Quantumcomputers en Cryptografie

Profielwerkstuk

14-12-2016

Arthur Rump
Praedinius Gymnasium

1. Voorwoord

Voor u ligt het verslag van mijn profielwerkstuk over quantumcomputers en cryptografie. De keuze voor dit onderwerp vloeit voort enerzijds uit mijn persoonlijke interesse voor informatica, anderszijds uit het feit dat dit onderwerp een goede combinatie is van de vakken wiskunde en natuurkunde. Het idee van het profielwerkstuk is ook dat de leerling met een onderwerp aan de slag gaat wat hem aanspreekt en daarnaast een raakvlak heeft met twee gevolgde vakken, in mijn geval zijn dat wiskunde D en natuurkunde.

Quantumcomputers komen vaak in het nieuws en worden dan beschreven als mythische machines, die zo snel zijn in hun berekeningen dat de apparaten die nu in gebruik zijn haast een soort speelgoed worden. In bijna elk artikel wordt ook gesproken over de invloed van quantumcomputers op de hedendaagse cryptografie. De NSA zou bijvoorbeeld een grote belangstelling hebben voor quantumcomputers, omdat ze met een werkende quantumcomputer alle versleutelde gegevens ter wereld zouden kunnen lezen. Spoiler: werkende quantumcomputers bestaan en de NSA kan ook alles lezen zonder deze apparaten. Desalniettemin leek het mij interessant om te onderzoeken hoe een quantumcomputer nou echt werkt en waarom deze machines zoveel efficiënter zouden zijn bij het kraken van versleutelde informatie.

Minder relevant, maar ook interessant: de juiste spelling van 'quantumcomputer' is enigszins discutabel. De juiste Nederlandse spelling lijkt 'kwantumcomputer' te zijn, hier geeft Wikipedia ook duidelijk de voorkeur aan. De rest van het internet is echter meer verdeeld, zo worden bij bijvoorbeeld de NOS en NU.nl beide spellingen door elkaar gebruikt en op de website van het populair wetenschappelijk tijdschrift NewScientist is in de nieuwere artikelen de spelling 'quantumcomputer' te lezen. Ik heb in dit verslag gekozen voor de spelling 'quantumcomputer', met name omdat ik deze spelling mooier vind.

In dit verslag zal uiteraard een hoop wiskunde voorbij komen, maar ik heb geprobeerd het zo begrijpelijk mogelijk te houden. Voor zowel quantumcomputers als cryptografie is wiskunde nodig die niet onder de middelbareschoolwiskunde, met uitzondering van wiskunde D, valt. Van deze onderwerpen is in hoofdstuk 4 een samenvatting te vinden, als ook verwijzingen naar uitgebreidere uitleg. Ik hoop dat dit verslag hierdoor ook voor mensen die geen wiskunde D (gehad) hebben te begrijpen is.

Ten slotte wil ik nog van de gelegenheid gebruik maken om een aantal mensen te bedanken: Daan Leijen van Microsoft Research voor het maken van Madoko.net, ik moet er niet aan denken om alles zelf in LaTeX te schrijven; het IBM Quantum team voor het beschikbaar stellen van hun quantumcomputer aan de hele wereld, echt geweldig dat dit mogelijk is; en natuurlijk mijn begeleiders dhr. Septer en mevr. Stadermann, ik heb dan wel weinig gebruik van jullie gemaakt, toch bedankt!

Dan rest mij nu niets anders dan u veel plezier te wensen met het lezen van dit verslag.

- Arthur Rump

2. Samenvatting

Kan een quantumcomputer de hedendaagse cryptografie veel efficiënter kraken dan een normale computer?

De meest gebruikte cryptografie bestaat uit twee algoritmes: RSA en AES. Deze twee verschillen nogal van elkaar. RSA is een asymmetrisch algoritme, wat betekent dat de sleutel die gebruikt wordt om gegevens te versleutelen een andere is dan die gebruikt wordt voor het ontsleutelen. Belangrijk hierbij is dat ook als een van de sleutels bekend is, de andere niet makkelijk af te leiden mag zijn. In RSA is dit gerealiseerd met behulp van het feit dat het makkelijk is om twee priemgetallen te vermenigvuldigen, maar heel moeilijk om die twee priemgetallen te vinden als alleen de vermenigvuldiging bekend is.

AES is een bloksversleuteling en kan alleen blokken van 128 bits versleutelen. Dit gebeurt door meerdere keren een XOR uit te voeren tussen de sleutel en de data, waarbij intussen de bits ook nog systematisch door elkaar gehusseld worden. Wiskundig gezien is AES dus veel simpeler dan RSA, maar een volledige beschrijving kost wel meer tijd.

Quantumcomputers werken met qubits in plaats van gewone bits. Deze qubits kunnen 0, 1, of een superpositie van deze twee zijn. Een superpositie houdt in dat de qubit in een staat tussen 0 en 1 zweeft. Zodra er echter gemeten wordt in welke staat de qubit is, valt deze weer terug naar 0 of 1. Als alle qubits gemeten worden kan een quantumcomputer dus net zoveel informatie bevatten als een normale computer met hetzelfde aantal bits. In de tussenstappen, voor de meting, is er door de superposities wel meer ruimte, waardoor voor sommige problemen speciale quantumalgoritmes ontworpen kunnen worden, waarmee deze problemen efficiënter opgelost kunnen worden dan met een gewone computer.

Een voorbeeld hiervan is het algoritme van Shor voor het vinden van de priemfactoren van een getal. Dit quantumalgoritme is beduidend efficiënter dan het beste algoritme voor dit probleem op een gewone computer. Met dit algoritme is RSA dus veel efficiënter te kraken. Het record voor het algoritme van Shor ligt echter op 21, terwijl bij RSA in de praktijk natuurlijk veel grotere getallen worden gebruikt.

Voor AES is een efficiënter quantumalgoritme minder waarschijnlijk, omdat het maar een vrij simpel algoritme is. In de praktijk wordt de sleutel voor AES vaak versleuteld met RSA, dus als deze gekraakt is, kunnen de met AES versleutelde gegevens gewoon met de sleutel ontsleuteld worden.

Inhoud

3. Inleiding

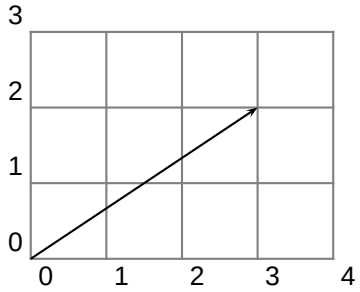
Quantumcomputers en cryptografie. Twee begrippen die misschien niet iedreen kent, maar zeker een ervan is ontzettend belangrijk in onze maatschappij: cryptografie. Cryptografie houd zich bezig met het versleutelen van gegevens, opdat onbevoegde personen geen toegang hebben tot deze gegevens. Men staat er niet vaak bij stil, maar een groot deel van het internetverkeer is versleuteld, ieder WhatsApp-berichtje is versleuteld, evenals de meeste gegevens op een moderne smartphone. De tekst wordt als het ware omgezet in een geheimtaal voordat deze verstuurd wordt en alleen de bedoelde ontvanger is in staat de originele tekst weer uit de geheimtaal tevoorschijn te halen, althans, dat is de bedoeling. Door de eeuwen heen zijn er verscheidene cryptografische systeem in gebruik genomen en vervolgens weer afgeschaft, omdat buitenstaanders toch in staat waren om de gegevens te lezen. Dit brengt ons bij het tweede begrip: de quantumcomputer.

Het begrip quantumcomputer is misschien wel beter bekend, het komt regelmatig langs in het nieuws, maar wat het precies inhoudt is voor de meeste mensen een raadsel. In de meeste artikelen over quantumcomputers wordt ook gesproken over de mogelijkheid dat deze apparaten in staat zouden zijn om de cryptografie van vandaag te kraken en met een beetje geluk wordt er ook nog gespeculeerd over de grote impact op ons dagelijks leven die dat zou hebben. Dit leidt tot de hoofdvraag van dit onderzoek: *kan een quantumcomputer de hedendaagse cryptografie veel efficiënter kraken dan een normale computer?* Een moderne computer kan namelijk ook de hedendaagse cryptografie kraken, maar dat kost erg veel tijd, denk aan duizenden jaren.

Om deze vraag te beantwoorden moeten we weten hoe de meest gebruikte cryptografie werkt en hoe een quantumcomputer werkt. Deze vragen worden respectievelijk beantwoord in hoofdstuk 5 en hoofdstuk 6. In 6.7 kijken we nog naar het algoritme van Shor, dat de sleutel zou kunnen zijn voor het kraken van de basis van de hedendaagse cryptografie. Uiteindelijk kunnen we in hoofdstuk 7 een conclusie trekken.

Bits	Hex	Bits	Hex	Bits	Hex	Bits	Hex
0000	0	0100	4	1000	8	1100	c
0001	1	0101	5	1001	9	1101	d
0010	2	0110	6	1010	a	1110	e
0011	3	0111	7	1011	b	1111	f

Tabel 1. Hexadecimale notatie



Figuur 1. De vector $\left[\begin{smallmatrix}3\\2\end{smallmatrix}\right]$

4. Voorbereidende wiskunde

Bij het rekenen met quantumcomputers en cryptografie komt wiskunde kijken die misschien niet algemeen bekend verondersteld kan worden. Om ervoor te zorgen dat er later geen al te grote onduidelijkheden ontstaan, worden hier een aantal wiskundige concepten kort uitelegd.

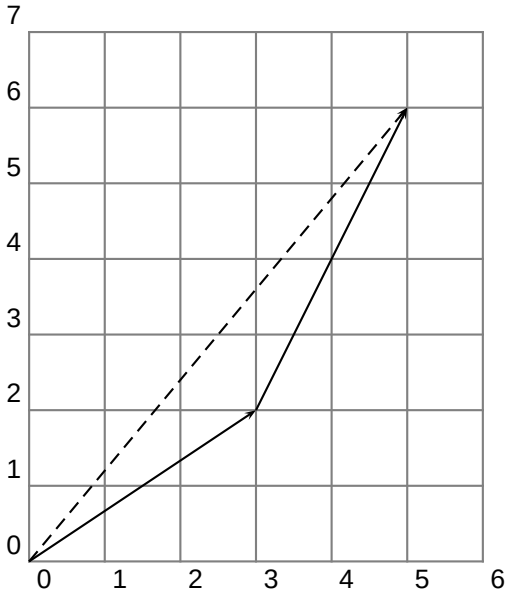
4.1. Hexadecimale notatie

Gewone computers rekenen met bits, die een waarde 0 of 1 kunnen hebben. Omdat er dus weinig informatie in één bit kan worden opgeslagen, worden ze vaak samengevoegd in groepjes van acht: een byte. Een byte is te noteren als een serie van acht binaire cijfers: 00101010. Dit neemt echter veel ruimte in, dus wordt vaak een verkorte versie van twee hexadecimale cijfers, waarbij elk cijfer vier bits representeerd, zoals weergegeven in tabel 1. 00101010 is dus ook te schrijven als 2A.

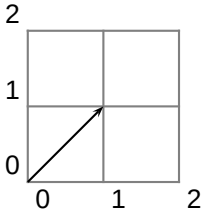
4.2. Vectoren

In een quantumcomputer is het vaak handig om de qubits als vectoren voor te stellen, maar om dat te doen is het handig om te weten wat vectoren inhouden. Een uitgebreide uitleg over dit onderwerp is bijvoorbeeld te vinden in [1], maar voor de volledigheid volgt hier een korte samenvatting.

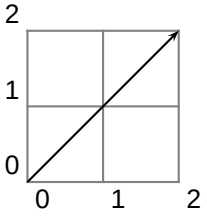
Een vector is, simpel gezegd, een lijnstuk met een bepaalde richting, dat in een tweedimensionaal vlak gedefinieerd wordt door het verschil in de x- en y-richting tussen het begin en het eind van het lijnstuk. De vector in figuur 1 heeft bijvoorbeeld een verschil van $3-0=3$ in de x-richting en een verschil van $2-0=2$ in de y-richting, dus geven we deze vector aan met de volgende notatie: $\begin{bmatrix}3\\2\end{bmatrix}$.



Figuur 2. $\left[\begin{smallmatrix}3\\2\end{smallmatrix}\right] + \left[\begin{smallmatrix}2\\4\end{smallmatrix}\right] = \left[\begin{smallmatrix}5\\6\end{smallmatrix}\right]$



(a) Origineel



(b) Na vermenigvuldiging met 2

Figuur 3. Voor en na een vermenigvuldiging met 2

4.2.1. Vectoren optellen

Een vector heeft geen vaste beginplaats, waardoor vectoren makkelijk opgeteld kunnen worden: door de ene vector met zijn beginpunt aan het eindpunt van de andere te 'plakken', is makkelijk te zien waar het resultaat uitkomt, zoals in figuur 2.

Voor het optellen van vectoren geldt dan de volgende regel:

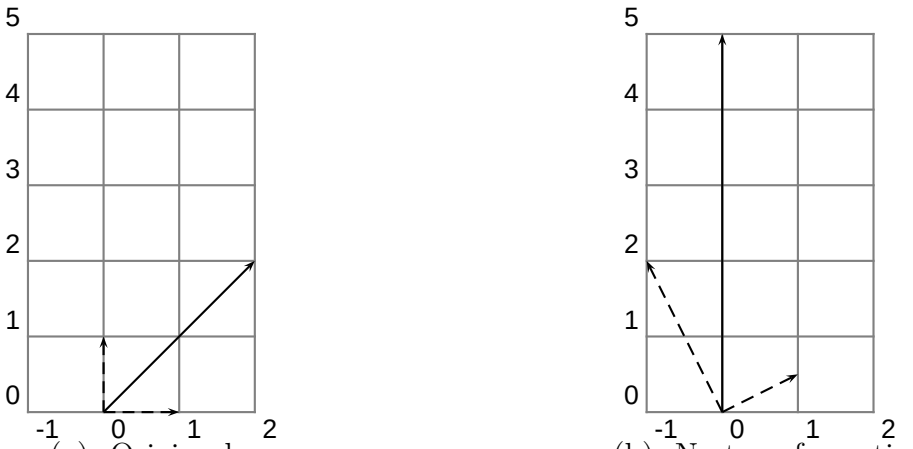
$$\begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a+c \\ b+d \end{bmatrix} \tag{1}$$

4.2.2. Vectoren vermenigvuldigen met een getal

Bij het vermenigvuldigen van een vector met een getal a blijft de richting van de vector gelijk en wordt de lengte a keer zo groot, wat hetzelfde inhoudt als het a keer zo groot worden van de afstand in de x-richting en het a keer zo groot worden in de y-richting, zoals te zien is in figuur 3. Voor het vermenigvuldigen van een vector geldt dus:

$$a \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax \\ ay \end{bmatrix} \tag{2}$$

4.2.3. Lineaire transformaties en matrices



(a) Origineel (b) Na transformatie

Figuur 4. Voor en na een transformatie

Een tweedimensionale lineaire transformatie is het beste voor te stellen als een vervorming van het gehele vlak waarbij het nulpunt niet verplaatst en de rasterlijnen parallel en gelijkmatig verdeeld blijven[1]. Zo'n transformatie is te beschrijven door bij te houden waar de basisvectoren $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ en $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ na de transformatie terechtkomen. In figuur 4a is een vector $\left[\begin{smallmatrix}2\\2\end{smallmatrix}\right]$ te zien, samen met de basisvectoren, die gestreept zijn weergegeven. In figuur 4b is een transformatie van deze vector weergegeven, die nu uitkomt als $\left[\begin{smallmatrix}0\\5\end{smallmatrix}\right]$. De transformaties van de basisvectoren zijn ook weergegeven: $\left[\begin{smallmatrix}-1\\2\end{smallmatrix}\right]$ en $\left[\begin{smallmatrix}1\\0{,}5\end{smallmatrix}\right]$. Deze vier getallen zijn genoeg om te beschrijven hoe deze transformatie eruit ziet voor elke vector. Elke vector is namelijk te schrijven als een combinatie van de twee basisvectoren:

$$\begin{bmatrix}x\\y\end{bmatrix} = x\begin{bmatrix}1\\0\end{bmatrix} + y\begin{bmatrix}0\\1\end{bmatrix} \tag{3}$$

Hierdoor kan ook voor elke vector berekend worden wat die na de transformatie is, als de nieuwe basisvectoren gegeven zijn:

$$\begin{bmatrix}x'\\y'\end{bmatrix} = x\begin{bmatrix}-1\\2\end{bmatrix} + y\begin{bmatrix}1\\0,5\end{bmatrix} \tag{4}$$

Om een transformatie te beschrijven, worden deze nieuwe basisvectoren samen in een matrix geplaatst:

$$\begin{bmatrix}-1 & 1\\2 & 0,5\end{bmatrix}$$

Om een transformatie, gegeven als een matrix, op een vector uit te voeren, geldt dan de volgende regel:

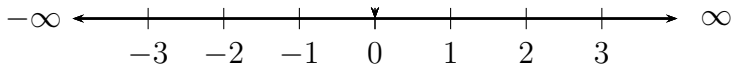
$$\begin{bmatrix}a & b\\c & d\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} = x\begin{bmatrix}a\\c\end{bmatrix} + y\begin{bmatrix}b\\d\end{bmatrix} = \begin{bmatrix}xa+yb\\xc+yd\end{bmatrix} \tag{5}$$

4.2.4. Inproduct

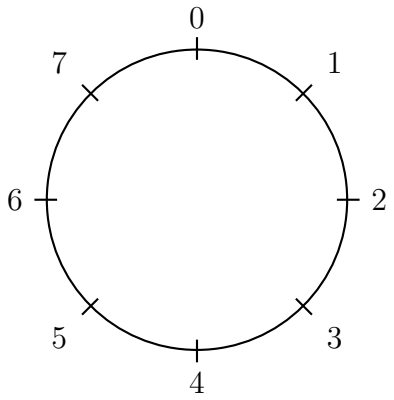
Het inproduct van twee vectoren is als volgt te berkenen:

$$\begin{bmatrix}a\\b\end{bmatrix} \cdot \begin{bmatrix}c\\d\end{bmatrix} = \begin{bmatrix}a & b\end{bmatrix}\begin{bmatrix}c\\d\end{bmatrix} = ac + bd \tag{6}$$

[1] Zie voor enkele duidelijke animaties hiervan [Sanderson [1]; vierde video].



(a) Een normale getallenlijn



(b) Getallenlijn mod8

Figuur 5. Een voorstelling van modulo

Merk op dat de eerste vector in de tussenstap wordt geschreven als een 2×1 matrix, waarna deze transformatie wordt toegepast. Zie [Sanderson [1]; tiende video] voor meer uitleg over waarom dit zo werkt.

4.3. Modulo

De beste voorstelling van modulo rekenen is waarschijnlijk om de normale getallenlijn (figuur 5a) voor te stellen als een cirkel, zoals in figuur 5b. Bij het tellen op deze 'getallencirkel' kom je na 7 weer uit bij 0, waardoor bijvoorbeeld geldt $5 + 4 \bmod 8 = 1$. Op de modulo 8 cirkel beginnend bij 5 is namelijk te zien dat vier stappen verder uitkomt bij 1. Dit betekent ook dat a mod b te berekenen is als de rest die overblijft bij het delen van a door b.

4.4. Exclusieve disjunctie

De exclusieve disjunctie wordt vaak afgekort als XOR, naar de Engelse term *exclusive or* en in vergelijkingen weergegeven met $\oplus$. In de logica geldt dat $a \oplus b$ waar is, als a of b waar is, maar niet allebei, zoals te zien in tabel 2.

Voor bits geldt 0 als onwaar en 1 als waar, dus $1 \oplus 0 = 1$, $0 \oplus 0 = 0$ en $1 \oplus 1 = 0$. Daarom geldt bij het rekenen met bits $a \oplus b = a + b \bmod 2$.

4.5. Complexe getallen

Normaal gesproken wordt er alleen gerekend met reëele getallen: al deze getallen kunnen we voorstellen op een lijn, en deze getallen worden verzameld in $\mathbb{R}$. In de verzameling

a	b	$a \oplus b$
onwaar	onwaar	onwaar
waar	onwaar	waar
onwaar	waar	waar
waar	waar	onwaar

Tabel 2. Waarheidstabel voor de exclusieve disjunctie

van reëele getallen is $\sqrt{-1}$, of de wortel van elk ander negatief getal, niet mogelijk, omdat er op de getallenlijn geen getal is dat met zichzelf vermenigvuldigd een negatief getal oplevert. Bij complexe getallen is een negatieve wortel wel mogelijk, waarvoor het getal i gebruikt wordt:

$$i = \sqrt{-1} \tag{7}$$

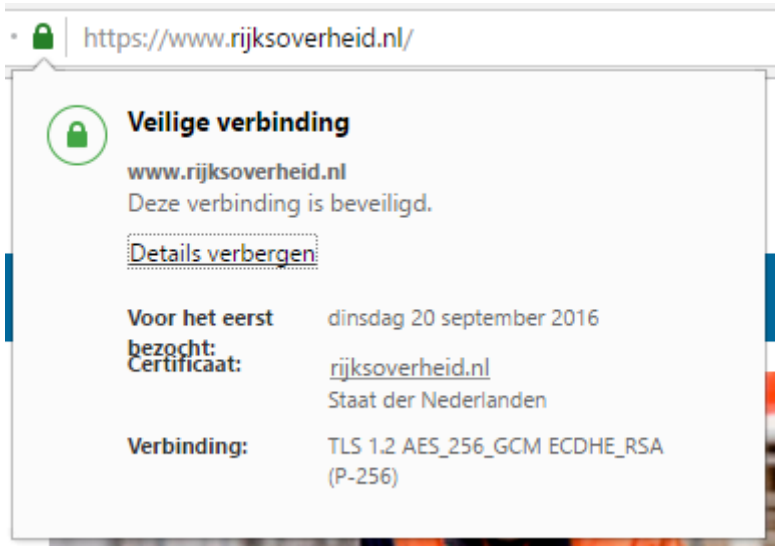
Alle complexe getallen ($\mathbb{C}$) zijn voor te stellen in de volgende vorm:

$$z = a + bi \in \mathbb{C} \qquad \text{met } a, b \in \mathbb{R} \tag{8}$$

Van het complexe getal z is de complex geconjugeerde z^* gedefinieerd:

$$z^* = a - bi \tag{9}$$

Dit is slechts een zeer beknopte weergave van complexe getallen, een uitgebreide uitleg is bijvoorbeeld te vinden in [2].



Figuur 6. Een versleutelde verbinding met www.rijksoverheid.nl

	Sleutel lengte Nk	**Invoer lengte** Nb	**Aantal rondes** Nr
AES-128	128	128	10
AES-192	192	128	12
AES-256	256	128	14

Tabel 3. Verschillende sleutellengtes voor AES

5. Hoe werkt de meest gebruikte cryptografie?

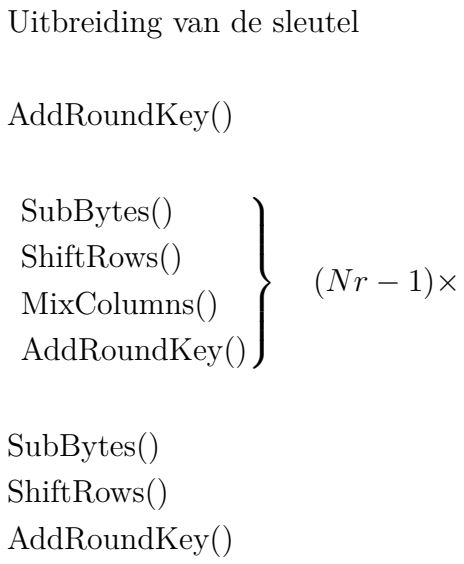
Cryptografie komt voor in vele verschillende soorten en maten, maar waarschijnlijk de bekendste vorm van encryptie wordt door bijna elke internetgebruiker dagelijks gebruikt en is te herkennen aan het groene slotje in de browser: HTTPS.

HTTPS is op zichzelf eigenlijk geen cryptografisch systeem, maar een protocol voor het gebruik van encryptie bij internetverbindingen. In de TLS specificatie [3], het protocol waar HTTPS gebruik van maakt, is dan wel vrijheid gegeven om uit verschillende cryptografische systemen te kunnen kiezen, maar standaard wordt er gebruik gemaakt van RSA en AES en in de praktijk wordt van de verdere vrijheid slechts weinig gebruik gemaakt. In figuur 6 is te zien dat bijvoorbeeld ook www.rijksoverheid.nl gebruik maakt van deze systemen, die hierna besproken worden.

5.1. Hoe werkt AES?

De Advanced Encryption Standard [4] is één van de meest gebruikte blokversleutelingen en is gebaseerd op het Rijndael algoritme [5], dat ontworpen is door de Belgische Joan Daemen en Vincent Rijmen. In tegenstelling tot Rijndael is AES alleen gespecificeerd voor een invoer van 128 bits, er kunnen dus alleen blokken van 128 enen en nullen mee versleuteld worden. Voor de sleutel zijn er drie lengtes mogelijk, namelijk 128, 192 of 256 bits. Bij een langere sleutel wordt de versleuteling meerdere malen herhaald, zoals te zien in tabel 3.

Om handig aan te duiden met welke sleutellengte AES gebruikt wordt, spreekt men



Figuur 7. Schematische weergave van het AES algoritme

meestal van AES-128, AES-192 of AES-256 bij respectievelijke sleutellengtes van 128, 192 of 256 bits.

De eerste stap van het AES algoritme is de uitbreiding van de sleutel. Vervolgens wordt er met `AddRoundKey()` een exclusieve disjunctie uitgevoerd met de eerste 128 bits van de uitgebreide sleutel en de invoer, waarna er in $Nr - 1$ rondes, achtereenvolgens de functies `SubBytes()`, `ShiftRows()`, `MixColumns()` en weer `AddRoundKey()`, deze laatste steeds met de volgende 128 bits van de uitgebreide sleutel. Als laatste worden nog een keer `SubBytes()`, `ShiftRows()` en `AddRoundKey()` uitgevoerd. Dit is schematisch weergegeven in figuur 7, de verschillende onderdelen worden hierna besproken.

5.1.1. Uitbreiding van de sleutel

De sleuteluitbreiding genereert een sleutel van $Nb(Nr+1)$ bits: voor elke ronde zijn er 128 bits (Nb) nodig om de invoer mee te versleutelen, naast een initiële 128 bits, die nog voor de eerste ronde gebruikt worden bij de initiële versleuteling. De uitbreiding van een sleutel met 128 bits levert dus $128 \times (10 + 1) = 1408$ bits en gaat als volgt: eerst wordt de sleutel opgedeeld in 4 blokken van 4 bytes, die worden opgeslagen in rij w. In het volgende voorbeeld wordt gebruik gemaakt van de sleutel `2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c`.

w_0 `2b7e1516`
w_1 `28aed2a6`
w_2 `abf71588`
w_3 `09cf4f3c`

In vergelijking (10) wordt beschreven hoe de rest van de rij eruit gevormd wordt.

$$w_i = w_{i-1} \oplus w_{i-4} \qquad \text{als } i > 3 \wedge i \bmod 4 \neq 0 \tag{10}$$

Als i wel deelbaar is door 4, wordt er nog een serie andere wijzigingen doorgevoerd:

1. `RotWord()`: deze functie krijgt als invoer een rij van vier bytes $[a_0, a_1, a_2, a_3]$ en geeft als uitvoer $[a_1, a_2, a_3, a_0]$. De eerste byte wordt dus weggehaald en achteraan teruggeplakt.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabel 4. Vervangingswaarden voor bytes in `SubBytes()`

i	w_{i-1}	**(1)**	**(2)**	**(3)**	w_{i-4}	**(4)**
4	09cf4f3c	cf4f3c09	8a84eb01	8b84eb01	2b7e1516	a0fafe17
5	a0fafe17				28aed2a6	88542cb1
6	88542cb1				abf71588	23a33939
7	23a33939				09cf4f3c	2a6c7605
8	2a6c7605	6c76052a	50386be5	52386be5	a0fafe17	f2c295f2
…						

Tabel 5. Voorbeeld van het sleuteluitbreidingsalgoritme voor AES-128

2. `SubWord()`: alle bytes worden vervangen door de bijbehorende waarde in de S-box, zie tabel 4. De byte {10} bijvoorbeeld wordt zo vervangen door {ca}.
3. temp$\oplus[2^{\frac{i}{4}-1}, 00, 00, 00]$, waarin temp de waarde is die bij de vorige stap is verkregen. Bij bijvoorbeeld $i = 8$: $[50, 38, 6b, e5] \oplus [02, 00, 00, 00] = [52, 38, 6b, e5]$.
4. $w_i = \text{temp} \oplus w_{i-4}$, waarin temp wederom de waarde verkregen uit de vorige stap is.

In het kort wordt er dus steeds de exclusieve disjunctie genomen van de voorgaande waarde en de waarde vier plaatsen eerder, elke vierde keer wordt de voorgaande waarde eerst nog een beetje veranderd, om een patroon in de uitgebreide sleutel te voorkomen. Het resultaat van deze stappen is te zien in tabel 5 voor $4 \leq i \leq 8$. Een volledige uitwerking is te vinden in Appendix A van [4].

5.1.2. Versleutelen

Bij het versleutelen wordt de invoer opgeslagen in de variabele s, ook wel de *state* genoemd, die we kunnen voorstellen als een raster van vier bij vier met in elk hokje een byte, zoals te zien in figuur 8.

invoer bytes					**state**			
in_0	in_4	in_8	in_{12}		$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
in_1	in_5	in_9	in_{13}	$\rightarrow$	$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
in_2	in_6	in_{10}	in_{14}		$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
in_3	in_7	in_{11}	in_{15}		$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

Figuur 8. Invoer en *state*

s					s'			
$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$		$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$		$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$		$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$		$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

Figuur 9. `ShiftRows()`

`AddRoundKey()`

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{4r+c}] \qquad \text{voor } 0 \leq c < 4 \tag{11}$$

In vergelijking (11) wordt weergegeven hoe de `AddRoundKey()` functie werkt. In deze vergelijking is w de rij met de uitgebreide sleutel, zoals beschreven in 5.1.1 en r de ronde waarin de functie wordt aangeroepen. Bij de eerste aanroep, die immers buiten de herhaling valt, geldt $r = 0$; bij de laatste, ook buiten de herhaling, geldt $r = Nr$. `AddRoundKey()` neemt dus steeds de exclusieve disjunctie van de huidige staat en de volgende 128 bits in de uitgebreide sleutel.

`SubBytes()` Deze `SubBytes()` functie is vergelijkbaar met de functie `SubWord()` beschreven in 5.1.1 en vervangt elke byte door de bijbhorende byte in de S-box, zie tabel 4.

`ShiftRows()` De `ShiftRows()` functie zorgt voor een verschuiving in de rijen van s. De eerste rij blijft hetzelfde, bij de tweede rij wordt de eerste byte naar achteren verplaatst, bij de derde rij worden de eerste twee bytes naar achteren verplaatst en bij de vierde rij worden de eerste drie bytes naar achteren verplaatst. Dit is weergegeven in figuur 9.

`MixColumns()` De `MixColumns()` functie is een matrix transformatie die werkt op de kolommen van s, zoals weergegeven in vergelijking (12).

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \qquad \text{voor } 0 \leq c < 4 \tag{12}$$

5.1.3. Ontsleutelen

Het ontsleutelen van een met AES versleutelde tekst is gemakkelijk te doen door het algoritme simpelweg in omgekeerde volgorde uit te voeren, waarbij gebruik gemaakt wordt

tekst	00101010
sleutel	11011000
versleuteld	11110010
versleuteld	00101010

Tabel 6. Opnieuw versleutelen levert de originele tekst op

van een prettige eigenschap van de exclusieve disjunctie, die wordt weergegeven in vergelijking (13).

$$(a \oplus b) \oplus b = a \tag{13}$$

Hierdoor kunnen we de cijfertekst als het ware opnieuw versleutelen, om de originele tekst terug te krijgen. Om dit een beetje te verduidelijken, is in tabel 6 een voorbeeld gegeven.

Omdat voor het versleutelen en ontsleutelen van de data dezelfde sleutel gebruikt wordt, is AES een symmetrisch algoritme.

5.2. Hoe werkt RSA?

RSA is, in tegenstelling tot AES, een asymmetrisch algoritme: voor het versleutelen wordt namelijk een andere sleutel gebruikt als voor het ontsleutelen. Het RSA algoritme, zo genoemd naar de drie bedenkers **R**ivest, **S**hamir en **A**dleman, is gepubliceerd in 1977 [6] en wordt nog altijd gebruikt. Vergeleken met AES is RSA een langzaam algoritme, maar door de asymmetrische sleutels is het veel handiger in gebruik: er hoeft namelijk geen geheime sleutel tussen twee partijen gecommuniceerd te worden, wat het zwakke punt is bij AES. Om een beveiligde verbinding tot stand te brengen is een bij beide partijen bekende sleutel nodig en om de sleutel te communiceren is een beveiligde verbinding nodig. In de praktijk worden de twee daarom vaak gecombineerd: de sleutel voor AES wordt met RSA versleuteld en naar de andere partij verstuurd; de data zelf wordt met AES versleuteld.

5.2.1. Asymmetrische cryptografie

Het idee voor asymmetrische cryptografie werd in 1976 geopperd door Diffie en Hellman in [7], maar zij gaven geen details over hoe dit gerealiseerd kon worden. Wel gaven zij de vier criteria waar asymmetrische cryptografie aan moet voldoen:

1. Het onstsleutelen van een versleuteld getal, levert datzelfde getal: $D(E(M)) = M$, waarin M de tekst, E de encryptiefunctie en D de decryptiefunctie is.
2. Zowel E als D zijn makkelijk te berekenen.
3. Bij het publiceren van E wordt er geen makkelijke manier gegeven om D te berekenen.
4. Als M eerst wordt ontcijferd en vervolgens vercijferd, is het resultaat M: $E(D(M)) = M$.

Er moeten dus verschillende sleutels zijn voor encryptie en decryptie, die gemakkelijk te maken, maar, indien een van twee bekend is, moeilijk te herleiden zijn.

5.2.2. Encryptie en decryptie functies

De encryptie functie voor RSA is gedefinieerd in vergelijking (14), de decryptie functie in vergelijking (15).

$$C = E(M) = M^e \bmod n \tag{14}$$

$$M = D(C) = C^d \bmod n \tag{15}$$

In deze vergelijkingen is M het klare getal en C het versleutelde getal. Als encryptiesleutel kan dan (e, n) worden gebruikt, als decryptiesleutel (d, n). Om deze sleutels te berekenen worden twee priemgetallen p en q gekozen. De waarde van n in beide sleutels wordt hiermee gedefinieerd in vergelijking (16).

$$n = p \cdot q \tag{16}$$

De waarde van d wordt zo gekozen dat d geen gemeenschappelijke delers met $(p-1)(q-1)$ groter dan 1 heeft, oftewel $ggd(d, (p-1)(q-1)) = 1$. Met d wordt dan e berekend, zoals in vergelijking (17).

$$e = d^{-1} \bmod (p-1)(q-1) \tag{17}$$

Deze functies voldoen aan de door Diffie en Hellman gestelde eisen voor assymetrische cryptografie. Om bij een bekende encryptiefunctie de bijbehorende decryptiefunctie te vinden is nodig om de priemfactoren p en q te berekenen, wat zelfs met de beste algoritmes bij grote getallen met een gewone computer al snel miljarden jaren kan duren. Op deze moeilijkheid is de veiligheid van RSA gebaseerd.

5.2.3. Voorbeeld

Als voorbeeld worden $p = 47$, $q = 59$ en $d = 157$ gekozen, vergelijking (16) geeft dan $n = 47 \cdot 59 = 2773$, en $(p-1)(q-1) = 46 \cdot 58 = 2668$. De waarde van e kan worden berekend met het algoritme van Euclides:

	2668	**157**
2668	1	0
157	0	1
156	1	-16
1	-1	17

Hieruit volgt dat $e = 157^{-1} \bmod 2668 = 17$. De encryptiesleutel is dus $(17, 2773)$, de decryptiesleutel $(157, 2773)$. Het invullen van deze waardes in vergelijkingen (14) en (15) geeft dan de volgende encryptie- en decryptiefuncties voor dit voorbeeld:

$$E(M) = M^{17} \mod 2773$$
$$D(C) = C^{157} \mod 2773$$

Het versleutelen van het getal $M = 15$ levert dan $E(15) = 15^{17} \bmod 2773 = 1392$ en ontsleutelen van deze uitkomst levert $D(1392) = 1392^{157} \bmod 2773 = 15$. In de praktijk worden natuurlijk vele malen grotere getallen gebruikt; de priemfactoren van 2773 zijn, met een beetje doorzettingsvermogen, ook met de hand nog wel te berekenen en dus voor een computer een fluitje van een cent.

6. Hoe werkt een quantumcomputer?

Op een zoektocht naar een werkende quantumcomputer zijn er twee partijen die snel boven komen drijven: D-Wave Systems, een in 1999 opgericht Canadees bedrijf, specifiek gericht op quantumcomputers; en IBM, de in 1911 opgerichte computergigant, met name bekend door een van de eerste PC's: de IBM PC. De quantumcomputers van beide bedrijven zijn zeer verschillend: de systemen van D-Wave zijn commercieel verkrijgbaar en worden onder andere gebruikt door Lockheed Martin; IBM's quantumcomputer is slechts een onderzoeksproject. Een ander groot verschil is het aantal qubits dat gebruikt wordt in de computers: D-Wave presenteerde recent een preview van een systeem met 2000 qubits [8], IBM's systeem heeft er 5. Het lijkt er dus op dat D-Wave de standaard moet worden voor een onderzoek naar quantumcomputers; hun systeem werkt immers het best.

Nader onderzoek wijst echter uit dat de claim van D-Wave van de eerste commerciele quantumcomputer niet geheel geaccepteerd wordt: zo laat David DiVincenzo, auteur van een van de meest invloedrijke artikelen over quantumcomputers [9], in een interview [10] weten dat de D-Wave computers weliswaar enkele kenmerkende eigenschappen van een quantumcomputer hebben, maar dat de tijd waarin de qubits zich in superpositie bevinden, slechts enkele nanoseconden, wat hem betreft te kort is om deze systemen ook daadwerkelijk als quantumcomputers te bestempelen. Daarnaast is de hardware van de D-Wave computers speciaal gebouwd om één specifiek probleem op te lossen, dus is het niet mogelijk deze computer te programmeren. De winnaar is dus IBM.

Bijkomend voordeel is dat IBM veel opener is over hun quantumcomputer en deze zelfs op het internet heeft aangesloten, zodat iedereen via de IBM Quantum Experience (https://quantumexperience.ng.bluemix.net) programma's op deze 5 qubit quantumcomputer kan uitvoeren.

6.1. IBM Quantum Experience

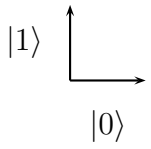
In de IBM Quantum Experience is het mogelijk om met behulp van de 'Composer' een programma te schrijven dat kan worden uitgevoerd op de IBM 5Q. De Composer is een grafisch programma, maar achter de schermen wordt zogenaamde QASM gegenereerd, de taal die IBM ontworpen heeft om hun quantumcomputer te kunnen programmeren. Bij de uitleg over de werking van quantumcomputers zullen enkele voorbeelden van programma's in de Quantum Experience gebruikt worden, deze zijn te herkennen aan de term QASM. Hier zullen alleen de grafische weergave van het programma en de resultaten van de laatste uitvoering getoond worden, een link naar het programma in de Quantum Experience is te vinden in appendix A.

6.2. Wat is een quantumcomputer niet?

Voor we verdergaan met een echte uitleg van een quantumcomputer, is het wellicht handig om wat mythes de wereld uit te helpen.

Een veelgehoorde uitleg van een quantumcomputer gaat als volgt: een quantumcomputer gebruikt in plaats van gewone bits, qubits, die op waardes 1, 0, of allebei tegelijk kunnen staan. Hierdoor kunnen heel veel berekeningen parallel worden uitgevoerd, wat het bij bijvoorbeeld het kraken van een versleutelde code veel makkelijker maakt om simpelweg alle opties uit te proberen.

Helaas, zo werkt dat niet. Een qubit kan inderdaad in een zogenaamde superpositie terechtkomen, maar dat deze qubit tegelijkertijd voor meerdere berekeningen gebruikt



Figuur 10. Qubits $|0\rangle$ en $|1\rangle$ als vectoren

kan worden is de verkeerde conclusie.

Een antwoord op de vraag "Waarom is een quantumcomputer sneller dan een gewone computer?" wordt wel beantwoord met een uitleg dat het aantal verschillende mogelijke toestanden van het gehele systeem exponentieel toeneemt bij het toenemen van het aantal qubits. Het aantal mogelijkheden wordt hierdoor al snel heel erg groot, zoals bij de beroemde legende van het schaakbord vol graankorrels.

Op zich is dit verhaal correct, een antwoord op de vraag is het echter niet. Bij een gewone computer is dit namelijk niet anders, ook hier neemt het aantal toestanden exponentieel toe, en de gemiddelde computer heeft vele malen meer bits tot zijn beschikking dan zelfs het experimentele D-Wave systeem. Voor een uitleg waarom een quantumcomputer soms wel sneller is, zie paragraaf 6.6.

Nog een misverstand is dat op een quantumcomputer alles sneller gaat, maar dat zal meestal niet het geval zijn. Een quantumcomputer is alleen sneller bij algoritmes die gebruik maken van de quantummogelijkheden van de qubits. Een ouderwets algoritme dat ontworpen is voor normale computers zal waarschijnlijk alleen maar langzamer werken, omdat er rekening gehouden moet worden met de willekeurigheid die nu eenmaal bij quantummechanica hoort.

Een quantumcomputer is dus niet een magische machine die alles veel sneller kan; nu eenmaal niet alle problemen worden makkelijker door er quantummechanica aan toe te voegen.

6.3. Wat is een qubit?

Een normale computer rekent met bits, die een waarde 0 of 1 hebben. Een quantumcomputer werkt op een vergelijkbare manier, maar dan met quantum bits (qubits), die een waarde 0, 1, of een superpositie tussen deze twee hebben. Om dit voor te kunnen stellen, worden qubits aangegeven als een vector: de qubit met waarde 0 krijgt de vector $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ en die met waarde 1 $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$. Deze qubits krijgen ook hun eigen naam in de speciale bra-ket notatie: 0 wordt $|0\rangle$, 1 wordt $|1\rangle$, zie ook figuur 10.

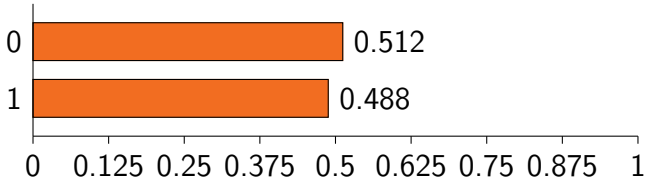
Behalve deze twee basiswaarden kan een qubit zich ook in superpositie bevinden: ergens tussen beide basiswaarden, met als voorwaarde dat de qubit naar een van de basiswaarden moet kunnen terugvallen, waardoor de lengte altijd gelijk moet zijn, namelijk 1.

In 4.2.3 is beschreven hoe elke vector geschreven kan worden als een combinatie van twee basisvectoren. Zo is ook elke qubit te schrijven als een combinatie van de twee basiswaarden, waarmee een qubit als volgt wordt gedefinieerd:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \qquad \text{als } \alpha, \beta \in \mathbb{C} \text{ en } |\alpha|^2 + |\beta|^2 = 1 \tag{18}$$

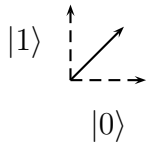
Het probleem met superpositie is dat het niet gemeten kan worden: zodra de qubit geobserveerd wordt, zal hij terugvallen naar een van de basiswaarden. Door deze routine

(a) QASM



(b) Resultaat

QASM 1. Gelijkmatige superpositie (1024 Shots)



(a)

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$$

(b)

Figuur 11. Gelijkmatige superpositie als vector

vaak te herhalen, kan de kans dat de qubit naar $|0\rangle$ of $|1\rangle$ terugvalt worden berekent, waarmee vervolgens de superpositie bepaald kan worden.

In QASM 1 is een qubit met een Hadamard poort in een gelijkmatige superpositie gebracht, die bij het meten terugvalt naar een van de basiswaarden. Aan het resultaat is te zien dat de qubit in ongeveer de helft van de tijd naar 0 terugvalt en de andere helft naar 1, vandaar de naam gelijkmatige superpositie. De vectorvoorstelling van deze superpositie is te zien in figuur 11.

Om te controleren dat dit inderdaad een geldige qubit is, kan deze vector geschreven worden in de vorm $\alpha|0\rangle + \beta|1\rangle$, waarbij moet gelden dat $|\alpha|^2 + |\beta|^2 = 1$ (zie vergelijking (18)):

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \Rightarrow \left(\frac{1}{\sqrt{2}}\right)^2 + \left(\frac{1}{\sqrt{2}}\right)^2 = 1 \tag{19}$$

6.3.1. Bra-ket notatie

Eerder is vermeld dat de basiswaarden een speciale bra-ket notatie kregen: $|0\rangle$ en $|1\rangle$. Dit zijn de ket-versies van de vectoren. Naast de ket-versie is er, zoals de naam van de notatie doet vermoeden, ook een bra-versie, die als volgt gedefinieerd is:

$$\langle\cdot| = (|\cdot\rangle^*)^T \tag{20}$$

Dit betekent dat de bra de geconjugeerde transpositie van de ket is. Als voorbeeld wordt de vector $|\psi\rangle$ gebruikt:

$$|\psi\rangle = \begin{bmatrix} a+bi \\ c+di \end{bmatrix} \tag{21}$$

$$\langle\psi| = (|\psi\rangle^*)^T = \begin{bmatrix} (a+bi)^* \\ (c+di)^* \end{bmatrix}^T = \begin{bmatrix} (a-bi) \\ (c-di) \end{bmatrix}^T = \begin{bmatrix} a-bi & c-di \end{bmatrix} \tag{22}$$

Het sterretje naast de ket betekent dus dat van alle getallen in de vector de geconjugeerde wordt genomen; de T naast de vector dat deze vector geschreven wordt als een 2×1 matrix.

De bra-ket notatie is vooral handig bij het berekenen van het inproduct:

$$|a\rangle \cdot |b\rangle = \langle a||b\rangle = \langle a|b\rangle \tag{23}$$

Of het berekenen van de lengte van een vector:

$$\text{lengte}(|a\rangle) \ = \sqrt{\langle a|a\rangle} \tag{24}$$

6.3.2. Kans op een uitkomst

Het is voor elke superpositie mogelijk om te berekenen wat de kans is dat de qubit bij het uitlezen terugvalt naar een bepaalde basiswaarde door te kijken "hoeveel van die basiswaarde in de superpositie zit." Dit kan berekend worden met het inproduct van de qubit en de basiswaarde:

$$p_0 = |\langle\psi|0\rangle|^2 \tag{25}$$

Door de notatie voor een qubit uit vergelijking (18) te gebruiken, kan deze kans simpeler uitgedrukt worden:

$$p_0 = |(\alpha\langle 0| + \beta\langle 1|)|0\rangle|^2 = |\alpha\langle 0|0\rangle + \beta\langle 1|0\rangle|^2 = |\alpha \times 1 + \beta \times 0|^2 = |\alpha|^2 \tag{26}$$

Op dezelfde manier kan de kans dat de qubit naar 1 terugvalt worden berekend:

$$p_1 = |\langle\psi|1\rangle|^2 = |\beta|^2 \tag{27}$$

Hierdoor wordt ook duidelijk waarom in vergelijking (18) de voorwaarde $|\alpha|^2 + |\beta|^2 = 1$ gesteld wordt: de totale kans dat de qubit naar een van de twee basiswaarden terugvalt moet natuurlijk 1 zijn.

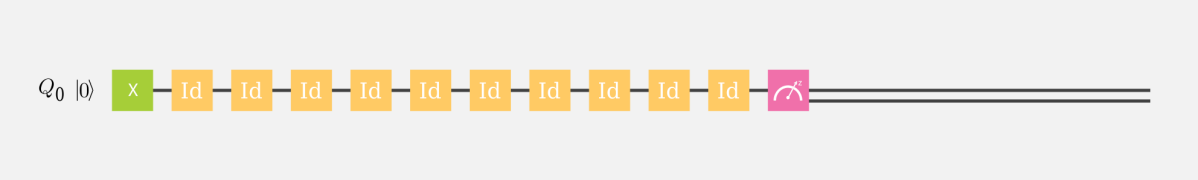
We kunnen nu ook de kansen uit QASM 1 narekenen. Hiervoor schrijven we de superpositie eerst als een combinatie van de basiswaarden:

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \tag{28}$$

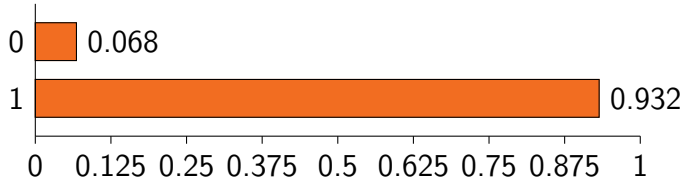
Zowel de kans op 0, als op 1 is dan $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$.

6.3.3. Neiging naar 0?

Gebaseerd op de resultaten van QASM 1 en de uitkomst van de berekening ontstaat het vermoeden dat de qubit een lichte neiging naar basiswaarde 0 heeft. Dit vermoeden wordt bevestigd in QASM 2: de qubit wordt met een Pauli-X poort op 1 gezet, waarna er tien keer de Id-poort wordt toegepast, alvorens de qubit wordt uitgelezen. De Id-poort



(a) QASM



(b) Resultaat

QASM 2. De qubit heeft een neiging naar 0 (1024 Shots)

verandert niets aan de qubit, maar zorgt er enkel voor dat er dezelfde tijd wordt gewacht als nodig zou zijn voor elke andere poort.

In de resultaten van QASM 2 is te zien dat de qubit inderdaad na verloop van tijd ook vanuit waarde 1 weer terugvalt naar 0. Deze afwijking is te verklaren door te kijken naar de fysieke implementatie van de qubit: in de IBM 5Q zijn de qubits gemaakt van supergeleiders bij een temperatuur van ongeveer 15 milliKelvin, om warmte of achtergrondgeluid geen invloed te laten hebben op de qubits, opdat de qubits zo lang mogelijk in de aangeslagen toestand $|1\rangle$ kunnen blijven. Deze afsluiting is echter niet perfect, waardoor de qubits naar verloop van tijd weer terugvallen naar de grondtoestand $|0\rangle$, dit principe heet decoherentie.

6.4. Qubits combineren

Een staat van 2 qubits is te schrijven als een vierdimensionale vector met behulp van het tensorproduct.

$$\begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a \begin{bmatrix} c \\ d \end{bmatrix} \\ b \begin{bmatrix} c \\ d \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix} \tag{29}$$

De staat van twee qubits is ook te schrijven als ket: een combinatie van $|0\rangle$ en $|1\rangle$ wordt $|01\rangle$, ongeveer op dezelfde manier als dat bij gewone bits gedaan wordt. De vector van deze staat is als volgt te berekenen:

$$|01\rangle = |0\rangle \oplus |1\rangle = \begin{bmatrix} 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{30}$$

6.5. Quantumpoorten

De quantumpoorten in een quantumcomputer zijn vergelijkbaar met de logische poorten in een normale computer. Onder deze logische poorten vallen bijvoorbeeld de AND, OR,

XOR etc. De meeste van deze logische gebruiken twee bits als invoer en geven als resultaat één bit. In de negen poorten in de IBM 5Q is er echter maar een poort die niet op een enkele qubit werkt, namelijk de CNOT.

6.5.1. Puali poorten

De handigste manier om quantumpoorten te beschrijven is met een matrix; een poort doet namelijk niets anders dan een qubit, die wordt voorgesteld als een vector, transformeren. De Pauli-X poort is al gebruikt in QASM 2, waarin te zien is dat deze poort $|0\rangle$ omzet naar $|1\rangle$. In matrixvorm ziet de de Pauli-X poort er als volgt uit:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{31}$$

Het uitvoeren van deze transformatie op $|0\rangle$ resulteert inderdaad in $|1\rangle$:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \tag{32}$$

De Pauli-X poort heeft dus een vergelijkbare werking als de klassieke NOT poort en wordt daarom ook wel een *bit-flip* genoemd. Naast Pauli-X zijn er nog twee andere Pauli-poorten, namelijk Pauli-Y en Pauli-Z:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{33}$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{34}$$

6.5.2. Poorten voor superpositie

In QASM 1 werd een Hadamard poort gebruikt om een gelijkmatige superpositie tussen $|0\rangle$ en $|1\rangle$ te creëren. De matrixtransformatie van de Hadamard poort ziet er als volgt uit:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{35}$$

In plaats van de standaardbasis $\{|0\rangle, |1\rangle\}$ wordt soms ook de Hadamardbasis $\{|+\rangle, |-\rangle\}$ gebruikt:

$$|+\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} |0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{36}$$

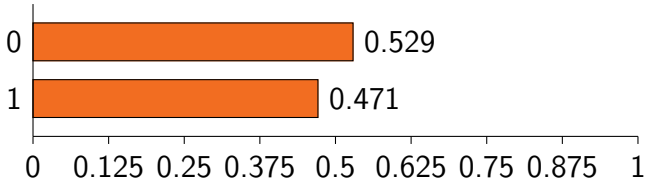
$$|-\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} |1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{37}$$

Twee poorten die veel gebruikt worden om superposities aan te passen, zijn de S en $S^\dagger$ poorten:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \tag{38}$$

$$S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \tag{39}$$

(a) QASM



(b) Resultaat

QASM 3. Demonstratie van de CNOT poort (1024 Shots)

6.5.3. Poort voor meerdere qubits

Er is een poort in de IBM 5Q die werkt op twee qubits: de CNOT (Controlled NOT). Deze poort voert een Pauli-X poort uit op de bestuurde qubit, als de controlerende qubit in de staat $|1\rangle$ is.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{40}$$

De CNOT poort werkt op een gecombineerde staat van twee qubits $|ab\rangle$, waarbij a de controlerende functie heeft. In QASM 3 is de CNOT poort in actie te zien. De controlerende qubit Q_1 wordt in een gelijkmatige superpositie gebracht, deze waarde wordt uitgelezen en op basis daarvan wordt op Q_2 wel of niet de Pauli-X poort uitgevoerd. Het resultaat is de meting van Q_2.

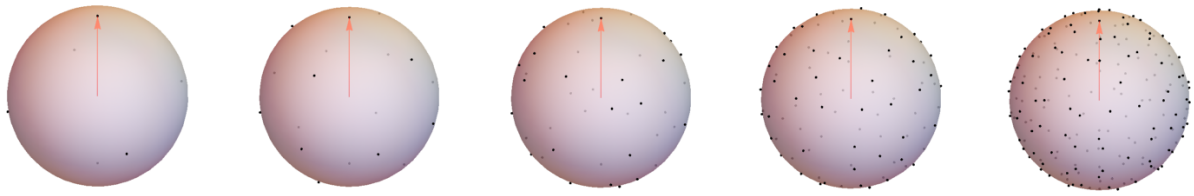
Omdat de CNOT poort de enige poort is die met twee qubits iets te maken heeft is deze poort vereist voor het maken van een verstrengeling tussen twee qubits. In een verstrengelde staat kan de staat van de ene qubit de staat andere beïnvloeden, ook al lijkt er geen verbinding tussen de twee te zijn. De twee qubits in QASM 3 zijn in een speciale verstrengelde staat gebracht, namelijk de Bell-staat:

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{41}$$

Deze relatief simpele verstrengelde staat wordt vaak gebruikt om te bewijzen dat twee qubits verstrengeld kunnen zijn.

6.5.4. Functioneel compleet

Alle tot nu toe genoemde quantumpoorten zijn onderdeel van de zogenaamde Clifford-groep. Deze poorten kunnen efficient gesimuleerd worden op een normale computer, wat



Figuur 12. Te bereiken qubit waardes bij T-dieptes 0, 1, 2, 3, 4

betekent dat deze poorten niet genoeg zijn om alle kracht uit een quantumcomputer te halen. Met deze poorten zijn namelijk maar weinig verschillende superposities te bereiken, om dat aantal te vergroten moet minstens één poort die niet tot de Clifford-groep behoord beschikbaar zijn. In de IBM 5Q zijn T en $T^\dagger$ geïmplementeerd.

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix} \tag{42}$$

$$T^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{-\frac{i\pi}{4}} \end{bmatrix} \tag{43}$$

Ook met één T poort is het niet mogelijk om alle mogelijke superposities te bereiken. Door het verhogen van het aantal T poorten in een circuit, de zogenaamde T-diepte, neemt het aantal te bereiken superposities toe. In figuur 12 is dit gevisualiseerd als zwarte punten op de Blochbol, een bol die alle mogelijke waarden voor een qubit representeerd.

6.6. Waarom is een quantumcomputer sneller dan een gewone computer?

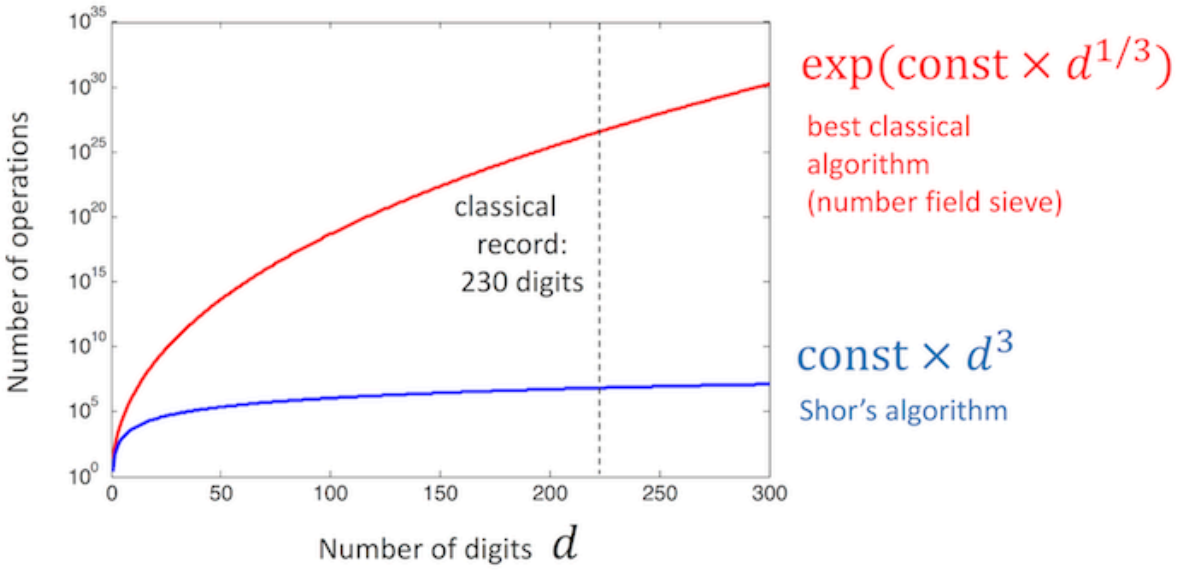
Zoals al vermeld in paragraaf 6.2 is een quantumcomputer niet universeel sneller dan een normale computer, dit is alleen het geval bij het uitvoeren van algoritmes die ontworpen zijn om gebruik te maken van superposities en verstrengelingen. Net als bij een gewone computer zijn er op een quantumcomputer bij een n aantal bits 2^n mogelijke toestanden voor de begin- en eindstaat (die moet gemeten worden, dus valt de qubit altijd terug naar $|0\rangle$ of $|1\rangle$), maar tussendoor zijn er door de mogelijkheden tot superpositie en verstrengeling veel meer mogelijke toestanden, waar een gewone computer ook tussendoor blijft steken op 2^n. Door slim van deze extra ruimte gebruik te maken, kan een quantumcomputer sommige problemen in minder stappen en dus sneller oplossen.

6.7. Algoritme van Shor

In 1994 kwam Peter Shor met een idee om op een efficiëntere manier de priemfactoren van een getal te berekenen met behulp van een quantumcomputer [11]. Het berekenen van priemfactoren is precies datgene wat men moet doen om bij RSA-versleuteling de decryptiesleutel te vinden als de encryptiesleutel bekend is.

In figuur 13 is te zien dat het algoritme van Shor veel beter presteert dan het beste, tot nu toe bekende, klassieke algoritme. Shor maakt gebruik van een periodieke herhaling bij rijen van dit type:

$$x \bmod N, x^2 \bmod N, x^3 \bmod N, x^4 \bmod N, \ldots \tag{44}$$



Figuur 13. Algoritme van Shor tegen het beste klassieke algoritme

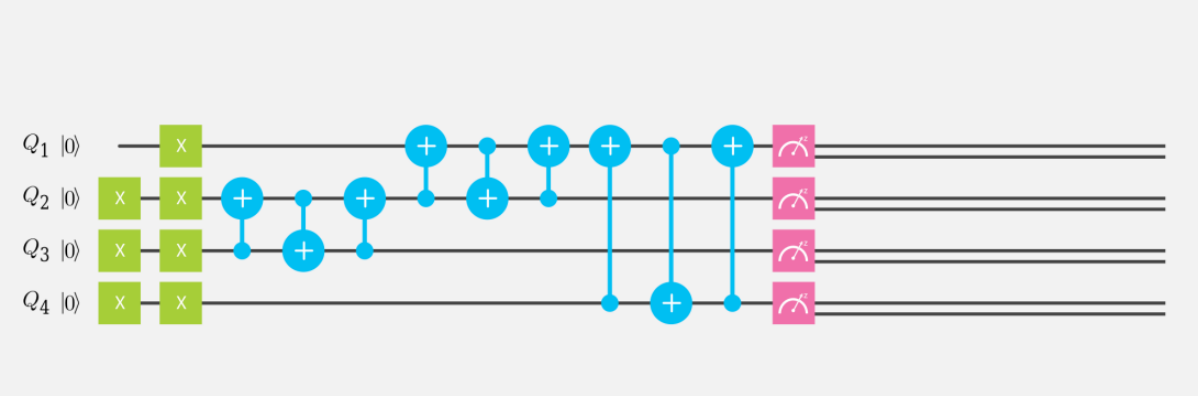
Als N het product van twee priemgetallen p en q is en x is niet deelbaar door p of q, dan is de periode van deze rij, dus na hoeveel getallen de rij zichzelf herhaalt, gelijk is aan een deler van $(p-1)(q-1)$. Door dit vaak te herhalen kunnen meerdere delers van $(p-1)(q-1)$ worden gevonden, en uiteindelijk hiermee $(p-1)(q-1)$ zelf.

Een manier om a^2 mod b uit te rekenen op een quantumcomputer is gegeven in QASM 4. Dit programma is helaas alleen uit te voeren als een simulatie, de IBM 5Q kan alleen qubit Q_2 laten controleren met een CNOT poort. Als resultaat is te zien dat er `0100` uitkomt, omgerekend naar decimale getallen is dit 4.

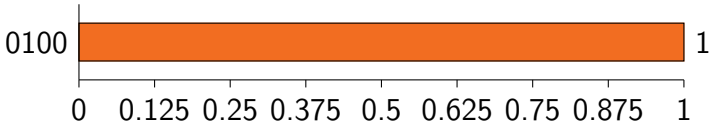
Om de periode van een rij te vinden met een quantumcomputer is het mogelijk om een grote superpositie te maken van alle getallen in die rij. Vervolgens is met een quantum-Fourier-transformatie, die ervoor zorgt dat herhalingen in de rij elkaar uitdoven, zodat uiteindelijk de periode te vinden is. Deze transformatie is helaas niet mogelijk om uit te voeren op de IBM 5Q, waardoor een goed voorbeeld van het algoritme van Shor niet mogelijk is.

Het record voor de uitvoering van het algoritme van Shor is $N = 21$ [12].

De in dit hoofdstuk gebruikte achtergrondinformatie is te vinden in [13], [14] en [15].



(a) QASM



(b) Resultaat

QASM 4. $7^2 \bmod 15 = 4$ (simulatie)

7. Conclusie

Het antwoord op de vraag *'Kan een quantumcomputer de hedendaagse cryptografie veel efficiënter kraken dan een normale computer?'* is niet makkelijk te geven. Bij het kraken van RSA is een quantumcomputer dankzij het algoritme van Shor inderdaad veel efficiënter dan een gewone computer. Over het kraken van AES-encryptie is echter minder makkelijk een uitspraak te doen. In tegenstelling tot RSA is AES wiskundig gezien een heel simpel systeem, waarbij absoluut geen informatie over de sleutel gepubliceerd wordt, zoals bij RSA wel het geval is met de publieke sleutel. Er zijn dus geen speciale berekeningen nodig waarbij superposities veel verschil kunnen maken. Het is misschien wel mogelijk dat een quantumcomputer efficiënter verschillende opties kan proberen, maar ook dan blijft het een *brute-force*-aanval, en dus niet in dezelfde mate efficiënter dan een gewone computer als bij RSA.

Daarnaast maakt het in het huidige systeem ook weinig uit of AES bestand is tegen een quantumaanval, de sleutel voor AES wordt over het algemeen versleuteld met RSA, dus als die gekraakt wordt, kunnen de met AES versleutelde gegevens gewoon met de sleutel ontsleuteld worden. Bij het vervangen van het huidige systeem houd dit gelukkig wel in dat AES niet prompt vervangen hoeft te worden.

Een verleidelijke oplossing voor het probleem met RSA is het vergroten van de getallen waarmee gerekend wordt. Zoals echter in figuur 13 te zien is, heeft dit bij het algoritme van Shor een veel kleinere invloed dan bij het algoritme voor een normale computer. Het is natuurlijk wel mogelijk om de getallen simpelweg reusachtig te maken, waardoor het ook voor een quantumcomputer moeilijk wordt, maar dat houdt ook in dat het berekenen van de sleutels aan de 'goede' kant veel langer gaat duren, dus of deze oplossing echt praktisch is, vind ik dubieus.

Een andere logische stap is natuurlijk het gebruiken van quantumcomputers om gegevens te versleutelen, met behulp van algoritmes die ook gebruik maken van superposities en ver-

strengelingen. Dit lijkt me een goede optie, al is het met belangrijke gegevens natuurlijk wel een beetje eng om die met iets willekeurigs als een quantumcomputer te beveiligen. In de superpositie van een qubit zit nu eenmaal veel willekeurigheid, dat is juist de kracht van deze machines, dus er moeten goede voorzieningen zijn om mogelijke fouten die hierdoor kunnen ontstaan op te vangen.

Wat er ook gaat gebeuren als quantumcomputers beter worden, gezien het record voor het algoritme van Shor nog maar op 21 zit, maak ik me voorlopig geen zorgen.

8. Evaluatie

De komende tijd ga ik de nieuwsberichten over quantumcomputers even ontwijken. Ik heb helemaal geen spijt van mijn onderwerpkeuze, maar na zoveel tijd erin gestoken te hebben is het wel even mooi geweest. Daarnaast zal ik waarschijnlijk de (iets te) versimpelde verklaringen over de werking van quantumcomputers niet meer aankunnen, omdat ik heb geleerd hoe het wel werkt. Als iemand die graag wil weten hoe dingen werken, vind ik het ook leuk om nu door te hebben hoe de beveiliging bij bijvoorbeeld HTTPS werkt. Dit zijn dingen die ik waarschijnlijk anders nooit geleerd had.

Ik vond het ook leuk om te zien hoe sommige dingen uit de lessen nu in de praktijk terugkomen. Een voorbeeld hiervan is de vectorwiskunde, die we vorig jaar bij wiskunde D behandeld hebben. Ik had echt nooit verwacht hier gebruik van te maken bij het rekenen aan quantumcomputers, maar dat schijnt toch wel handig te zijn. Hetzelfde geldt bijvoorbeeld ook voor complexe getallen, weliswaar in mindere mate.

Nog iets wat ik niet verwacht had, is dat de specificatie van RSA [6] best leuk is om te lezen. Omdat het geschreven is in 1978, hadden de auteurs geen idee hoezeer het internet op hun technologie zou steunen en de gebruiksscenarios die zij voorstellen hadden voor mij een hoge amusementswaarde; dat zal waarschijnlijk wel aan mij liggen.

En dan nu nog een alinea over het mislopen van de planning. Hier had ik me op voorbereid en dus bewust zo weinig mogelijk gepland, om te voorkomen dat ik meer tijd zou besteden aan het aanpassen van de planning, dan daadwerkelijk iets uitvoeren. Ik weet dat ik altijd te weinig tijd zal hebben, hoe ruim de planning ook is, maar uiteindelijk komt het ook altijd wel af. Van deze niet-planning heb ik dan ook geen spijt.

Waar ik meer spijt van heb, is dat ik niet eerder ben begonnen met schrijven. Ik heb ervoor gekozen om me eerst overal in te verdiepen, veel te lezen, oefensommetjes maken, af en toe een paar aantekeningen krabbelen, maar achteraf had ik toch beter tijdens deze verdieping al wat dingen kunnen schrijven voor het verslag. Nu moest ik tijdens het schrijven steeds weer ergens diep in mijn geheugen opgraven hoe iets nog maar weer werkte en dat nog maar eens controleren met het bronmateriaal. Het schrijven van het verslag kostte daardoor meer tijd verwacht. (Ook al viel dat te verwachten.)

Al met al kijk terug op een plezierig pws, maar ik ben ook blij dat het nu eindelijk klaar is.

Geraadpleegde literatuur

[1] Sanderson G. (2016). Essence of linear algebra. 3Blue1Brown. Via http://www.3blue1brown.com/essence-of-linear-algebra/

[2] Kahn S. Complex numbers. Kahn Academy. Via https://www.khanacademy.org/math/algebra-home/alg-complex-numbers

[3] Dierks T. en Rescorla E. (2008). The Transport Layer Security (TLS) Protocol. IETF. Via https://tools.ietf.org/html/rfc5246

[4] (2001). Announcing the ADVANCED ENCRYPTION STANDARD (AES). United States National Institute of Standards and Technology. Via http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[5] Daemen J. en Rijmen V. (1999). AES Proposal: Rijndael. Via http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf

[6] Rivest R.L., Shamir A. en Adleman L. (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Via http://people.csail.mit.edu/rivest/Rsapaper.pdf

[7] Diffie W. en Hellman M.E. (1976). New Directions in Cryptography. Via https://www-ee.stanford.edu/~hellman/publications/24.pdf

[8] (2016). D-Wave Systems Previews 2000-Qubit Quantum System. D-Wave Systems. Via http://www.dwavesys.com/press-releases/d-wave-systems-previews-2000-qubit-quantum-system

[9] DiVincenzo D. (1996). Topics in Quantum Computers. Via https://arxiv.org/pdf/cond-mat/9612126v2.pdf

[10] Ostler U. (2016). Wie lassen sich mehrere Hundert Qubits kontrollieren? DataCenter-Insider. Via http://www.datacenter-insider.de/wie-lassen-sich-mehrere-hundert-qubits-kontrollieren-a-561709/

[11] Shor P.W. (1996). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. Via https://arxiv.org/pdf/quant-ph/9508027v2.pdf

[12] Martín-López E., Laing A., Lawson T., Alvarez R., Zhou X. en O'Brien J.L. (2012). Experimental realisation of Shor's quantum factoring algorithm using qubit recycling. Via https://arxiv.org/pdf/1111.4147v2.pdf

[13] Wehner S. en Ng N. (2016). Lecture Notes edX Quantum Cryptography: Week 0. CaltechDelftX. Via https://courses.edx.org/courses/course-v1:CaltechDelftX+QuCryptox+3T2016/pdfbook/0/

[14] IBM Quantum Experience User Guide. IBM Corporation. Via https://quantumexperience.ng.bluemix.net/qstage/#/tutorial?sectionId=c59b3710b928891a1420190148a72cce&pageIndex=0

[15] Aaronson S. (2007). Shor, I'll do it. Via http://www.scottaaronson.com/blog/?p=208

A. Lijst van QASM programma's

1. **Gelijkmatige superpositie (1024 Shots):** https://quantumexperience.ng.bluemix.net/share/code/1618efaf28f7eee0e065d3849a9c5d78 ??
2. **De qubit heeft een neiging naar 0 (1024 Shots):** https://quantumexperience.ng.bluemix.net/share/code/c6f793acc0a9f40de1155025babb811e ??
3. **Demonstratie van de CNOT poort (1024 Shots):** https://quantumexperience.ng.bluemix.net/share/code/e31df3d7d2419178065ea807ad00e951 ??
4. $7^2 \bmod 15 = 4$ **(simulatie):** https://quantumexperience.ng.bluemix.net/share/code/045e18a6577c72a40788f5fda2166f3e ??