RADBOUD UNIVERSITEIT

BACHELORSCRIPTIE

# De impact van het kwantumalgoritme van Shor op het RSA-algoritme zoals voorgeschreven door NIST

*Auteur:*
Sanne VEENSTRA
(s4305329)

*Begeleiders:*
Assoc. Prof. Wieb BOSMA
Ir. Charlotte RUGERS

# *Samenvatting*

**De impact van het kwantumalgoritme van Shor op het RSA-algoritme zoals voorgeschreven door NIST**

Dit onderzoek licht toe waarom het kwantumalgoritme van Shor impact heeft op het RSA-algoritme en waarom dit ook van invloed is op de NIST standaard voor RSA. Doordat de publieke sleutel in elk van de implementaties van RSA in NIST openbaar is, kan het algoritme van Shor toegepast worden op elke modulus[6].

Met het kwantumcircuit van Beauregard kan berekend worden wat de theoretische complexiteit en de praktische complexiteit van het algoritme van Shor is[1]. De theoretische complexiteit $O(32L^3)$ met $L$ de bit-lengte van de modulus, geeft slechts een asymptotische grens van het aantal benodigde bit-operaties[13]. De praktische complexiteit $2L$ geeft een goede indicatie van het aantal kwantumbits dat nodig is om het algoritme van Shor uit te voeren.

# Inhoudsopgave

*Voor mijn hond Harrie*

# Hoofdstuk 1

# Inleiding

## 1.1 Achtergrond

Het bestaan van de kwantumcomputer is de afgelopen jaren gegroeid van fabel naar feit. Er wordt ontzettend veel onderzoek naar verricht en er werden ook veel artikelen over geschreven. Deze interesse is niet iets van de laatste tijd. Onderzoek naar de kwantumcomputer begon in de jaren 1970 nadat er voor het eerst gepubliceerd werd over kwantum-informatie theorie[14]. In de jaren 1980 werd er veel onderzoek gedaan naar de bouw van een kwantumcomputer en in de jaren 1990 werden er veel kwantumalgoritmes ontdekt waarvan één algoritme grote zorgen wekte voor de veiligheid van cryptosystemen. Dit algoritme zorgde daarnaast voor een grote interesse in de kwantumcomputer en in 1998 werd dan ook al de eerste (twee kwantumbit-grote) werkende kwantumcomputer gedemonstreerd. In de jaren die hierop volgde werd er veel geëxperimenteerd met kwantumcomputers zowel in kwantiteit als kwaliteit. Maar tot op heden is er nog geen kwantumcomputer gebouwd die krachtig genoeg is om de veiligheid van cryptosystemen in gevaar te brengen.

## 1.2 Onderzoeksvragen

Om te concluderen wat voor impact het algoritme van Shor heeft op de implementaties van RSA in NIST, moet er eerst een aantal deelvragen worden beantwoord:

1. Wat is het verschil tussen de klassieke computer en de kwantumcomputer?
2. Wat is het verschil tussen de implementaties van RSA voorgeschreven door NIST?
3. Waarom heeft het algoritme van Shor impact op deze implementaties van RSA in NIST?
4. Hoe werkt het algoritme van Shor?
5. Wat is ervoor nodig om het algoritme van Shor te laten werken op een kwantumcomputer?
6. Hoeveel tijd heeft een kwantumcomputer nodig om het algoritme van Shor uit te voeren?

In hoofdstuk 2 zal er een theoretische achtergrond geschetst worden met behulp van klassieke en kwantum definities. Dit zal een antwoord geven op onderzoeksvraag 1.

In hoofdstuk 3 zal het RSA-algoritme beschreven worden dat wereldwijd gebruikt wordt om veilig te kunnen communiceren. De veiligheid van dit algoritme berust zich op de complexiteit van het factoriseren van een getal dat bestaat uit de vermenigvuldiging van twee grote en verschillende priemgetallen. Een priemgetal

is een geheel positief getal groter dan 1, dat alleen deelbaar is door 1 en zichzelf en kan met deze bijzondere eigenschap daarom ook veel betekenen voor de veiligheid van het algoritme. In dit hoofdstuk worden ook de verschillende implementaties van RSA in de NIST standaard beschreven en dus zal dit een antwoord geven op onderzoeksvraag 2.

In hoofdstuk 4 wordt er een antwoord gegegeven op de onderzoeksvragen 3 en 4 door uit te leggen hoe het algoritme van Shor in zijn werk gaat. Dit wordt nader toegelicht door middel van een voorbeeld.

Tenslotte wordt er in hoofdstuk 5 nadruk gelegd op het verschil tussen de klassieke computer en de kwantumcomputer en daarmee een antwoord geformuleerd op de onderzoeksvragen 5 en 6.

*Opmerking*: Aan het einde van elk hoofdstuk wordt de bijbehorende onderzoeksvraag beantwoord.

# Hoofdstuk 2

# Theoretische achtergrond

In de volgende hoofdstukken worden er bepaalde termen uit de natuurkunde en wiskunde gebruikt die niet altijd vanzelfsprekend zijn. Hieronder worden deze notaties, functies en begrippen kort toegelicht om een beter beeld te kunnen geven van de desbetreffende onderwerpen.

## 2.1 Klassieke definities

### Kettingbreukalgoritme

Het doel van het *kettingbreukalgoritme* is om een reëel getal om te schrijven naar een zogenoemde kettingbreuk waar alleen gehele getallen in voorkomen[8]. Dit is het beste te begrijpen door middel van een voorbeeld.
Bekijk de breuk $\frac{71}{17}$. Als eerste wordt deze breuk opgesplitst in een geheel getal en een kleinere breuk,

$$\frac{71}{17} = 4 + \frac{3}{17}.$$

Deze wordt vervolgens geïnverteerd,

$$\frac{71}{17} = 4 + \frac{1}{\frac{17}{3}}.$$

Dit splitsen en inverteren wordt herhaald totdat er in de breuk de combinatie $1 + \frac{1}{n}$ voor $n \geq 2$ overblijft,

$$\frac{71}{17} = 4 + \frac{1}{5 + \frac{2}{3}} = 4 + \frac{1}{5 + \frac{1}{\frac{3}{2}}} = 4 + \frac{1}{5 + \frac{1}{1 + \frac{1}{2}}}.$$

Dit kettingbreukalgoritme wordt in het kwantumalgoritme van Shor gebruikt, wat in hoofdstuk 4 verder toegelicht wordt.

### GGD en KGV

De *grootste gemene deler* (ggd) is het grootste gehele getal dat twee (positieve) gehele getallen deelt. Bekijk de getallen 42 en 70. Dan,

$$\text{ggd}(42, 70) = \text{ggd}(2 \cdot 3 \cdot 7, 2 \cdot 5 \cdot 7) = 14.$$

Het *kleinste gemeenschappelijke veelvoud* (kgv) is de kleinste gehele getal dat zowel een veelvoud is van het ene als het andere getal. Bekijk weer de getallen 42 en 70. Dan,

$$\text{kgv}(42, 70) = \text{kgv}(2 \cdot 3 \cdot 7, 2 \cdot 5 \cdot 7) = 2 \cdot 3 \cdot 5 \cdot 7 = 210.$$

Om de ggd en kgv van een getal efficiënt te berekenen is het niet per se van belang dat de priemfactoren van dit getal bekend zijn. De ggd en kgv kan ook berekend worden door middel van het Euclidisch algoritme. Voor uitleg van dit algoritme wordt naar de literatuur verwezen.

### Indicator $\varphi$

De *Eulerindicator* of *-totiënt* van een positief geheel getal $n \in \mathbb{N}$ geeft het aantal getallen $i$ $(0 < i \leq n)$ dat onderling ondeelbaar is met $n$. Twee getallen $a, b \in \mathbb{N}$ zijn onderling ondeelbaar als ggd$(a, b) = 1$. Voor een priemgetal $p$ geldt dat alle getallen $1, \ldots, p-1$ onderling ondeelbaar zijn met $p$ en dus geldt

$$\varphi(p) = p - 1.$$

Hieruit volgt ook dat

$$\varphi(p^k) = p^k - p^{k-1}.$$

Dit geldt doordat alleen de getallen $p$, $2p$, $3p$, $\ldots$, $p^{k-1}p$ een factor met $p^k$ gemeen hebben.
Stel nu dat $n = p \cdot q$ met $p$ en $q$ verschillende priemgetallen. Dan zijn de enige getallen die niet onderling ondeelbaar zijn met $n$, de getallen die deelbaar zijn door $p$ of $q$. Dit zijn alle veelvouden van $p$ en alle veelvouden van $q$ waarbij het getal $pq$ dubbel geteld wordt ($pq = qp$). Dan volgt dat

$$\varphi(n) = \varphi(pq) = pq - p - q + 1 = (p-1)(q-1).$$

Dit is eenvoudig te generaliseren tot de multiplicatieve eigenschap.
Uit bovenstaande vergelijkingen volgt nu de algemene formule voor de Eulerindicator:

$$\varphi(n) = n \prod_{p|n} (1 - \frac{1}{p}).$$

Bekijk als voorbeeld $n = 10$. Dan $\varphi(10) = \varphi(2 \cdot 5) = \varphi(2) \cdot \varphi(5) = 1 \cdot 4 = 4$. Deze vier getallen zijn 1, 3, 7 en 9.

Zonder de priemfactoren van $n$ te weten, is het niet mogelijk om $\varphi(n)$ te berekenen. Dit is een belangrijke eigenschap die gebruikt wordt in het RSA-algoritme (hoofdstuk 3).

## 2.2 Kwantum definities

### Dirac-notatie

Een kwantumbit wordt vaak in de *Dirac*-notatie geschreven[8]. Zoals een klassieke bit zich in de staat 0 of 1 kan bevinden, bevindt een kwantumbit zich met een bepaalde kansamplitude in de staat $|0\rangle$ en $|1\rangle$. Deze lineaire combinatie van $|0\rangle$ en $|1\rangle$ heet een *superpositie*. Dit is van de vorm

$$|\varphi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

met $\alpha$ en $\beta$ de kansamplitudes waarvoor moet gelden dat $|\alpha|^2 + |\beta|^2 = 1$. Deze kansamplitude is een complex getal dat de kans representeert dat een kwantumbit zich na een meting in de staat $|0\rangle$ of $|1\rangle$ bevindt. Een kwantumbit $|i\rangle$ met $i \in \mathbb{N}$

bevindt zich in de *beginstaat* als er nog geen operatie op uitgevoerd is. Deze beginstaat kan lengte 1 hebben, maar ook lengte n[1]. Dit wordt genoteerd als $|i\rangle$ met $i \in \{0,1\}^n$ of als $|i\rangle^{\otimes n}$. Een kwantumbit kan ook geschreven worden in vectorvorm. Zo kan $|0\rangle$ geschreven worden als $\binom{1}{0}$, $|1\rangle$ als $\binom{0}{1}$ en $|\varphi\rangle$ in het voorbeeld hierboven als $\binom{\alpha}{\beta}$.

### Hadamardtransformatie

Een kwantumpoort is een unitaire transformatie op een klein aantal kwantumbits. Een belangrijke kwantumpoort die in het algoritme van Shor gebruikt wordt is de *Hadamardtransformatie*[8]. Deze transformatie brengt de kwantumbits in superpositie en wordt dus vaak als eerste in een algoritme toegepast.

De twee-dimensionale Hadamardtransformatie is van de vorm $H = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right)$. Om te bewijzen dat deze matrix unitair is moet gelden dat $H^*H = HH^* = I$. Nu:

$$H^*H = HH^* = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} = \frac{1}{2}\begin{pmatrix}2 & 0\\ 0 & 2\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} = I.$$

Dus $H$ is unitair. Dit geldt ook voor de n-dimensionale Hadamardtransformatie.

De Hadamardtransformatie toegepast op een enkel kwantumbit ziet er als volgt uit:

$$H|0\rangle = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 1\end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$H|1\rangle = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0\\ 1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ -1\end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

Het is echter van belang dat deze Hadamardtransformatie op meerdere kwantumbits kan worden toegepast. Stel een kwantumbit bevindt zich in de beginstaat $|i\rangle$ met $i \in \{0,1\}^n$. De Hadamardtransformatie toegepast op deze staat geeft:

$$H^{\otimes n}|i\rangle = \frac{1}{\sqrt{2^n}}\sum_{j\in\{0,1\}^n}(-1)^{i\cdot j}|j\rangle$$

Neem bijvoorbeeld $|0\rangle^{\otimes n}$, dan

$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}}\sum_{j\in\{0,1\}^n}(-1)^{0\cdot j}|j\rangle = \frac{1}{\sqrt{2^n}}\sum_{j\in\{0,1\}^n}|j\rangle$$

### Kwantum Fouriertransformatie

Een goede manier om een wiskundig probleem op te lossen, is om dit te transformeren in een probleem waarvan de oplossing al bekend is. Een transformatie met deze eigenschap is de *Fouriertransformatie*.

Bekijk het probleem waarbij twee polynomen met elkaar worden vermenigvuldigd. Doordat de termen van deze twee polynomen niet paarsgewijs met elkaar vermenigvuldigd kunnen worden, resteert een zeer grote berekening. Om dit probleem te vereenvoudigen kunnen de twee polynomen in punten worden vastgelegd door

[1] $|i_1 i_2 \dots i_n\rangle$ met $i_j \in \{0,1\}$.

middel van de Fouriertransformatie[8]:

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j (\zeta_N^k)^j$$

waarbij $\zeta_N = e^{\frac{2\pi i}{N}}$ $(\star)$, $y_k$ de waarden van het polynoom in een punt en $N$ de graad van het polynoom. Als gevolg kunnen deze nieuwe sets punten wel paarsgewijs met elkaar worden vermenivuldigd en is de vermenigvuldiging makkelijker te berekenen. Door middel van de *inverse Fouriertransformatie* kan de vermenigvuldiging weer worden omgezet in een nieuwe rij coëfficiënten om zo weer een polynoom te construeren.

$(\star)$ De term $e^{\frac{2\pi i}{N}}$ representeert de eenheidscirkel die in $N$ gelijke stukken verdeeld is, de variabele $k$ geeft weer hoeveel 'stukjes' eenheidscirkel de term in de som omvat en door $\zeta_N^k$ tot de variabele $j = 0, \ldots N-1$ te verheffen zijn alle $N$ punten op de eenheidscirkel bevat in de som. De term $x_j$ zorgt ervoor dat al deze berekeningen gewogen worden in de som. Anders zou deze som gelijk zijn aan 1.

Een voorbeeld ter verduidelijking: bekijk de eenheidscirkel met $N = 3$ punten en kies het tweede punt $\zeta_3^2$ op de cirkel. Nu is $j = 0, 1, 2$ en dit geeft:
$(\zeta_3^2)^0 = 1 = \zeta_3^0$
$(\zeta_3^2)^1 = \zeta_3^2$
$(\zeta_3^2)^2 = \zeta_3^4 = \zeta_3^1$
en dus zijn alle $\zeta_3^k$ bevat in de som.

Van deze Fouriertransformatie bestaat ook een kwantumvariant, de *kwantum Fouriertransformatie*:

$$|j\rangle \longrightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} (\zeta_N^k)^j |k\rangle .$$

In plaats van op een polynoom wordt deze transformatie toegepast op de kwantumbits van een superpositie.

Belangrijker dan de kwantum Fouriertransformatie is de inverse hiervan. Ook deze wordt gebruikt in het algoritme van Shor. Wanneer een kwantumbit zich in een superpositie bevindt bevat deze superpositie informatie die uitgelezen moet worden door middel van een meting. Deze meting zorgt er echter voor dat alle informatie verloren gaat. Het is daarom van belang een schatting te geven van de informatie in de superpositie. Dit gebeurt door middel van de *inverse kwantum Fouriertransformatie:*

$$\frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{2\pi i \varphi j} |j\rangle |u\rangle \longrightarrow |\widetilde{\varphi}\rangle |u\rangle$$

waarbij $|\widetilde{\varphi}\rangle$ een schatting geeft voor de superpositie $|\varphi\rangle$[2].

Meer informatie over deze kwantum Fouriertransformatie wordt gegeven in [8].

[2]De $|u\rangle$ representeert een eigenwaarde van een unitaire operator $U$. Voor verdere toelichting wordt verwezen naar [8].

*Wat is het verschil tussen de klassieke computer en de kwantumcomputer?*
Zoals een klassieke bit zich in de staat 0 of 1 kan bevinden, bevindt een kwantumbit zich met een bepaalde kansamplitude in de staat $|0\rangle$ en $|1\rangle$[8]. Doordat een kwantumbit in superpositie kan worden gebracht, bestaande uit een lineaire combinatie van nieuwe kwantumbits, kan een kwantumcomputer sneller bit-operaties uitvoeren ten opzichte van een klassieke computer.

# Hoofdstuk 3

# RSA

Rivest–Shamir–Adleman (RSA) is een veelgebruikt crypto-systeem bedacht door Ron Rivest, Adi Shamir, and Leonard Adleman in 1977[10]. Het is een van de eerste zogeheten 'Public-Key' cryptografiesystemen, ook wel asymmetrische crypto genoemd, waarbij een deel van de versleuteling publiekelijk bekend is en een deel geheim.

Er bestaan verschillende implementaties van RSA. In dit hoofdstuk zal als eerste beschreven worden hoe de kern van RSA in elkaar steekt. Vervolgens wordt uitgelegd aan welke eisen uit de NIST standaard RSA moet voldoen om voor een veilige uitwisseling van berichten te zorgen.

## 3.1 Algemeen

RSA is een 'Public-Key' systeem. Dit houdt in dat geheime berichten tussen twee partijen (vanaf nu: Alice en Bob) niet afhangen van een gedeelde geheime sleutel, maar van een publieke sleutel die voor iedereen beschikbaar is. Dat betekent dat er niet eerst een ontmoeting tussen Alice en Bob hoeft plaats te vinden om vertrouwelijk met elkaar te kunnen communiceren.

Alice en Bob hebben beide een publieke en een private sleutel. Deze sleutels zijn van de vorm $(n, e)$ resp. $(n, d)$ met $n$ de modulus en $e$ en $d$ exponenten[10]. Voor de modulus $n = p \cdot q$ geldt dat $p$ en $q$ oneven priemgetallen zijn, waarbij $p \neq q$ en beide onbekend. Ook voor de exponenten $e$ en $d$ gelden een aantal eisen:

1. $1 < e < \varphi(n)$
2. $\text{ggd}(e, \varphi(n)) = 1$
3. $d = e^{-1} \bmod \varphi(n)$

Stel Bob wil Alice een bericht $m$ sturen. Hiervoor moet hij eerst zijn bericht $m$ coderen met de publieke sleutel van Alice:

$$m^e = x \bmod n$$

Dit gecodeerde bericht $x$ stuurt hij naar Alice. Alice kan dit bericht decoderen met haar private sleutel:

$$\begin{aligned} x^d &= (m^e)^d && \bmod n \\ &= (m^e)^{e^{-1}} && \bmod n \\ &= m^{(e \cdot e^{-1})} && \bmod n \\ &= m && \bmod n \end{aligned}$$

Het bericht $m$ moet een positief geheel getal zijn. Dit betekent dat de woorden in een geheim bericht eerst moeten worden omgezet in positieve gehele getallen. Hierover volgt meer in de volgende paragraaf.

Een voorbeeld ter verduidelijking. Neem als priemgetallen $p = 7$ en $q = 11$ en als geheime bericht $m = 27$. Dan is $n = 77$ en $\varphi(n) = 6 \cdot 10 = 60$. Neem als exponent $e = 13$, dan is $d = 13^{-1} \bmod 60 = 37 \bmod 60$. Nu:

$$x = 27^{13} = 48 \bmod 77$$

$x = 48$ is het versleutelde bericht dat nu veilig is om te versturen. Om te controleren dat $m$ het versleutelde bericht is moet de volgende berekening worden uitgevoerd:

$$m = 48^{37} = 27 \bmod 77.$$

Dus $m$ is inderdaad het versleutelde bericht.

De veiligheid van dit algoritme is afhankelijk van de modulus $n$. Zoals hierboven al vermeld is, is $n$ de vermenigvuldiging van twee verschillende oneven priemgetallen, zeg $p$ en $q$. Doordat er vanuit wordt gegaan dat $n$ zeer groot is, zijn de priemgetallen $p$ en $q$ niet openbaar bekend en dus kan $n$ niet efficiënt ontbonden worden in deze twee priemfactoren. Zonder deze $p$ en $q$ kan ook niet het getal $\varphi(n)$ berekend worden, wat nodig is voor de berekening van de private sleutel $(n, d)$.

## 3.2 NIST

[6] De versie van RSA in de paragraaf hierboven is RSA in zijn simpelste vorm. In de praktijk wordt RSA op een complexere manier geïmplementeerd. De NIST standaarden zijn documenten waarin wordt uitgelegd hoe RSA in zijn werk gaat en hoe RSA het veiligst kan worden geïmplementeerd. Er zijn twee primitieven die in elke vorm van RSA, beschreven in de NIST standaard, gebruikt worden: encryptie en decryptie.

### 3.2.1 Encryptie en decryptie

In de NIST standaard wordt onder andere een basisvorm van encryptie voor RSA gegeven: de 'RSA Encryption Primitive' (RSAEP). Deze encryptieprimitieve produceert gecodeerde tekst met behulp van een publieke sleutel.

RSAEP[1]:
**Input:**
1. De publieke sleutel $(n, e)$
2. Een geheel getal $k$ zodat $1 < k < n - 1$

**Algoritme:**
1. $c = k^e \bmod n$

**Output**
1. De gecodeerde tekst $c$ zodat $1 < c < n - 1$

[1]Dit algoritme zal geen ERROR weergeven doordat er vanuit wordt gegaan dat alle variabelen voldoen aan de eisen die zijn opgelegd.

De encryptieprimitieve is vrijwel gelijk aan de codering van RSA, beschreven in paragraaf 3.1. Dit geldt ook voor de decryptieprimitieve: de 'RSA Decryption Primitive' (RSADP). Deze primitieve achterhaalt de originele tekst uit de gecodeerde tekst.

RSADP[2]:
**Input:**
1. De private sleutel $(n, d)$
2. De gecodeerde tekst $c$ met $1 < c < n - 1$

**Algoritme:**
1. $k = c^d \bmod n$

**Output:**
1. Een geheel getal $k$ zodat $1 < k < n - 1$

In het algoritme hierboven is voor het gemak niet vermeld dat de (gecodeerde) tekst nog moet worden omgezet naar een getal en vice versa. Dit gebeurt door middel van '*Byte String to Integer*' (BS2I) en '*Integer-to-Byte String*' (I2BS) converteerders[3]. In deze converteerders spelen vooral de lengte van de modulus $n$ en van het stuk tekst een grote rol. Voordat de converteerders kunnen worden toegepast is het vaak nodig om de tekst in blokken van gelijke grootte te hakken. De I2BS- en BS2I-converteerders worden ook gebruikt in de algoritmes beschreven in de volgende paragraaf.

De primitieven hierboven vormen samen de absolute basis van RSA. In de praktijk wordt RSA echter niet op deze manier toegepast, maar zijn deze primitieven een onderdeel van andere algoritmen. In de volgende paragraaf is beschreven hoe RSA onderdeel uitmaakt van deze algoritmes.

### 3.2.2 Implementatie RSA in NIST

In de NIST standaard staat op verschillende manieren beschreven hoe RSA voor verschillende doeleinden het veiligst geïmplementeerd kan worden. De algoritmes die hierin beschreven zijn heten RSASVE, RSA-OAEP en RSA-KEM-KWS.
Om een duidelijk verschil te omschrijven tussen deze implementaties van RSA, is het belangrijk eerst de encryptie-operaties van deze algoritmes te bekijken.

Het algoritme 'RSA Secret-Value Encapsulation' (RSASVE) genereert een geheime waarde die het geheime bericht verpakt dat partij A naar partij B wil sturen. Deze geheime waarde is afhankelijk van de lengte van de modulus $n$. Met de waarde van deze lengte genereert een 'random bit generator' een willekeurig stuk tekst dat een converteerder vervolgens omzet in een nieuwe waarde. Samen met de publieke sleutel van partij B, zet RSAEP deze nieuwe waarde om in een gecodeerde tekst die partij A veilig naar partij B kan versturen.

RSASVE wordt vooral toegepast in 'key-agreement' schema's. Het doel van een dergelijk schema is om sleutelmateriaal te produceren, zodat twee partijen hierna

[2]Dit algoritme zal geen ERROR weergeven doordat er vanuit wordt gegaan dat alle variabelen voldoen aan de eisen die zijn opgelegd.

[3]De I2BS- en BS2I-converteerders staan beide beschreven in de bijbehorende appendix van de NIST standaard[6].

makkelijker en sneller kunnen communiceren door middel van een symmetrisch algoritme.

In 'RSA with Optimal Asymmetric Encryption Padding' (RSA-OAEP) wordt een geheime waarde in de vorm van een 'data block' geconstrueerd, in plaats van een willekeurige geheime waarde. Dit blok is opgebouwd uit sleutelmateriaal en aanvullende tekst. Anders dan RSASVE, maskeert een 'mask-generation function' de geheime waardes in het blok dat OAEP genereert. Nadat OAEP deze (gemaskeerde) waarde heeft gegenereerd, zet RSAEP deze waarde samen met de publieke sleutel van partij B weer om in een gecodeerde tekst.

RSA-OAEP wordt vooral toegepast in 'key-transport' schema's. Door middel van dit schema stelt partij B het sleutelmateriaal vast dat partij A geselecteerd heeft als gedeelde geheime sleutel. Ook dit heeft als gevolg dat het communiceren tussen de twee partijen makkelijker en sneller gaat. Het verschil tussen de twee schema's is de inbreng in het sleutelmateriaal. In een 'agreement' schema hebben beide partijen invloed op het resulterende sleutelmateriaal. In een 'transport' schema wordt het sleutelmateriaal door één van de twee partijen gekozen.

Het doel van 'RSA-based Key-Encapsulation Mechanism with a Key-Wrapping Scheme' (RSA-KEM-KWS) is om een sleutel van willekeurige lengte veilig te verpakken, zodat de verdere communicatie tussen partij A en partij B afhangt van deze gedeelde sleutel. De sleutel die RSA-KEM-KWS codeert, bestaat uit twee aaneengeschakelde gecodeerde teksten. Deze teksten worden afzonderlijk van elkaar gecodeerd op twee verschillende manieren. Het eerste gedeelte wordt gecodeerd door middel van RSASVE. Het willekeurige stuk tekst dat het algoritme van RSASVE genereert, speelt ook een rol in de codering van het tweede gedeelte. Een speciale methode gebruikt dit willekeurige stuk tekst, samen met twee andere gekozen teksten om er geheim sleutelmateriaal van te maken. Ook RSA-KEM-KWS wordt gebruikt in 'key-transport' schema's.

Het verschil in deze algoritmen is vooral toe te schrijven aan de codering van de geheime tekst die partij A naar partij B wil sturen. Belangrijker is de overeenkomst die ze alledrie hebben: het gebruik van de RSA encryptieprimitieve. In al deze algoritmen is de publieke sleutel $(n, e)$ algemeen bekend en is de modulus $n$ niet vooraf gecodeerd. Dit is ook logisch, aangezien de sleutel waarmee het geheime bericht gecodeerd moet worden publiekelijk moet zijn om RSA te kunnen toepassen.

Hoewel de private sleutel $(n, d)$ ook de modulus $n$ bevat, wordt de decryptieoperatie van deze algoritmes hier niet besproken. In hoofdstuk 4 wordt er gefocust op de publieke sleutel en in het bijzonder de modulus aangezien deze voor iedereen beschikbaar is.

## 3.3 Priemgetallen

De veiligheid van RSA berust op de modulus $n$. Een veilige lengte volgens de NIST standaard voor deze $n$ is 2048 of 3072 bits. Omdat de modulus $n$ opgebouwd is uit twee priemgetallen $p$ en $q$, is het van belang dat er ook eisen verbonden zijn aan deze twee priemgetallen. De eisen waar $p$ en $q$ aan moeten voldoen zijn als volgt:

1. $n$ is het product van twee verschillende oneven priemfactoren $p$ en $q$ die geheim zijn.
2. $e$ is een oneven geheel getal waarvoor moet gelden dat $2^{16} < e < 2^{256}$

3. $p$ en $q$ worden gegenereerd met behulp van één van de methoden in FIPS186[4] zodat:
   a. $(\sqrt{2})(2^{L/2-1}) \leq p \leq (2^{L/2}-1)$
   b. $(\sqrt{2})(2^{L/2-1}) \leq q \leq (2^{L/2}-1)$
   c. $|p-q| > 2^{L/2-100}$
   d. $\text{ggd}(e,\ \text{kgv}(p-1, q-1)) = 1$

   Hierbij is $L$ de lengte van modulus $n$ in bits.
4. $d$ moet voldoen aan:
   a. $(2^{L/2}) < d < \ \text{kgv}(p-1, q-1)$
   b. $d = e^{-1} \bmod \ \text{kgv}(p-1, q-1)$

Merk op dat er een aantal eisen verschillen van de eisen beschreven in paragraaf 3.1. Een voorbeeld ter verduidelijking:
Stel dat $p = 3, q = 7$ en $e = 7$. Nu kan $d$ op twee verschillende manieren berekend worden:
Manier 1: $\varphi(n) = 12$ dus ggd($e, \varphi(n)$) = 1. Dan $d = 7^{-1} \bmod 12 = 7$.
Manier 2: kgv($p-1, q-1$) = 6 dus ggd($e$, kgv($p-1, q-1$)) = 1. Dan
$d = 7^{-}1 \bmod 6 = 1$.
Deze $d's$ zijn niet hetzelfde, al zijn beide manieren goed[10]. Het voordeel van manier 2 is dat kgv($p-1, q-1$) kleiner is waardoor de berekening voor $d$ makkelijker en sneller uit te voeren is. Doordat $\varphi(n)$ deelbaar is door kgv($p-1, q-1$) kunnen beide manieren gebruikt worden voor het berekenen van $d$, mits ze niet door elkaar worden gebruikt. Een extra vereiste voor manier 2 is dat $p-1$ en $q-1$ slechts weinig kleine factoren gemeen hebben (naast de factor 2). Daarom moet ook gelden dat $|p-q| > 2^{L/2-100}$.

De reden dat exponent $d$ groter moet zijn dan $2^{L/2}$ is *Wiener's attack*[15]. Deze aanval gebruikt het kettingbreukalgoritme om de exponent $d$ te onthullen wanneer deze klein gekozen is.

Bekijk de voorwaarden a. en b. op $p$ en $q$ in eis 3. Er geldt dat $n = p \cdot q$. Dit geeft

$$\begin{aligned}(\sqrt{2})^2(2^{L/2-1})^2 \leq p \cdot q < (2^{L/2})^2 \\ 2 \cdot (2^{L/2} \cdot 2^{-1})^2 \leq n < 2^L \\ \frac{1}{2} \cdot 2^L \leq n < 2^L\end{aligned}$$

Er geldt dat het aantal bits van een getal $n$ gelijk is aan $\lceil \log_2(n) \rceil = L$, ofwel $n < 2^L$.

Alle eisen die in dit hoofdstuk genoemd zijn zouden er samen voor moeten zorgen dat RSA, en de implementaties daarvan, garant staat voor een veilige uitwisseling van berichten. Dit is waar voor een klassieke computer maar voor een kwantumcomputer is dit mogelijk niet het geval. Het factoriseren van een grote modulus $n$ op een klassieke computer heeft een te lange tijd nodig om nog van toepassing te zijn. Bijvoorbeeld bij het decoderen van geheime berichten door ongeauthoriseerde gebruikers. Een kwantumcomputer kan hier verandering in brengen.

[4] FIPS[5] (Federal Information Processing Standards) is een van de standaarden van NIST. In deze standaard wordt onder andere beschreven op welke manieren de priemgetallen $p$ en $q$ gegenereerd kunnen worden.

*Wat is het verschil tussen de implementaties van RSA voorgeschreven door NIST?*
Het verschil tussen de implementaties is de codering van de geheime tekst vóórdat de encryptie-primitive van RSA er op toegepast wordt. De overeenkomst tussen de implementaties is echter belangrijker. In alledrie de implementaties is de publieke sleutel $(n, e)$ algemeen bekend en is de modulus $n$ voor iedereen beschikbaar. De lengte voor de modulus $n$ voorgeschreven door NIST is 2048 bits of 3072 bits[6].

# Hoofdstuk 4

# Het algoritme van Shor

Het algoritme van Shor is een factorisatie-algoritme dat door Peter Shor in 1994 ontwikkeld is[12]. Het doel van het algoritme is om een zeer groot oneven getal $n$ in twee verschillende (oneven) priemfactoren te ontbinden.

Het factorisatie-algoritme is een van de bekendste kwantumalgoritmes. Een kwantumalgoritme houdt in dat het alleen praktisch uitvoerbaar is op een kwantumcomputer. Voor priemfactorisatie is dit ook nodig, aangezien een klassieke computer een te lange tijd nodig heeft om een zeer groot getal in priemfactoren te ontbinden, om nog van toepassing te zijn. Het vermoeden is dat de kwantumcomputer met dit algoritme hetzelfde grote getal in een zeer korte tijd kan factoriseren.

## 4.1 Algoritme

Voordat wordt toegelicht hoe het algoritme wordt toegepast op een kwantumcomputer, is het belangrijk om het algoritme eerst theoretisch te bekijken. Het algoritme voor het factoriseren van modulus $n$ gaat als volgt[8]:

**Input:**

1. Een geheel getal $n$ samengesteld uit twee priemgetallen $p$ en $q$ waarbij $p \neq q$.

**Algoritme:**

1. Als $n$ even is, dan is 2 een factor.
2. Als $n = p^q$ met $p \geq 2$ en $q \geq 2$, dan is $p$ een factor[1][2].
3. Kies een willekeurig getal $x$ zodat $1 < x \leq n - 1$. Als $\text{ggd}(x, n) > 1$, dan is $\text{ggd}(x, n)$ een factor.
4. Gebruik het kwantumalgoritme om de orde $r$ te bepalen van $x$ mod $n$.
5. Als $r$ oneven, ga dan terug naar stap 3.
   Als $r$ even maar $x^{r/2} \equiv -1 \bmod n$, ga dan terug naar stap 3.
   Anders: bereken $\text{ggd}(x^{r/2} - 1, n)$ en $\text{ggd}(x^{r/2} + 1, n)$. Als de uitkomsten hiervan niet gelijk zijn aan 1 en $n$, dan zijn dit de factoren van modulus $n$.

**Output:**

1. Twee verschillende priemgetallen $p$ en $q$ zodat $n = p \cdot q$.

Voor stap 4 van bovenstaand algoritme en de veronderstelling dat $n$ een zeer groot getal is, moet er gebruik worden gemaakt van een kwantumcomputer om het factorisatieprobleem binnen afzienbare tijd te berekenen. Het volgende kwantumalgoritme is ook niet uitvoerbaar op een klassieke computer. Dit algoritme wat de

[1] Zie de opmerkingen op bladzijde 15 ter verduidelijking van de manier waarop dit berekend wordt.
[2] In het boek van Nielsen en Chuang wordt er een zwakkere eis gesteld: $p \geq 1$ en $q \geq 2$.

orde $r$ van $x$ mod $n$ bepaald gaat als volgt:

**Input:**

i. De functie $U_{x,n} : |j\rangle |k\rangle \longmapsto |j\rangle |x^j k \bmod n\rangle$.

ii. Het aantal Hadamard transformaties $t = 2L + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon}) \rceil$ met $L$ de lengte in bits van modulus $n$ en $0 \leq \epsilon \leq 1$ de precisie.

**Algoritme:**

i. Start met de beginstaat: $|0\rangle |1\rangle$.

ii. Creëer de superpositie

$$\frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle |1\rangle$$

door middel van $t$ Hadamard transformaties, toegepast op het eerste register.

iii. Pas $U_{x,n}$ toe op het tweede register:

$$\frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle |1\rangle \longmapsto \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle |x^j \bmod n\rangle \approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j/r} |j\rangle |u_s\rangle$$

iv. Pas de inverse Fouriertransformatie toe op het eerste register:

$$\frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j/r} |j\rangle |u_s\rangle \longmapsto \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\widetilde{s/r}\rangle |u_s\rangle$$

v. Meet het eerste register: $|\widetilde{s/r}\rangle$

vi. Pas het kettingbreuk-algoritme toe op $\widetilde{s/r}$. De noemer $r$ is de gezochte orde.

**Output:**

i. Het kleinste gehele getal $r > 0$ zodat $x^r = 1 \bmod n$.

Vervolgens kan nu stap 5 van het algoritme van Shor uitgevoerd worden om de priemfactoren van $n$ te berekenen.

### 4.1.1 Voorbeeld

Ter verduidelijking van het algoritme van Shor volgt hier een uitwerking van de priemfactorisatie van het getal 91.

1. $n = 91$ is geen even getal dus 2 is geen factor.
2. $n = 91$ is niet van de vorm $p^q$ dus bestaat er geen dergelijke factor $p$[3].
3. Kies $x = 4$. Dan $\text{ggd}(4, 91) = 1$ en dus is $\text{ggd}(4, 91)$ geen factor.
4. Gebruik het kwantumalgoritme om de orde $r$ te bepalen van 4 mod 91.

[3]Dit wordt in de opmerkingen hieronder berekend

i. Start met de beginstaat $|0\rangle\,|1\rangle$
ii. Creëer de superpositie:

$$\frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1}|j\rangle\,|1\rangle = \frac{1}{\sqrt{2^{17}}}\sum_{j=0}^{2^{17}-1}|j\rangle\,|1\rangle$$
$$= \frac{1}{\sqrt{2^{17}}}\big(\,|0\rangle + |1\rangle + |2\rangle + |3\rangle + ...\big)\,|1\rangle$$

door middel van 17 Hadamard transformaties, toegepast op het eerste register.
iii. Pas $U_{4,91}$ toe op het tweede register:

$$\frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1}|j\rangle\,|x^j \bmod n\rangle = \frac{1}{\sqrt{2^{17}}}\sum_{j=0}^{2^{17}-1}|j\rangle\,|4^j \bmod 91\rangle$$
$$= \frac{1}{\sqrt{2^{17}}}\big(\,|0\rangle\,|1\rangle + |1\rangle\,|4\rangle + |2\rangle\,|16\rangle + |3\rangle\,|64\rangle$$
$$+ |4\rangle\,|74\rangle + |5\rangle\,|23\rangle + |6\rangle\,|1\rangle + |7\rangle\,|4\rangle + |8\rangle\,|16\rangle + ...\big)$$

De stappen iv tot en met vi zijn niet uitvoerbaar doordat doordat hier een kwantumcomputer voor nodig is. Uit de som in stap iii kan er geconcludeerd worden dan de orde van $x$ mod $n$ gelijk is aan 6. Dit is te herkennen aan de herhaling van getallen in het tweede register. Na zes paar registers is het getal in het tweede register weer hetzelfde. Dit blijft zich herhalen.

5. $r = 6$ is even en $x^{r/2} = 4^3 = 64 \neq -1 \bmod 91$ dus kunnen de factoren worden berekend door middel van de ggd.
ggd$(x^{r/2}-1, n)$ = ggd$(4^3-1, 91)$ = ggd$(63, 91) = 7$, en
ggd$(x^{r/2}+1, n)$ = ggd$(4^3+1, 91)$ = ggd$(65, 91) = 13$.

Er geldt dat $91 = 7 \cdot 13$. Een aantal opmerkingen:

1. In stap 2 van het algoritme van Shor wordt er geconcludeerd dat $n$ niet van de vorm $p^q$ is. Dit gebeurt op de volgende manier.
Bekijk $\lfloor\sqrt{91}\rfloor = 9$, dan hoeven alleen de priemgetallen kleiner dan 9 overwogen te worden. Er geldt:
$2^6 = 64 < 91$ en $2^7 = 128 > 91$,
$3^4 = 81 < 91$ en $3^5 > 91$,
$5^2 = 25 < 91$ en $5^3 > 91$ én
$7^2 = 49 < 91$ en $7^3 > 91$.
Dus $n$ is niet van de vorm $p^q$.
2. In stap ii van het kwantumalgoritme wordt er gebruik gemaakt van 17 Hadamardtransformaties om de kwantumbits in superpositie te brengen. In het kwantumalgoritme staat beschreven dat $t = 2L + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon})\rceil$. Het getal 91 is gelijk aan het binaire getal 1011011 en dus bestaat $n$ uit 7 bits ($L = 7$). Dan $t = 2 \cdot 7 + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon})\rceil = 15 + \lceil \log_2(2 + \frac{1}{2\epsilon})\rceil \leq 15 + \lceil \log_2(2\frac{1}{2})\rceil = 17$. Dus zijn er (maximaal) 17 Hadamardtransformaties nodig.

Nu de theorie achter het algoritme besproken is, kan er gekeken worden naar de toepassing van het algoritme op een kwantumcomputer. Er zal kort besproken worden welke complexiteit het algoritme van Shor heeft en welke input er nodig is om het algoritme uit te voeren. Als laatste zal er een vergelijking worden gemaakt tussen de klassieke computer en kwantumcomputer.

*Waarom heeft het algoritme van Shor impact op deze implementaties van RSA in NIST?*
In het antwoord van onderzoeksvraag 3 is vermeld dat de publieke sleutel $(n, e)$ voor elk van deze implementaties algemeen bekend is en dat de modulus $n$ dus voor iedereen beschikbaar is. Dit betekent dat voor elk van deze $n$ het algoritme van Shor toegepast kan worden.

*Hoe werkt het algoritme van Shor?*
Het algoritme van Shor factoriseert een getal $n$ in twee verschillende priemfactoren $p$ en $q$[8]. Het algoritme is deels klassiek en deels kwantum. Het klassieke gedeelte van het algoritme sluit als eerst uit dat geen van de priemfactoren gelijk is aan 2 en dat $n$ niet te schrijven is als macht van een priemgetal. Vervolgens kiest het algoritme een getal dat onderling ondeelbaar is met $n$ en bepaald door middel van het kwantumgedeelte de orde van dit getal modulo $n$. Nadat deze orde bepaald is kunnen de delers van $n$ op een klassieke manier worden berekend. In paragraaf 4.1 worden de precieze stappen van het algoritme beschreven. Er wordt ook verondersteld dat de modulus $n$ zeer groot is, anders kunnen de priemfactoren ook door middel van een klassiek algoritme gevonden worden[4].

[4]Dit algoritme wordt vermeldt in hoofdstuk 5.

# Hoofdstuk 5

# Complexiteit

Het factoriseren van getallen is voor een klassieke computer erg lastig doordat er een hoop bit-operaties moeten worden uitgevoerd. Een kwantumcomputer kan hier beter mee omgaan en is hier dus sneller mee klaar. Om dit in concrete termen te weergeven spreken we van *complexiteit*.

De complexiteit van een algoritme wordt weergeven door middel van de 'Big $O$-notation'. Deze notatie representeert een asymptotische grens ('worst-case') van het maximaal aantal benodigde bit-operaties, in termen van het aantal bits van de input. Klassieke complexiteit wordt ook wel bitcomplexiteit genoemd omdat de operaties van een algoritme toegepast worden op de bits waar het getal uit bestaat.

## 5.1 Complexiteitsklassen

Computationele problemen kunnen in verschillende complexiteitsklassen onderverdeeld worden. De vier complexiteitsklassen die het meest van belang zijn voor dit onderwerp zijn **P**, **NP**[2], **BPP**[3] en **BQP**[4].

De complexiteitsklasse **P** bevat alle door *polynomiale tijd begrensde problemen*[2]. Dit betekent dat het aantal bit-operaties voor een algoritme $\mathcal{A}$ met input-grootte $N$, begrensd is door $C \cdot N^c$ met constante $C, c \in \mathbb{R}$. Klasse **P** wordt ook wel de klasse genoemd met alle 'makkelijke problemen' waarbij $\mathcal{A}$ een *efficiënt algoritme* is voor het oplossen van het probleem.

De complexiteitsklasse die 'moeilijke problemen' omvat **NP**[2]. Deze klasse bevat alle *non-deterministisch polynomiale tijd begrensde problemen*, ofwel alle problemen waarvoor er een algoritme bestaat die binnen polynomiale tijd kan vaststellen of een gegeven oplossing correct is. Dat betekent dat het algoritme alleen bevestigt of het gevonden antwoord correct is, niet dat het antwoord binnen polynomiale tijd gevonden kan worden. Als het antwoord wel binnen polynomiale tijd gevonden kan worden is het probleem bevat in **P**. Hiermee volgt dat $\mathbf{P} \subseteq \mathbf{NP}$.

Naast de exacte complexiteitsklasse die hierboven beschreven zijn, bestaan er ook probabilistische complexiteitsklassen. Een van deze klassen is **BPP**[3]. Deze klasse bevat alle *probabilistische polynomiale tijd error-begrensde problemen*, ofwel problemen die oplosbaar zijn in polynomiale tijd en het goede antwoord geven met kans $\geq 2/3$. Er geldt dat $\mathbf{P} \subseteq \mathbf{BPP}$ maar er is niet bekend wat de relatie is tussen **BPP** en **NP**.

Omdat **BBP** een klasse is die alleen computationele problemen bevat van klassieke computers, bestaat hier ook een kwantumvariant van. De klasse **BQP** bevat alle probabilistische problemen die in polynomiale tijd oplosbaar zijn op een kwantumcomputer en met kans $\geq 2/3$ het goede antwoord geven[4]. Er geldt dat

**P** $\subseteq$ **BPP** $\subseteq$ **BQP**. Ook voor **BQP** is niet bekend wat de relatie met **NP** is.

Het algoritme van Shor kan voor een samengestelde $n$ in ruwweg $(\log(n))^2$ (kwantum)bit-operaties een oplossing vinden, wat dus polynomiaal is in de lengte van de input[16]. Er is echter geen klassiek algoritme bekend dat het factorisatieprobleem binnen polynomiale tijd kan oplossen. Dit betekent dat het probleem dus bevat is in **NP**. Het vermoeden luidt dat het probleem niet bevat is in **P** maar dit is nog niet bewezen. Het factorisatieprobleem is ook bevat in **BQP** maar het is niet duidelijk of dit ook bevat is in **BPP**.

## 5.2 Theoretische en praktische complexiteit

Er bestaat een verschil tussen theoretische en praktische complexiteit. De theoretische complexiteit van het factorisatieprobleem wordt begrensd door het algoritme van Shor en wordt uitgedrukt in 'Big $O$-notation'. De theoretische complexiteit geeft dus een asymptotische grens aan voor het oplossen van een probleem. Doordat deze asymptotische grens alleen opgaat voor een zeer groot samengesteld getal, is de theoretische complexiteit geen richtlijn voor de praktische complexiteit. Stéphane Beauregard heeft in 2003 een circuit voor het algoritme van Shor beschreven die vooral focust op het minimaliseren van het aantal kwantumbits die hiervoor nodig zijn[1]. De praktische complexiteit van het factorisatieprobleem wordt dan ook beschreven door dit circuit.

Bij het implementeren van het algoritme kan er onderscheid worden gemaakt tussen 'implementation depth' en 'implementation width'. De 'depth' van een algoritme geeft aan wat het grootste pad is wat doorlopen moet worden om een output te krijgen. Het minimaliseren van de 'depth' verkleint dit pad en daarmee de tijd die nodig is om het algoritme te doorlopen. De 'width' van een algoritme geeft aan wat de benodigde input is. Het verkleinen van deze benodigde input zorgt voor efficiëntie doordat het algoritme minder input nodig heeft om tot eenzelfde output te komen. In enig opzicht geven de factoren 'depth' en 'width' dus een implicatie van de theoretische en praktische complexiteit.

Het is niet mogelijk om zowel 'depth' als 'width' te minimaliseren. Dit heeft te maken met 'space/time trade-off'[11]. Deze bewering houdt in dat het minimaliseren van de doorlooptijd van een algoritme gepaard gaat met het vergroten van de ruimte dat een algoritme nodig heeft om uitgevoerd te worden.

Het circuit van Beauregard heeft een 'depth' van $O(32L^3)$ en een 'width' van $2L$ kwantumbits met $L$ het aantal bits van modulus $n$[13]. In de NIST-standaarden worden verschillende lengtes voor $n$ aangeraden wat zorgt voor verschillende 'width'[6]:

| **Lengte $n$ in bits** | **'Width'** |
|---|---|
| 1024 | 2048 |
| 2048 | 4096 |
| 3072 | 6144 |

FIGURE 5.1: 'Width' voor verschillende lengtes van $n$.

De 'depth' van bovenstaande waardes is niet berekend doordat deze mogelijk geen goede indicatie geven voor de theoretische complexiteit van het algoritme. De notatie $O(32L^3)$ omschrijft een polynoom van de vorm $32L^3 + bL^2 + cL + d$. Stel nu dat

$b \gg 32$, dan zal de term $bL^2$ voor kleine $L$ een grotere invloed hebben op de complexiteit dan de term $32L^3$. Nogmaals, de notatie $O(32L^3)$ geeft een asymptotische grens aan en gaat alleen op voor $L$ groot genoeg.

Het circuit van Beauregard is polynomiaal[1]. Dat wil zeggen dat het aantal gates dat nodig is voor het algoritme van Shor een polynoom als functie van de input als bovengrens heeft. Of het circuit ook optimaal is niet per definitie duidelijk. Naast het algoritme van Beauregard bestaan er ook andere circuits die het algoritme van Shor implementeren en in sommige opzichten beter zijn[13]. Waar het circuit van Beauregard zich op focust is het minimaliseren van de benodigde kwantumbits voor het algoritme van Shor, waarbij wordt aangenomen dat er alleen gebruik gemaakt kan worden van elementaire kwantumpoorten[1].

## 5.3 Klassiek vs. Kwantum

Het is nu duidelijk dat het algoritme van Shor deels op een klassieke computer uitvoerbaar is en deels op een kwantumcomputer. Er bestaan echter ook factorisatie-algoritmen die alleen gebruik maken van een klassieke computer. Het meest bekende algoritme dat hiervoor gebruikt wordt is 'Number Field Sieve' (NFS) en heeft een complexiteit van $O(e^{(Lk \log(L)^2)^{1/3}})$ waarbij $L$ de lengte van een getal $n$ in bits en $k = \frac{64}{9}\log(2)$[7]. Een probleem dat bij het gebruik van dit algoritme optreedt is dat een klassieke computer meer dan duizend jaar nodig heeft om een modulus $n$ van lengte 1000 bits te factoriseren. Zie hiervoor de volgende tabel.

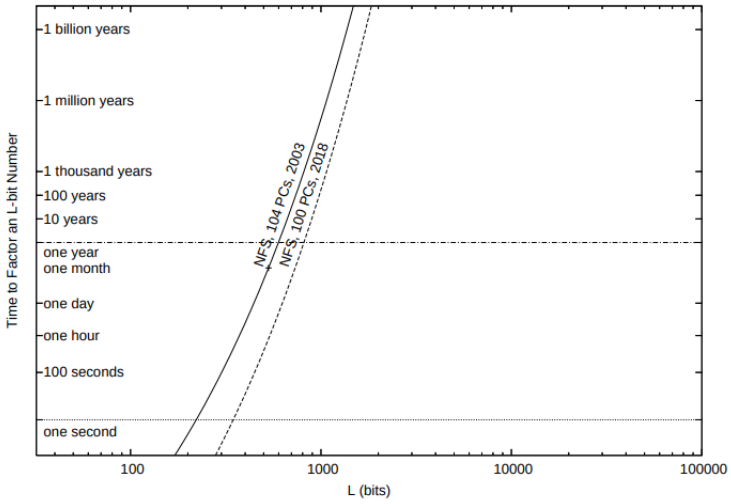


FIGURE 5.2: Een schaling van NFS op klassieke computers. De linkercurve is gebasseerd op prestaties uit 2003 en de rechtercurve is een schatting voor 2018. Beide assen zijn logaritmisch geschaald[7].

Uit berekeningen volgt dat het algoritme van Shor slechts 26,7 uur nodig heeft om een modulus $n$ van lengte 2000 bits te factoriseren[11]. In 2015 heeft M. Mosca de voorspelling gedaan dat er een kwantumcomputer (met een grote fout-tolerantie) beschikbaar is in 2026 met kans $\frac{1}{7}$ en in 2031 met kans $\frac{1}{2}$[9]. Dit wil echter nog niet zeggen dat deze kwantumcomputer dan aan alle eisen voldoet om het algoritme van Shor uit te voeren.

[1] Kwantumpoorten die alleen gebruik maken optellen, aftrekken, vermenigvuldigen en delen.

*Wat is ervoor nodig om het algoritme van Shor te laten werken op een kwantumcomputer?*
Het circuit van Beauregard geeft een indicatie voor zowel de theoretische als praktische complexiteit van het algoritme van Shor[1]. De praktische complexiteit weergeeft het aantal benodigde kwantumbits om het algoritme uit te voeren op een kwantumcomputer. De formule voor het aantal benodigde kwantumbits is $2L$ met $L$ de lengte in bits van de modulus $n$[13]. De meest gebruikte lengte voor $n$ beschreven in NIST is 2048 bits. Om een getal van lengte 2048 bits te factoriseren door middel van het algoritme van Shor zijn er dus 4096 kwantumbits nodig.

*Hoeveel tijd heeft een kwantumcomputer nodig om het algoritme van Shor uit te voeren?*
De theoretische complexiteit weergeeft het maximaal aantal benodigde (kwantum)bit-operaties om het algoritme van Shor uit te voeren. Deze complexiteit is $O(32L^3)$ en is slechts een asymptotische grens voor het aantal benodigde (kwantum)bit-operaties[13]. Dit betekent dat deze grens alleen opgaat voor $L$ groot genoeg.

# Hoofdstuk 6

# Conclusie

In dit onderzoek is er als eerste gekeken naar de verschillen tussen de klassieke computer en de kwantumcomputer. De conclusie die hieruit getrokken kan worden is dat een kwantumcomputer meerdere bit-operaties tegelijk kan uitvoeren en daardoor sneller is. Vervolgens is er gekeken naar de verschillen tussen de verschillende implementaties van RSA beschreven in de NIST standaard. Er is een duidelijk verschil in de codering van de geheime teksten maar ook een belangrijke overeenkomst: elke implementatie maakt gebruik van de RSA encryptieprimitieve. Dat betekent dat in elke implementatie de publieke sleutel bekend is en dus is het voor een derde partij erg makkelijk om de modulus $n$ te achterhalen. Als de modulus $n$ eenmaal bekend is kan het algoritme van Shor toegepast worden. Dit algoritme factoriseert een modulus $n$ in twee verschillende priemfactoren $p$ en $q$. Met de kennis van deze twee factoren kan de exponent $d$ berekend worden waardoor de private sleutel niet meer geheim is.

In de NIST standaard staan een aantal lengtes voorgeschreven voor $n$ waarvan de meestgebruikte de lengte 2048 bits is. Volgens het circuit van Beauregard betekent dit dat de input van het algoritme van Shor gelijk moet zijn aan 4096 kwantumbits. Een andere lengte die ook wordt aangeraden door de NIST standaard is 3072 bits. Het aantal kwantumbits dat hiervoor nodig is om het algoritme van Shor te doorlopen is 6144. Naast het aantal benodigde kwantumbits voor het algoritme van Shor, geeft het circuit van Beauregard ook een asymptotische bovengrens voor het aantal benodigde (kwantum)bit-operaties. Deze grens wordt aangeduid met $O(32L^3)$ en gaat alleen op voor $L$ groot genoeg gekozen.

Een daadwerkelijke kwantumcomputer kan dus hét verschil maken in het 'kraken' van RSA. Zoals een klassieke computer meer dan duizend jaar nodig heeft om een modulus $n$ van lengte 1000 bits te factoriseren, heeft een kwantumcomputer hier slechts 26,7 uur voor nodig. Er is voorspeld dat er in 2026 mogelijk al een kwantumcomputer (met grote fout-tolerantie) bestaat met kans $\frac{1}{7}$. Met deze dreiging op de hielen is het dus belangrijk ook onderzoek te verrichten naar veilig *post-kwantum* communiceren.

# Referenties


[1] S. Beauregard. "Circuit for Shor's algorithm using 2n+3 qubits". In: (2003). URL: https://arxiv.org/abs/quant-ph/0205095v3.

[2] W. Bosma. *Cryptografie*. 2014.

[3] *BPP (complexity)*. URL: https://en.wikipedia.org/wiki/BPP_(complexity).

[4] *BQP*. URL: https://en.wikipedia.org/wiki/BQP.

[5] *Digital Signature Standard (DSS)*. 2013. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf#page=62.

[6] L. Chen en D. Moody E. Barker. *Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography*. 2014. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br1.pdf.

[7] R. Doyle Van Meter III. "Architecture of a Quantum Multicomputer Optimized for Shor's Factoring Algorithm". In: (2008). URL: ArchitectureofaQuantumMulticomputerOptisFactoringAlgorithm.

[8] M.A. Nielsen en I.L. Chuang. *Quantum Computation and Quantum Information*. 2000.

[9] M. Mosca. "Cybersecurity in an era with quantum computers: will we be ready?" In: (2015). URL: https://eprint.iacr.org/2015/1075.pdf.

[10] *RSA (cryptosystem)*. URL: https://en.wikipedia.org/wiki/RSA_(cryptosystem).

[11] C.M. Rugers. "Risk management and the quantum threat". In: (2018). URL: https://homepages.cwi.nl/~schaffne/projects/reports/Rugers_RiskManagement.pdf.

[12] P.W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: (1994). URL: https://arxiv.org/abs/quant-ph/9508027v2.

[13] A.G. Fowler en L.C.L. Hollenberg S.J. Devitt. "Investigating the practical implementation of Shor's algorithm". In: (2005). URL: https://www.researchgate.net/publication/241410201_Investigating_the_practical_implementation_of_Shor%27s_algorithm.

[14] *Timeline of quantum computing*. URL: https://en.wikipedia.org/wiki/Timeline_of_quantum_computing.

[15] *Wiener's attack*. URL: https://en.wikipedia.org/wiki/Wiener%27s_attack.

[16] R. de Wolf. *Quantum Computing: Lecture Notes*. 2011. URL: https://homepages.cwi.nl/~rdewolf/qcnotes.pdf.