM- of $(SUM)^\dagger$-circuits, en daarbovenop nog even veel $CNOT$s als twee keer het aantal enen in de binaire voorstelling van N (te schatten als circa n $CNOT$s in totaal) om `reg_k` te verwerken, en 2 extra $CNOT$s plus 2 NOTs om t en c te verwerken. Al met al is de lengte van SMD ongeveer gelijk aan $5(152n-204)+n+11 = 761n-1009$.

De CV-circuits zijn een speciaal geval van SMD. $CV_{k,l}$ heeft twee inputregisters `reg_a` en `reg_b` - net zoals SMD, maar ook twee controle-bits c' en m_l, met als extra eis dat `reg_a` in het nultoestand verkeert. Met een reeks aan $CCNOT$s met c' en m_l als controles kan `reg_a` de waarde van $|2^l x^{2^k} \mod N\rangle$ gegeven worden wanneer c' en m_l allebei in $|1\rangle$ zitten, mits de waarde van $|2^l x^{2^k} \mod N\rangle$ van tevoren bekend is. Dan kan SMD uitgevoerd worden op `reg_a` en `reg_b` zodat de waarde van $|2^l x^{2^k} \mod N\rangle$ bij `reg_b` opgeteld wordt (wanneer de controle-qubits in $|1\rangle$-toestand zitten), waarna met dezelfde reeks aan $CCNOT$s `reg_a` hersteld wordt tot 0.

Nu kan CU_k geconstrueerd worden. CU_k neemt registers `reg_m` met qubits $m_0, m_1, \ldots, m_{n-1}$ en `reg_b` in nultoestand als inputs, samen met een extra controle-qubit c'. De qubits van `reg_m` en c' spelen dan de rollen van controle-qubits voor de CV-circuits, terwijl `reg_b` ook steeds `reg_b` is voor de CV-circuits. Een extra register `reg_a` komt voor de CV-circuits. Hieruit blijkt dat CU_k bijna niets anders is dan $CV_{k,0}, CV_{k,1}, \ldots, CV_{k,n-1}$ achter elkaar. Het heeft maar één extra component: wanneer $|c'\rangle = |0\rangle$, moet `reg_m` gekopiëerd worden naar `reg_b` om de constructie van het volledige circuit te vergemakkelijken. Hiervoor voert men $NOT(c')$ uit en dan een reeks aan n $CCNOT$s met telkens c' en een qubit van `reg_m` als controles, zodat de $|0\rangle$ in `reg_b` een $|1\rangle$ wordt precies dan als de overeenkomstige qubit in `reg_m` een $|1\rangle$ is. Daarna moet $NOT(c')$ nog een keer uitgevoerd worden om de waarde van c' te herstellen.

De reeks aan CV-circuits bevat n SMDs en circa $\frac{2n^2+n}{2} = n^2 + \frac{n}{2}$ $CCNOT$s, dus de circuitlengte van CU_k is $n(761n-1009) + n^2 + \frac{n}{2} \approx 762n^2 - 1008n$. Merk op dat - omdat alleen NOTs, $CNOT$s en $CCNOT$s tot nu toe gebruikt werden - alle circuits omkeerbaar zijn door simpelweg van achteren te beginnen.

De laatste constructie voegt alle CU-circuits samen. Bekijk de volgende constructie:

Definieer een register `reg_y` met qubits $y_0, y_1, \ldots, y_{n-1}$ die overeenkomen met de binaire

cijfers van y (de macht waarvoor ($x^y \mod N$) berekend moest worden). Definieer daarnaast registers `reg_x1` en `reg_x2`, allebei met n qubits, waarbij `reg_x1` begint in $|1\rangle$ en `reg_x2` begint in $|0\rangle$.
Gebruik telkens x_k als controle-(qu)bit (ze verkeren eigenlijk niet in superpositie en zijn effectief klassieke bits) voor CU_k en $(CU_k)^\dagger$.
Gebruik `reg_x1` als `reg_m` en `reg_x2` als `reg_b` voor CU_k wanneer k even is,
en gebruik `reg_x2` als `reg_m` en `reg_x1` als `reg_b` (andersom) wanneer k oneven is.
Gebruik `reg_x2` als `reg_m` en `reg_x1` als `reg_b` voor $(CU_k)^\dagger$ wanneer k even is,
en gebruik `reg_x1` als `reg_m` en `reg_x2` (andersom) als `reg_b` voor $(CU_k)^\dagger$ wanneer k oneven is.
Voer nu $CU_0, (CU_0)^\dagger, CU_1, (CU_1)^\dagger, \ldots, CU_{n-1}, (CU_{n-1})^\dagger$ in die volgorde uit. Uiteindelijk zit ($x^y \mod N$) in `reg_x2` als n even is, of in `reg_x1` als n oneven is. Waarom?

Het effect van $(CU_k)^\dagger$ is eigenlijk dat het register dat als `reg_m` gebruikt werd, onveranderd blijft, terwijl het andere register vermenigvuldigd wordt met de multiplicatieve inverse $x^{-y_k 2^k}$ modulo N. Die inverse bestaat, omdat x copriem is met N en daarmee alle x-machten ook.
Stel (zonder verlies van algemeenheid) dat `reg_x1` de inputregister `reg_m` was voor CU_k, het circuit dat voor zijn inverse afgaat. Na CU_k is `reg_x1` onveranderd - noem zijn waarde $|w\rangle$ - en `reg_x2` is $|0 + wx^{y_k 2^k}\rangle = |wx^{y_k 2^k}\rangle$. Nu draaien de rollen om voor $(CU_k)^\dagger$: $|wx^{y_k 2^k}\rangle$ blijft deze keer onveranderd en `reg_x1` wordt $|w - wx^{y_k 2^k} \cdot x^{-y_k 2^k}\rangle = |w - w\rangle = |0\rangle$. Zo krijgt steeds één van de registers een extra "deel" van de hele exponent $x^{y_k 2^k}$, maar uiteindelijk zijn de twee registers (tot op volgorde na) $|x^y \mod N\rangle$ en $|0\rangle$.

Aangezien hier $2n$ CU-circuits gebruikt worden, is de totale circuitlengte van machtsverheffen modulo N ongeveer $1524n^3 - 2016n^2$. Zoals verwacht, geldt $O(1524n^3 - 2016n^2) = O(n^3)$.

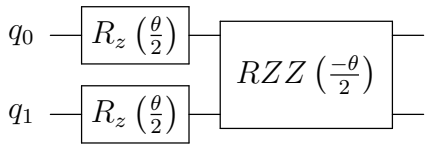
Analyse van het QFT-circuit

De QFT voor t qubits wordt opgebouwd zoals staat afgebeeld op pagina 117 van (Kaye et al., 2006). De R_k-bewerkingen met controle-qubit zijn speciale gevallen van $CR(\theta)$:

$$CR(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix}$$

die $|11\rangle$ van een systeem van twee qubits vermenigvuldigt met $e^{i\theta}$ (Kaye et al., 2006). Dit matrix kan worden opgebouwd uit twee $R_z\left(\frac{\theta}{2}\right)$-bewerkingen en de $RZZ\left(\frac{-\theta}{2}\right)$, een hiervoor nog niet genoemde basisbewerking op twee qubits (IBM Quantum Platform, z. j.). Het

circuitdiagram ziet er zo uit:



Uitgeschreven als product van twee matrices:

$$\left(R_z\left(\frac{\theta}{2}\right)\otimes R_z\left(\frac{\theta}{2}\right)\right)\left(RZZ\left(\frac{-\theta}{2}\right)\right) = \left(\begin{pmatrix} e^{i\frac{-\theta}{4}} & 0 \\ 0 & e^{i\frac{\theta}{4}} \end{pmatrix}\otimes\begin{pmatrix} e^{i\frac{-\theta}{4}} & 0 \\ 0 & e^{i\frac{\theta}{4}} \end{pmatrix}\right)\begin{pmatrix} e^{i\frac{\theta}{4}} & 0 & 0 & 0 \\ 0 & e^{i\frac{-\theta}{4}} & 0 & 0 \\ 0 & 0 & e^{i\frac{-\theta}{4}} & 0 \\ 0 & 0 & 0 & e^{i\frac{\theta}{4}} \end{pmatrix}$$

$$= \begin{pmatrix} e^{i\frac{-\theta}{2}} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\frac{\theta}{2}} \end{pmatrix}\begin{pmatrix} e^{i\frac{\theta}{4}} & 0 & 0 & 0 \\ 0 & e^{i\frac{-\theta}{4}} & 0 & 0 \\ 0 & 0 & e^{i\frac{-\theta}{4}} & 0 \\ 0 & 0 & 0 & e^{i\frac{\theta}{4}} \end{pmatrix}$$

$$= \begin{pmatrix} e^{i\frac{-\theta}{4}} & 0 & 0 & 0 \\ 0 & e^{i\frac{-\theta}{4}} & 0 & 0 \\ 0 & 0 & e^{i\frac{-\theta}{4}} & 0 \\ 0 & 0 & 0 & e^{i\frac{3\theta}{4}} \end{pmatrix}$$

$$= e^{i\frac{-\theta}{4}}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix} = e^{i\frac{-\theta}{4}}CR(\theta).$$

Wederom geldt dat de factor $e^{i\frac{-\theta}{4}}$ er niet toe doet, omdat het een globaal faseverschil is. Merk op dat de circuitlengte van $CR(\theta)$ maar 2 is, omdat twee R_z-bewerkingen tegelijkertijd gebeuren.

Het ontwerp voor de QFT van Kaye en collega's bevat $(n-1)+(n-2)+\ldots+2+1 = \frac{n(n-1)}{2}$ CR-bewerkingen en n Hadamard-bewerkingen. Deze bewerkingen moeten elkaar in volgorde opvolgen zonder dat twee tegelijkertijd plaats kunnen vinden, dus de circuitlengte van de QFT is $\frac{n^2+n}{2}$, verwaarloosbaar klein ten opzichte van de lengte van machtsverheffen modulo N.

Evenals de QFT zijn de klassieke gedeeltes van de algoritme instantaan vergeleken met machtsverheffen. Zelfs het voorbereiden van de constantes $2^l x^{2^k}$ gaan zeer snel. Computers zijn namelijk optimaal geschikt voor vermenigvuldigingen met 2, want dat is niets anders dan alle bits van een getal een plekje naar links opschuiven (ervan uitgaande dat links meer significant is). Dus, zodra x^{2^k} berekend is, kunnen telkens alle bits daarvan een plek doorgeschoven worden, waarna N ervanaf getrokken moet worden, en herhalen totdat

$2^{n-1}x^{2^k}$ berekend is. De x^{2^k} zijn ook gemakkelijk inductief te berekenen. $x^{2^{k+1}}$ is namelijk gewoon $(x^{2^k})^2$, en kwadrateren modulo N gaat vrij snel. (Onthoud dat de getallen nooit te groot worden; ze blijven binnen de restklasses modulo N.) Om deze redenen kan de tijd dat de algoritme van Shor in beslag neemt, afgeschat worden op $1524n^3 - 2016n^2 + 2n^2 = 1524n^3 - 2014n^2$ keer 68 nanoseconden.

Literatuur

AbuGhanem, M. (2025). IBM quantum computers: evolution, performance, and future directions. *The journal of supercomputing*, *81*, (art. 687). doi: 10.1007/s11227-025-07047-7

Al-Baytay, A. & Perkowski, M. (2024, 10). *Cost-effective realization of n-bit Toffoli gates for IBM quantum computers using the Bloch sphere approach and IBM native gates.* doi: 10.48550/arXiv.2410.13104

Atkin, A. O. L. & Bernstein, D. J. (2004, 4). Prime sieves using binary quadratic forms. *Mathematics of Computation*, *73*(246), 1023-1030. (Verkregen via Sci-Hub) doi: 10.1090/S0025-5718-03-01501-1

Axler, S. (2023). *Linear algebra done right (fourth edition).* Springer Cham. doi: 10.1007/978-3-031-41026-0

Bellare, M. & Rogaway, P. (1995). Optimal asymmetric encryption. In A. De Santis (red.), *Advances in cryptology — eurocrypt'94* (p. 92-111). Berlin, Heidelberg: Springer Berlin Heidelberg.

Daemen, J. & Rijmen, V. (2001). *The design of Rijndael, AES - the Advanced Encryption Standard.* Springer-Verlag. (Verkregen van `https://cs.ru.nl/~joan/papers/JDA_VRI_Rijndael_2002.pdf`) doi: 10.1007/978-3-662-04722-4

Ekerå, M. (2024, 6). On the success probability of quantum order finding. *ACM Transactions on Quantum Computing*, *5*, 1-40. doi: 10.1145/3655026

Goldwasser, S. & Micali, S. (1982). Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the fourteenth annual acm symposium on theory of computing* (p. 365–377). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/800070.802212

IBM. (z. j.). *What is cryptography?* Verkregen van `https://www.ibm.com/think/topics/cryptography`

IBM Quantum Platform. (z. j.). *Qiskit SDK: Circuit library.* `https://quantum.cloud.ibm.com/docs/en/api/qiskit/circuit_library`. (Gelezen op 28/1/2026)

Katzenbeisser, S. (2001). *Recent advances in RSA cryptography.* Springer New York. (Verkregen via Sci-Hub) doi: 10.1007/978-1-4615-1431-2

Kaye, P., Laflamme, R. & Mosca, M. (2006). *An introduction to quantum computing.* Oxford University Press.

Knudsen, L. (1999). Contemporary block ciphers. In I. B. Damgård (red.), *Lectures on data security: Modern cryptology in theory and practice* (p. 105-126). Springer Berlin Heidelberg. (Verkregen via Sci-Hub) doi: 10.1007/3-540-48969-X_5

Moriarty, K. et al. (2016). PKCS #1: RSA cryptography specifications version 2.2. RFC 8017. Verkregen van `https://www.rfc-editor.org/info/rfc8017` doi: 10.17487/RFC8017

Pavlidis, A. & Gizopoulos, D. (2014). Fast quantum modular exponentiation architecture

for Shor's factorization algorithm. *Quantum information and computation*, *14*, 649-682. doi: 10.48550/arXiv.1207.0511

Shannon, C. E. (1949). Communication theory of secrecy systems. *The Bell System Technical Journal*, *28*(4), 656-715. doi: 10.1002/j.1538-7305.1949.tb00928.x

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, *26*, 1484-1509. (Verkregen via Sci-Hub) doi: 10.1137/S0097539795293172

Suetonius. (2025). *The lives of the Caesars*. Penguin Classics. (Vertaald door Holland, T. Zie "Life of Julius Caesar", 56.)

Vedral, V., Barenco, A. & Ekert, A. (1996, 7). Quantum networks for elementary arithmetic operations. *Physical Review A*, *54*, 147-153. doi: 10.48550/arXiv.quant-ph/9511018

Zagier, D. (1997). Newman's short proof of the prime number theorem. *The American Mathematical Monthly*, *104*(8), 705-708. (Verkregen via Sci-Hub) doi: 10.1080/00029890.1997.11990704

Bijlage A. Voorbeelden

Ter verheldering van enkele relatief ingewikkelde processen omschreven in dit werkstuk zijn hieronder voorbeelden gegeven voor hoe die in hun werking gaan. De parameters zijn met opzet heel klein gekozen.

Algoritme van Euclides: berekenen van multiplicatieve inverses
Gegeven zijn $a = 62$ en $m = 231$. Met de Algoritme van Euclides kan berekend worden wat de multiplicatieve inverse van a is modulo m, omdat $\text{ggd}(a, m) = 1$.

$$\begin{aligned} 231 &= 3 \cdot 62 + 45 \\ 62 &= 45 + 17 \\ 45 &= 2 \cdot 17 + 11 \\ 17 &= 11 + 6 \\ 11 &= 6 + 5 \\ 6 &= 5 + 1 \\ 5 &= 5 \cdot 1 + 0. \end{aligned}$$

Met deze vergelijkingen kan 1 uitgedrukt worden als $231x + 62y$:

$$\begin{aligned} 6 = 5 + 1 &\Rightarrow 1 = 6 - 5 \\ 11 = 6 + 5 &\Rightarrow 5 = 11 - 6 \Rightarrow 1 = 6 - (11 - 6) = 2 \cdot 6 - 11 \\ 17 = 11 + 6 &\Rightarrow 6 = 17 - 11 \Rightarrow 1 = 2 \cdot (17 - 11) - 11 = 2 \cdot 17 - 3 \cdot 11 \\ 45 = 2 \cdot 17 + 11 &\Rightarrow 11 = 45 - 2 \cdot 17 \Rightarrow 1 = 2 \cdot 17 - 3 \cdot (45 - 2 \cdot 17) = 8 \cdot 17 - 3 \cdot 45 \\ 62 = 45 + 17 &\Rightarrow 17 = 62 - 45 \Rightarrow 1 = 8 \cdot (62 - 45) - 3 \cdot 45 = 8 \cdot 62 - 11 \cdot 45 \\ 231 = 3 \cdot 62 + 45 &\Rightarrow 45 = 231 - 3 \cdot 62 \Rightarrow 1 = 8 \cdot 62 - 11 \cdot (231 - 3 \cdot 62) = 41 \cdot 62 - 11 \cdot 231. \end{aligned}$$

Merk nu op dat $1 = 41 \cdot 62 - 11 \cdot 231 \equiv 41 \cdot 62 \mod 231$, waaruit volgt dat 41 de multiplicatieve inverse van 62 is modulo 231.

RSA Encryptie en decryptie

1. Alina kiest $p = 7$ en $q = 11$. Daarmee krijgt zij $N = 77$ en $\varphi(N) = (7-1)(11-1) = 60$.
2. $a = 13$ heeft geen delers (behalve 1) gemeen met $\varphi(N)$. Met de algoritme van Euclides berekent zij dat $13 \cdot 37 \equiv 1 \mod \varphi(N)$, dus $b = 37$.
3. Alina stuurt $(77, 13)$ op als public key. Zowel Casper als Boas krijgen dit te weten.
4. (Omdat hier gewerkt wordt met te kleine parameters, kan OAEP - het paddingprotocol - niet gebruikt worden. Een samenvatting van de werking van OAEP volgt na dit stuk.)

5. Boas wil een bericht $M = 29$ sturen. Hij berekent $M^a = 29^{13} \equiv 57 \mod 77$. 55 stuurt hij naar Alina, maar Casper onderschept het ook.

6. Alina berekent $57^b = 57^{37} \equiv 29 \mod 77$, precies het bericht dat Boas heeft gestuurd.

Casper krijgt nu te weten dat $(N, a) = (77, 13)$ en $M^a \equiv 57 \mod N$, maar er is geen algemene manier om N snel te factoriseren, of om een soort 13e-machtswortel te nemen modulo N. Dus, Casper krijgt M niet te weten.

Optimal Asymmetric Encryption Padding

OAEP werkt op basis van een zogenaamde *mask generating function* (maskerfunctie) genaamd MGF. MGF stuurt een *seed* (datgene wat gemaskerd moet worden) gepaard met een natuurlijk getal ℓ naar een output van lengte ℓ, en dat doet MGF zodanig dat het altijd dezelfde output geeft met dezelfde seed en ℓ, maar zonder de seed te weten, is het nagenoeg onmogelijk om die te ontdekken met alleen de output. Hieronder zullen de essentiële delen van OAEP belicht worden:

1. Van tevoren moeten Alina en Boas een lengte voor de gemaskerde seed afspreken - laat die lengte ℓ heten. ℓ moet (aanzienlijk) kleiner zijn aan n, de lengte van de modulus N. Boas moet het bericht M aanvullen totdat het lengte $n - \ell - 1$ is. Daartoe zet hij 0001 voor de bits van M, waarna net zoveel nullen voor de 0001 gezet worden als nodig is om M aan te vullen tot een lengte van $n - \ell - 1$. Het resultaat ziet eruit als $00 \ldots 001M$, noem dit M'.

2. Nu kiest Boas willekeurig een getal als seed. Noem dat getal S. Hij berekent $MGF(S, \ell)$ om S_ℓ te krijgen: ℓ bits die nagenoeg geen informatie weggeven over S. Nu gebruikt hij de XOR-operatie S_ℓ en M' met elkaar te mengen. Dit gaat per bit: als de overeenkomstige bits van S_ℓ en M' hetzelfde zijn, is de output 0. Als ze verschillen van elkaar, is de output 1. Dat wordt genoteerd als $S_\ell \oplus M'$. (N.B.: S_ℓ wordt hiervoor aangevuld met nullen voorin zodat het even lang is als M'.)

3. Als laatste berekent Boas nog $MGF((S_\ell \oplus M'), \ell)$ om S', een masker voor S, te krijgen. De volledige M_{pad} is $(S \oplus S')$ en $(S_\ell \oplus M')$ achter elkaar geplakt met een totale lengte van $\ell + (n - \ell - 1) = n - 1$ bits, dus is M_{pad} gegarandeerd kleiner dan N, zoals vereist.

4. Zodra Alina door middel van ontcijfering M_{pad} te weten krijgt, kan zij M daaruit krijgen. Eerst splitst zij de eerste ℓ bits van M_{pad} eraf (vandaar de van tevoren afgesproken lengte). Dat stuk is $(S \oplus S')$ en het resterende gedeelte is $(S_\ell \oplus M')$. Zij kan S' berekenen door $MGF((S_\ell \oplus M'), \ell)$ uit te voeren, net zoals Boas. Merk nu op dat $(S \oplus S') \oplus S' = S$: de operatie is namelijk associatief en $A \oplus A = 0$ omdat alle bits overeenkomen; $A \oplus 0 = A$ omdat $0 \oplus 0 = 0$ en $1 \oplus 0 = 1$. Alina weet nu S,

waardoor zij $MGF(S, \ell) = S_\ell$ kan berekenen en $(S_\ell \oplus M') \oplus S_\ell = M'$ ontdekt. Deze is niets anders dan $00 \ldots 001M$, dus zij kan gemakkelijk de eerste 1 en alle nullen daarvoor weghalen om M af te lezen.

Stel Casper had het idee dat Boas' bericht M was, dan kan hij zijn vermoeden niet verifiëren door de codetekst van M te berekenen met dezelfde paddingprotocol en (N, a), omdat hij S niet weet. Om S te berekenen, moet Casper $(S_\ell \oplus M')$ weten, terwijl hij dat niet kan weten zonder de codetekst van Boas te kraken, omdat MGF onvoorspelbaar is. OAEP zorgt dus dat RSA geen informatie lekt over M omdat het encryptieprocedure geen stochastisch element bevat (Moriarty et al., 2016).

Algoritme van Shor

TBA