Quantum computers kunnen RSA kraken in een tijd die in vergelijking tot een normale computer realistisch is

De kunst van het kraken

Door: Thomas Ammerlaan, Jelle Groenendijk, Jurre Groenendijk, Simon Looij en Wouter van der Welle
Datum: van 15-10-2018 tot 04-02-2019
Begeleider: R. Kustermans

Samenvatting

In dit profielwerkstuk hebben we onderzocht in hoeverre een quantum computer het RSA encryptie algoritme sneller kan kraken dan een normale computer. Dit hebben we gedaan door tests uit te voeren op onze computers en met de resultaten een formule te creëren waarmee we de tijd die nodig is om een getal te factoriseren uit te zetten tegen het aantal bits van het te factoriseren getal. Ook hebben we onderzoek gedaan naar hoe een quantum computer getallen factoriseert. Hierna hebben we onderzocht hoeveel quantum gates hiervoor nodig zijn en hoelang het duurt om die quantum gates uit te voeren. Met deze informatie hebben we ook formules gemaakt waarmee de tijd die een quantum computer nodig heeft om een getal te factoriseren wordt uitgezet tegen het aantal bits van het te factoriseren getal. Uiteindelijk hebben we gevonden dat, afhangend van het aantal bits, quantum computers steeds sneller worden dan computers. Vooral op grote hoeveelheden bits (waar RSA het meest gebruikt wordt), zijn quantumcomputers quadrillioenen malen sneller.

Inleiding

Misschien heb je ooit wel gehoord dat “quantum computers heel gevaarlijk zijn voor onze gegevens” en dat “door quantum computers onze gegevens op straat zullen liggen”. Dit komt omdat heel veel van onze encryptie algoritmes (dat zijn manieren om data te versleutelen) vertrouwen op de moeilijkheid van het factoriseren, dus het in factoren opdelen, van grote getallen. Quantum computers zijn hier veel beter in dan normale computers, dus als er eenmaal goede quantum computers bestaan liggen onze gegevens op straat toch?! Is RSA nog wel veilig?!

Dat is een goede vraag en met ons profielwerkstuk willen wij dichter komen bij het beantwoorden daarvan. Dat doen wij door het beantwoorden van de vraag: “In hoeverre is een quantum computer beter in het kraken van RSA dan een normale computer?”. Wij hebben hier voor RSA gekozen omdat het een veelgebruikte encryptiemethode is waarvan we al wat voorkennis hadden.

Voorkennis

§1 RSA

§1.1 Wat is RSA?

RSA is een “public- and private key encryption systeem”.
Dit betekent dat de twee personen of machines die berichten naar elkaar versturen allebei privé- en openbare sleutels hebben. Deze sleutels worden gebruikt om berichten te versleutelen en te ontsleutelen. De openbare sleutels geef je aan elkaar en de privé sleutel houd je allebei zelf. Als je een bericht naar de andere persoon wilt sturen, kan je dat bericht met zijn openbare sleutel versleutelen en kan hij dat met zijn privé sleutel ontsleutelen. Die andere persoon kan ook een bericht naar jou sturen door het te versleutelen met jouw openbare sleutel. Hierna kan jij het weer ontsleutelen met jouw privé sleutel. Waarom dit werkt is ingewikkeld, maar aangezien wij de wiskunde later nodig gaan hebben moeten wij dit wel even uitleggen en een voorbeeld geven.

§1.2 Hoe werkt RSA?

Tekens

Allereerst is zijn er een aantal tekens die wij zullen gebruiken voor RSA:

Teken:	Wat het inhoud:	In ons voorbeeld:
p	Een priemgetal	5
q	Een ander priemgetal	11
n	$n = q \cdot p$	$5 \cdot 11 = 55$
$\varphi(n)$,ook wel totiënt van n	Dit houdt in: het aantal positieve natuurlijke getallen kleiner dan of gelijk aan n die onderling deelbaar zijn met n $\varphi(n) = (q-1)(p-1)$	$(11-1) \cdot (5-1) = 10 \cdot 4 = 40$
e	Een priemgetal wat kleiner is dan n en geen gemeenschappelijke deler heeft met $\varphi(n)$	$40 = 2^3 \cdot 5$,dus nemen we 7 (maar je kan ook 11, 13 en elk ander priemgetal nemen wat kleiner is dan 5).
d	Het kleinste getal (d) wat aan de formule $(e \cdot d) \ mod \ \varphi(n) = 1$ voldoet.	Uiteindelijk komt d uit op 23.
$b \ mod(a)$, modulo	Er wordt steeds het getal a van b afgetrokken totdat dat niet meer mogelijk is. Hetgeen wat je overhoudt is de uitkomst van de modulo.	-
$\equiv$	Een $\equiv$ betekent hetzelfde als "$= \ mod \ x$", dus wanneer je een keer $= \ mod \ x$ heb opgeschreven, kun je daarna gewoon $\equiv$ opschrijven en hoe je niet meer iedere keer $= mod \ x$ op te schrijven.	-

$[x]$	De blokhaken zorgen ervoor dat het getal x naar beneden wordt afgerond. [2,913] wordt dus 2 en [3,5213] wordt 3.	-

Voor RSA zijn twee sleutels nodig, de publieke en de privé sleutel. De publieke sleutel bestaat uit n en e, en de privé sleutel bestaat uit n en d. In de eerste stap laten wij zien hoe een publieke sleutel wordt gemaakt en de tweede stap laten we zien hoe de bijbehorende privé sleutel wordt gemaakt.

Stap 1: Het maken van de publieke sleutel

Als eerste gaan we de publieke sleutel maken. Je kiest twee priemgetallen, p en q. Deze priemgetallen heb je nodig om de publieke sleutel te maken. Hierna vermenigvuldig je deze priemgetallen met elkaar. Hieruit krijg je n. Dit is al een deel van de publieke sleutel. Hierna trek je 1 van p en q af, en vermenigvuldig je die twee getallen om $\varphi(n)$ te maken. Dus om het even als een formule te schrijven: $\varphi(n) = (q-1)(p-1)$. Nu moet je nog twee dingen doen om het laatste deel van de publieke sleutel te maken. Schrijf $\varphi(n)$ op als het product van zijn priemfactoren. En neem als laatste een priemgetal wat geen deel is van de priemfactoren van $\varphi(n)$ en kleiner is dan n. Dit getal noem je e, en dit is je laatste deel van de publieke sleutel.

Voorbeeld:
Om te beginnen hebben wij die twee priemgetallen, p en q. Voor onze p en q nemen wij $q = 11$ en $p = 5$. Nu hebben we n nodig. $n = q \cdot p = 11 \cdot 5 = 55$. Mooi, dan hebben we die ook. Nu nog $\varphi(n)$ en e. $\varphi(n) = (q-1) \cdot (p-1) = (11-1) \cdot (5-1) = 40$. En als we $\varphi(n)$ uitschrijven als priemfactoren is het $\varphi(n) = 40 = 2^3 \cdot 5$. Nu hebben we een priemgetal nodig wat geen 2 en geen 5 is, en lager is dan 55. Dus kiezen wij in dit voorbeeld voor onze e een 7. Onze publieke sleutel is dus $n = 55$ en $e = 7$.

Stap 2: Het berekenen van de privé sleutel

We hebben voor de privé sleutel nu alleen d nog nodig. d kunnen we vinden door het uitgebreide algoritme van Euclides te gebruiken. Dit gebeurt in een aantal stappen die wij nu gaan uitleggen.

Stap 2.1: Het uitgebreide algoritme van Euclides

d is het kleinste getal wat voldoet aan $(e \cdot d) \bmod \varphi(n) = 1$. Dit moet zo zijn omdat we deze formule gebruiken in het bewijzen van RSA in paragraaf 1.4. De formule $(e \cdot d) \bmod \varphi(n) = 1$ transformeren we naar de vorm $ey + \varphi(n)x = 1$ zodat wij deze kunnen gebruiken in het uitgebreide algoritme van Euclides.

Dit kan omdat de formule $(e \cdot d) \bmod \varphi(n) = 1$ eigenlijk betekent "e keer d min iets keer $\varphi(n)$ is 1", volgens de definitie van modulo. Als we d, y noemen en dat iets, $-x$, dan krijg je $ey - -\varphi(n)x = 1$, dus krijg je $ey + \varphi(n)x = 1$

Voorbeeld:

$(e \cdot d) \bmod \varphi(n) = 1$

$(7 \cdot d) \bmod 40 = 1$

$7y - -40x = 1$

$7y + 40x = 1$

Stap 2.2: Het toepassen van het uitgebreide algoritme van Euclides (deel 1)

Het doel van dit deel van het algoritme is het krijgen van een aantal vergelijkingen die we nodig hebben voor het tweede deel. Als eerste moeten we de e en $\varphi(n)$ uit de eerder genoemde formule $e \cdot y + \varphi(n) \cdot x = 1$ nemen en hier de e, a noemen en de $\varphi(n), b$ noemen.

Deze a en b voeren we in in de formule $b = [b/a] \cdot a + (b \, mod(a))$. De formule $b = [b/a] \cdot a + (b \, mod(a))$ is misschien beter te begrijpen als je het ziet als $Totaal = aantal\ delen \cdot deel + restant.$ Hierna schrijven we deze vergelijking ergens op want deze hebben we nodig voor het tweede deel van het algoritme.

Vervolgens nemen we het vierde deel van deze vergelijking (de uitkomst van $(b \, mod \, (a))$) en noemen we dat onze nieuwe a en nemen we het derde deel (onze oude a) en noemen we dat onze nieuwe b. Deze a en b vullen we weer in in de formule $b = [b/a] \cdot a + (b \, mod \, (a))$. De vergelijking die we daaruit krijgen schrijven we weer ergens op, we noemen de vierde en derde delen onze nieuwe a en b en vullen die weer in. Dit blijven we doen tot het vierde deel 1 is. Dan stoppen we, schrijven we de laatste vergelijking ook over en nemen we de vergelijkingen die we hebben opgeschreven en gaan we daarmee door naar het tweede deel van het algoritme.

Voorbeeld:
In ons voorbeeld hebben we de formule $7y + 40x = 1$. Onze eerste a is dus 7, en onze eerste b is 40. Deze vullen we in de formule $b = [b/a] \cdot a + (b \, mod(a))$ in. Daar komt uit: $40 = 5 \cdot 7 + 5$. Deze vergelijking schrijven we op zodat we hem later kunnen gebruiken. Onze nieuwe a is het vierde deel van deze vergelijking, dus 5, en onze nieuwe b is het derde deel van deze vergelijking, dus 7. Deze vullen we weer in in die formule en dit geeft: $7 = 1 \cdot 5 + 2$. Deze vergelijking schrijven we ook weer op. Onze nieuwe a is 2 en onze nieuwe b is 5. Deze vullen we weer in en dat geeft: $5 = 2 \cdot 2 + 1$. Het restant is 1 dus deze vergelijking schrijven we op en we zijn klaar met deze stap. Onze opgeschreven lijnen zijn:
$40 = 5 \cdot 7 + 5$,
$7 = 1 \cdot 5 + 2$ en
$5 = 2 \cdot 2 + 1$

Stap 2.3: Het omschrijven van de vergelijkingen

Nu staan alle vergelijkingen in de vorm: $totaal = aantal\ delen \cdot deel + restant$. Deze gaan we omschrijven naar $restant = totaal - aantal\ delen \cdot deel$. Omdat we bij de laatste formule altijd eindigen met 1 als restant, schrijven we onze laatste formule bijvoorbeeld op als $1 = totaal - aantal\ delen \cdot deel$. Dit omschrijven doen we voor alle vergelijkingen in ons lijstje. Als laatste zetten we alle vergelijkingen op omgekeerde volgorde.

Voorbeeld:
We hebben nu de formules:
$40 = 5 \cdot 7 + 5$,
$7 = 1 \cdot 5 + 2$ en
$5 = 2 \cdot 2 + 1$.
Na het omschrijven wordt dit:
$5 = 40 - 5 \cdot 7$,
$2 = 7 - 1 \cdot 5$ en
$1 = 5 - 2 \cdot 2$.
En na het omkeren van de volgorde wordt dit:
$1 = 5 - 2 \cdot 2$,
$2 = 7 - 1 \cdot 5$ en
$5 = 40 - 5 \cdot 7$.

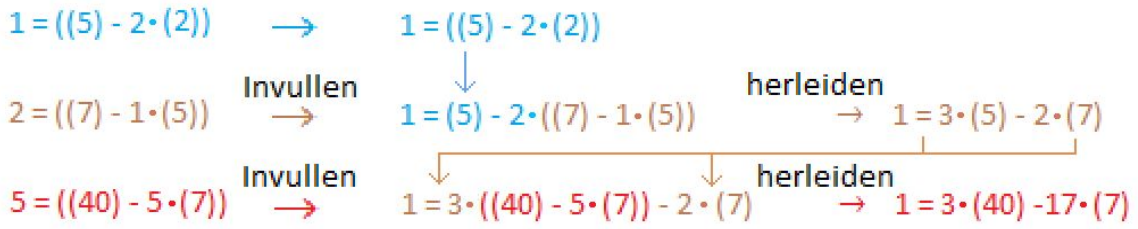
Stap 2.4: Het toepassen van het uitgebreide algoritme van Euclides (deel 2)

$1 = 5 - 2 \cdot 2,$
$2 = 7 - 1 \cdot 5$ en
$5 = 40 - 5 \cdot 7.$

Figuur 1. De getallen met gelijke kleuren geven aan waar gesubstitueerd moet worden

Wat dan opvalt is dat het aantal delen (het vierde deel) van een vergelijking gelijk is aan de restant (het eerste deel) van de vergelijking erna, zoals bij ons te zien in figuur 1. Wat we dus nu gaan doen is de vergelijkingen in elkaar substitueren. Dit doen we door het aantal delen (het vierde deel) te substitueren door datgene dat gelijkgesteld wordt aan het restant, dus deel 2, 3 en 4, bij de vergelijking erna. Hierna werken we de haakjes weg en herleiden we. Daarna houden we een vergelijking over in de vorm $ay + bx = 1$. Nu zien we dat het restant van de volgende lijn hetzelfde is als de x of de y van je nieuwe formule.
We substitueren dan de x of de y, afhankelijk van welke gelijk staat aan het restant, met datgene wat gelijkgesteld wordt aan dat restant, dus deel 2, 3 en 4, van de nieuwe lijn. Dit blijven we doen totdat we door alle vergelijkingen heen zijn en we uitgekomen zijn op de vorm $e \cdot y + \varphi(n) \cdot x = 1$.

Voorbeeld:



Stap 2.5: Het vinden van d

Nu we $e \cdot y + \varphi(n) \cdot x = 1$ hebben hoeven we alleen nog maar de formule $d = y$ of $d = y + \varphi(n)$ te gebruiken.
Wij gebruiken $d = y + \varphi(n)$ als y negatief is en $d = y$ als y positief is.

Voorbeeld:
Wij houden dus de formule $1 = 3 \cdot 40 - 17 \cdot 7$ over. Aangezien onze e 7 is, is y dus -17. Dit getal is negatief, dus gebruiken we de formule $d = y + \varphi(n)$.
$d = y + \varphi(n) = -17 + 40 = 23$. Onze d is dus 23. Dit maakt samen met onze n van 55 onze privé sleutel. (Cormen, Leiserson, Rivest, & Stein, 2011; Rivest et al., 1978)

§1.3 Het versleutelen en ontsleutelen met RSA

Nu we de privé sleutel, ($d = 23$ en $n = 55$), en de publieke sleutel, ($n = 55$ en $e = 7$) hebben, kunnen we door naar het versleutelen en ontsleutelen van berichten.

Voor het versleutelen van een bericht heb je de publieke sleutel nodig en voor het ontsleutelen de privé sleutel.

Als je een bericht wil versleutelen, dan stuurt de andere persoon zijn publieke sleutel naar jou. Daarmee versleutel jij het bericht en stuur je het terug naar de andere persoon die het met zijn privé sleutel weer kan ontsleutelen. Omdat de andere persoon zijn privé sleutel niet heeft gestuurd heeft hij ook d niet gestuurd. Hierdoor kan iemand die meeleest wat jullie versturen het bericht niet ontsleutelen, want daarvoor heb je d nodig.

Het versleutelen en ontsleutelen gebeurt als volgt:

Als eerst hebben wij een bericht nodig, wij gebruiken voor dit voorbeeld het getal 42. Het nummer dat wij versturen mag niet groter zijn dan n.
Nu wij een bericht hebben, kunnen wij deze samen met de publieke sleutel $(e = 7,\ n = 55)$ invoeren in de formule:

$$bericht^{e}\ mod\ (n)\ =\ versleuteld\ bericht$$

Na het invoeren ziet het er zo uit:

$$42^{7}\ mod\ 55\ =\ 48$$

48 is dus ons versleutelde bericht.

Het versleutelde bericht is met de volgende formule te ontcijferen:

$$versleuteld\ bericht^{d}\ mod\ (n)\ =\ bericht$$

Na het invoeren van het versleutelde bericht samen met de privé sleutel $(d = 23,\ n = 55)$ ziet het er zo uit:

$$48^{23}\ mod\ 55\ =\ 42$$

En zo hebben wij ons originele bericht, namelijk 42, weer terug
(Rivest et al., 1978).

§1.4 Het bewijs voor RSA

De reden dat dit werkt is niks meer dan wat snelle wiskunde.
Voor gebruiksgemak zeggen we dat het versleutelde bericht C (code) is en het originele bericht M (message) is. Verder is het belangrijk te herinneren dat we hier soms $\equiv$ gebruiken, dus overal waar een $\equiv$ staat moet je “mod iets” achter denken. Ook wordt Fermat’s little Theorem gebruikt, en dat zegt: Voor ieder priemgetal p, en iedere a die niet 0 is, geldt dat $a^{p-1} = 1 \ mod \ p$.

Het is genoeg om te bewijzen dat $C^d = M \ mod \ p$ en $C^d = M \ mod \ q$, want die leiden samen naar $C^d = M \ mod \ pq = M \ mod \ n$, met behulp van de Chinese Reststelling, dus dat gaan we nu doen.

Als M niet deelbaar is door p of q is, geldt deze uitleg:

$C^d \equiv (M^e)^d \ mod \ p$	Want $C \equiv M^e$. Verder laten we hierna $mod \ p$ weg door de $\equiv$
$\equiv M^{ed}$	Want $(x^a)^b = x^{ab}$
$\equiv M^{1 \ mod \ \varphi(n)}$	Want $ed = 1 \ mod \ \varphi(n),$ want we hadden d gekozen zodat dit zo zou zijn.
$\equiv M^{k \cdot \varphi(n) + 1}$	Want $1 \ mod \ \varphi(n) = 1 + k \cdot \varphi(n)$ door de definitie van modulo
$\equiv (M^{\varphi(n)})^k \cdot M$	Want $x^{y+1} = x^y \cdot x$
$\equiv (M^{(p-1)(q-1)})^k \cdot M$	Want $\varphi(n) = (p-1)(q-1)$
$\equiv (M^{(p-1)})^{k(q-1)} \cdot M$	Want $(x^{ab})^c = (x^a)^{bc}$
$\equiv (1)^{k(q-1)} \cdot M$	Want Fermat’s Little Theorem zegt $a^{(p-1)} \ mod \ p = 1$.
$\equiv M$	Want $1^{k(q-1)} = 1$

En door de symmetrie van de formule, geldt dit bewijs ook voor $C^d \equiv M \ mod \ q$ en omdat hij nu geldt voor $C^d \equiv M \ mod \ q$ en $C^d \equiv M \ mod \ p$ en omdat $n = p \cdot q$, geldt $C^d \equiv M \ mod \ n$ dankzij de Chinese reststelling.

En dan nu voor het geval dat M deelbaar is door p (of q, ze zijn verwisselbaar):
Dat betekent dat M tot de macht ieder positief getal deelbaar is door p, en daardoor geldt $M^{ed} \ mod \ p = 0 \ mod \ p = M \ mod \ p$. Dit geldt ook voor $M^{ed} \ mod \ q$ en daardoor ook voor $M^{ed} \ mod \ n$. Dit zorgt ervoor dat $C^d \equiv M \ mod \ n$ altijd waar moet zijn.

Nu hebben we dus bewezen dat $C^d mod \ n = M \ mod \ n$ altijd waar is, en aangezien M kleiner is dan n geeft dat $C^d mod \ n = M$
(Rivest et al., 1978).

§1.5 Methode voor kraken RSA op klassieke computer

De reden dat RSA zo moeilijk te kraken is, is omdat als je het wil ontsleutelen, je d nodig hebt. Om d uit te rekenen heb je $\varphi(n)$ en e nodig. Je hebt e en n, maar om bij $\varphi(n)$ te komen, moet je eerst n terug-factoriseren naar p en q, en dit is voor normale computers heel moeilijk (Rivest et al., 1978)
Voor quantum computers is dit veel simpeler. Daarom gaan wij in dit profielwerkstuk uitzoeken in hoeverre een quantum computer sneller is in het kraken van RSA dan een normale computer.

Maar eerst: Hoe doet een normale computer het?
Een normale computer factoriseert n in p en q met behulp van de *quadratic sieve*, en die werkt op de volgende manier

1. Je neemt een nummer, n, dat je wilt factoriseren. In ons geval nemen we $n = 63787$
2. Dan neem je meerdere nummers, bijvoorbeeld de nummers 430 tot 450, en kijken we naar de priemfactoren van de kwadraten $mod\,(n)$ ervan. Hierbij is het belangrijk dat alle priemfactoren lager zijn dan een getal B, voor ons gebruiksgemak kiezen we de getallen lager dan 19.
   $439^2 \equiv 2^4 \cdot 5^1 \cdot 17^1$
   $441^2 \equiv 2^4 \cdot 3^1 \cdot 5^1 \cdot 13^1$
   $444^2 \equiv 3^1 \cdot 5^2 \cdot 7^1 \cdot 11^1$
   $445^2 \equiv 2^3 \cdot 7^2 \cdot 17^1$
   $447^2 \equiv 2^8 \cdot 3^1 \cdot 11^1$
   $449^2 \equiv 2^{11} \cdot 5^1$
3. Daarna kijk je welke getallen je met elkaar kun vermenigvuldigen om een perfect kwadraat te krijgen, met andere woorden, welke nummers je bij elkaar kan voegen zodat alle exponenten even worden. In ons geval zijn dat de eerste, vierde en zesde nummers:
   $439^2 \cdot 445^2 \cdot 449^2 \equiv (2^4 \cdot 5 \cdot 17)(2^3 \cdot 7^2 \cdot 17)(2^{11} \cdot 5) = (2^{18} \cdot 5^2 \cdot 7^2 \cdot 17^2)$
   En zoals je ziet zijn daar alle exponenten even nummers, en dus kan je daarvan maken:
   $(2^9 \cdot 5 \cdot 7 \cdot 17)^2 \equiv 49492^2$
   Je kan het eerste gedeelte van de formule 3 lijnen terug herschrijven als:
   $(439 \cdot 445 \cdot 449)^2 \equiv 7270^2$
   En dus is de conclusie:
   $7270^2 \; mod\, n = 49492^2 \; mod\, n$

4. Als laatst neem je het verschil tussen die 2 nummers, en kijk je wat de Grootste Gemeenschappelijke Deler van de twee is:
   $GGD(49492 - 7270, 63787) = GGD(42222, 63787) = 227$
   En dat is een van de twee delers van 63787! De andere kun je dan makkelijk uitrekenen met $63787/227 = 281$
   Dus om te concluderen, $63787 = 227 \cdot 281$
5. Als het niet heeft gewerkt heb je pech en moet je helemaal opnieuw beginnen met een andere set getallen.

§2 Quantum Computers

Om onze hoofdvraag te kunnen beantwoorden moeten we onderzoek doen naar wat de relatie is tussen het aantal bits van n, en de tijd die de quantum computer nodig heeft om die n te factoriseren naar q en p.

§2.1 Wat zijn quantum computers?

Een quantum computer is op zich niet veel meer dan een normale computer die in plaats van normale bits die een toestand van 1 of 0 (aan of uit) kunnen hebben, qubits (ook wel quantum bits) gebruikt.
Qubits hebben drie dingen die ze anders maakt dan normale bits:

- Superposities: Qubits kunnen niet alleen een waarde van 1 of 0 hebben, zoals klassieke bits, maar ook iets er tussenin. Als de waarde een qubit tussen de 1 en de 0 in staat, heet dit een superpositie. Een superpositie betekent dat de waarde van het deeltje niet 1 is en ook niet 0 is, maar beide tegelijk. Je kan ook kijken naar een aantal deeltjes als systeem wat in een een superpositie zit.

  Voorbeeld: als je een systeem hebt met twee qubits die allebei in een superpositie zitten met 50% kans op een 1 en 50% kans op een 0, dan zit het hele systeem in de superpositie: 25% kans op 00, 25% kans op 01, 25% kans op 10 en 25% kans op 11.

- Metingen: Het moment dat je gaat kijken welke van de twee het is, vervalt de superpositie naar een waarde

  Voorbeeld: stel je voor: ik heb een qubit met 50% kans om een waarde van 1 te hebben en 50% kans om een waarde 0 te hebben en ik ga hem meten, dan komt er bijvoorbeeld toevallig deze keer een waarde van 1 uit. Het is niet zo dat de qubit altijd al de waarde 1 had en we het niet wisten, maar hij zat echt in een superpositie totdat we het meetten.

- Verstrengeling: Qubits kunnen ook met elkaar verstrengeld zijn. Dit betekent dat de deeltjes samen in een superpositie zitten. Als je de ene meet, vervalt de andere ook direct. Wanneer je tweede deeltje vervalt kan je weten door naar je eerste deeltje te kijken.

  Voorbeeld: als je een systeem maakt van twee qubits met een verstrengelde superpositie met 50% kans op de waarde 00 en 50% kans op de waarde 11. Het moment dat je het eerste deeltje meet en er komt een 0 uit dan vervalt de andere ook direct naar een 0 ongeacht de afstand, Hetzelfde zou gebeuren met twee enen.

Deze qubits zijn deeltjes zoals bijvoorbeeld elektronen. Elektronen kunnen op twee manieren draaien en de toestand “0” betekent bijvoorbeeld dat hij op de ene manier draait en de toestand “1” betekent dat hij op de andere manier draait. Quantum computers zijn machines die dus dit soort quantum(wat ‘heel klein’ betekent) gebeurtenissen gebruiken om dingen te berekenen.

Dit moet heel koud gebeuren, want de gebeurtenissen zijn heel klein, dus zijn ze heel moeilijk te lezen. Atomen die naast zo een elektron trillen omdat ze warmte hebben geven ook energie af en die maken de draai van zo’n elektron daarom heel moeilijk te lezen. Daarom moeten quantum computers zo koud mogelijk gemaakt worden. Dit wordt op verschillende manieren gedaan, waarvan een is: het laten stromen van helium die de warmte rond de deeltjes opneemt en de hele cpu van de quantum computer af kan koelen naar 15 millikelvin (dus $1.5 \cdot 10^{-2}$ Kelvin of -273,14 ℃) ("Introduction to the D-Wave Quantum Hardware | D-Wave Systems", z.d.).

Zelfs op 15 millikelvin maken ze soms fouten. Deze foutmarge hangt af van het type qubit wat je gebruikt. Google heeft een qubit uitgevonden met een foutmarge van 0,1% voor gates die gebruik maken van één qubit, en een foutmarge van 0,6% voor gates die gebruik maken van twee qubits (Kelly, 2018). Dit is een van de grote problemen van quantum computers, want je kan niet weten welk antwoord fout is en welk antwoord goed is. Bij ons is het gelukkig geen groot probleem. Wij gebruiken het namelijk alleen maar voor het factoriseren van priemgetallen en als wij een priemgetal hebben, kunnen wij het eerste getal (n) delen door het priemgetal wat wij hebben gekregen. Als er een ander priemgetal uitkomt, dan zijn deze priemgetallen p en q en dan kunnen wij d berekenen, en als dat niet zo is dan heeft de quantum computer een fout gemaakt en moet hij het simpelweg opnieuw proberen.

Quantum computers zijn sneller dan klassieke computers omdat ze heel veel berekeningen tegelijk kunnen doen door gebruik te maken van superposities. Het probleem hiervan is wel dat je maar één antwoord kan krijgen en niet alle antwoorden als je gaat meten. Gelukkig zijn er functies zoals de Quantum Fourier Transform die ervoor kunnen zorgen dat het antwoord dat de quantum computer laat zien waarschijnlijk het antwoord is wat je wilt, zolang je probleem maar een probleem is wat quantum computers op kunnen lossen. Quantum computers zijn namelijk wel heel krachtig, maar door hoe ze werken kunnen ze maar gebruikt worden voor een heel klein aantal dingen (Hayward, 2008).

§2.2 Wat zijn quantum gates?

Net als normale circuits die bestaan uit gates zoals de NOR of de AND gate, bestaan quantum computers ook uit gates. De gates van een quantum computer zijn gemaakt om Qbits te gebruiken in plaats van normale bits. We gebruiken vijf gates in dit profielwerkstuk, namelijk de Hadamard gate, de CNOT gate, de NOT gate, de Controlled Phase gate en de Toffoli gate. Het is niet nodig om te weten wat ze doen maar volledigheidshalve leggen we het toch even uit.

- De Hadamard gate brengt qubits in en uit een superpositie.
- De NOT gate flipt de status van een qubit (dus als hij 10% kans heeft op een 0 en 90% kans op een 1 dan heeft hij na de NOT gate een 90% kans op een 0 en een 10% kans op een 1).
- De CNOT en Controlled Phase gates zijn gates die een NOT gate en een Phase gate uitvoeren als een andere qubit (de controle qubit) zich in toestand 1 bevindt.
- Als laatste is de Toffoli gate een gate die op drie qubits werkt en als de twee control qubits zich in toestand 1 bevinden, een NOT gate uitvoert op de derde (Nielsen & Chuang, 2010; Barenco et al., 1995).

§2.3 Methode voor kraken RSA op quantum computer (klassieke deel)

Quantum computers gebruiken een algoritme genaamd “Shor’s Algorithm”. Dit algoritme is zeer complex. Daarom kiezen we ervoor om niet uit te leggen waarom het algoritme werkt (want dat is ook niet nodig voor onze onderzoeksvraag), maar wel hoe het werkt. Heel versimpeld werkt het zo:

1. Kies een willekeurig getal tussen 1 en het getal wat je wilt factoriseren. Dat gekozen getal noem je a, en het getal wat je wilt factoriseren noem je N.
2. Bereken de grootste gemene deler van a en N. Hiervoor heb je weer het algoritme van Euclides nodig.
3. Als de grootste gemene deler van a en N niet 1 is, dan is a een factor van N en ben je dus klaar.
4. Zo wel, dan moet je de periode berekenen van de functie $f(x) = a^x \ mod \ N$. Deze functie herhaalt zich iedere zoveel tijd, en het aantal tijdseenheden waarna deze functie zich herhaalt heet de periode. Het vinden hiervan doen we met het “quantum period finding subroutine”.
5. Uit het quantum period finding subroutine komt een nummer, wat, door gebruik te maken van kettingbreuken en wat algebra, een mogelijke kandidaat voor de periode geeft. Deze periode kunnen we checken door te kijken of de vergelijking $f(x) = f(x + p)$, met mogelijke periode als p, klopt.
6. Als deze periode oneven is, dan moet je terug naar stap 1 en opnieuw beginnen.
7. Als $a^{r/2} \ mod \ N \ = \ -1$, dan moet je ook terug naar stap 1 en opnieuw beginnen.
8. Zo niet, dan zijn de grootste gemene deler van $a^{r/2} + 1$ en N, en de grootste gemene deler van $a^{r.2} - 1$ en N de priemfactoren van N, oftewel p en q
9. Aangezien quantum computers relatief vaak fouten maken, is het handig als je even checkt of je p en q hebt door te kijken of N deelbaar is door een van de twee, en zowel, dat je als uitkomst de andere krijgt. Zo niet, dan moet je ook terug naar stap 1.
10. Nu je dus de p en q hebt kan je dus $\varphi(n)$ berekenen, waarmee je d kan berekenen.

Met al deze gegevens kan je de boodschap ontsleutelen en heb je RSA gekraakt.

Dit “quantum period finding subroutine” is het deel wat veruit het langst duurt, en daarom checken we later hoe lang dat duurt in plaats van hoelang het hele proces duurt. (Shor. 1996)

§2.4 Methode voor kraken RSA op quantum computer (quantum deel)

In stap vijf van het de vorige paragraaf, stond ‘Het vinden hiervan doen we met het “quantum period finding subroutine”’. Maar wat is dit nou eigenlijk?
Het quantum period finding subroutine is een manier om op een quantum computer de periode van een functie te vinden. Dit is ook het enige deel van Shor’s algoritme wat op een quantum computer uitgevoerd moet worden. De subroutine zit als volgt in elkaar:
(even een opfrisser: N is het te factoriseren getal en a is een willekeurig getal kleiner dan N)

1. Vind een getal q, waarbij $N^2 < 2^q < 2N^2$. Dus bijvoorbeeld bij $N = 131$, moet je een getal vinden waarbij 2 tot de macht dat getal tussen $131^2 = 17161$ en $2 \cdot 131^2 = 343222$ ligt. Dit kan bijvoorbeeld 15 zijn want $2^{15} = 32768$ en dat ligt tussen 17161 en 34322.
2. Neem een quantum computer met q qubits voor de input en q qubits voor de output, dus in ons geval een quantum computer met 30 qubits.
3. Zet alle input qubits in stand 0 en alle output qubits in stand 1 en pas op alle input qubits een Hadamard gate toe
4. Pas hierna de functie $f(x) = a^x \ mod \ N$ toe op de output qubits met als input de input qubits.
5. Pas een transformatie genaamd Quantum Fourier Transformatie toe op de input qubits.
6. Meet de input qubits. Dit geeft een getal waarmee de periode mogelijk berekend kan worden. (Shor ,1996)

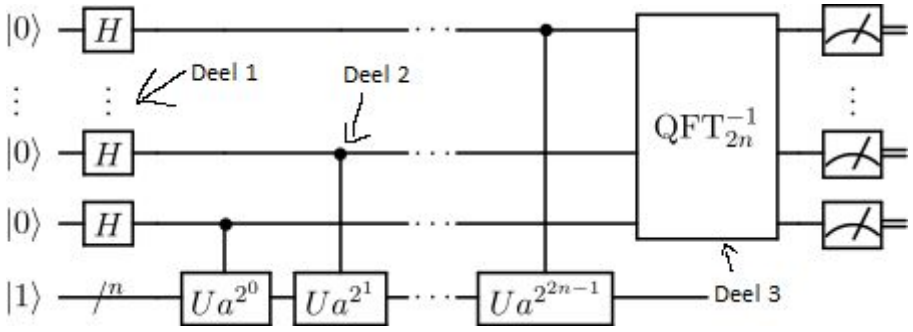
§3 Tijdsduur Shor's algoritme

Nu we weten hoe Shor's algoritme werkt, hoeven we alleen nog maar te weten hoe lang hij duurt.

§3.1 Hoeveel quantum gates heb je nodig voor Shor's algoritme?

Om deze vraag te beantwoorden moeten we het "period finding subroutine" opdelen in verschillende delen, en kijken hoeveel gates nodig zijn voor ieder deel.

Deel 1: pas Hadamard gates toe op de input qubits.
Deel 2: pas $x^a mod\, n$ toe op de output qubits met als input de input qubits.
Deel 3: pas "Quantum Fourier Transformation" toe op input qubits.
Deel 4: meet de qubits.



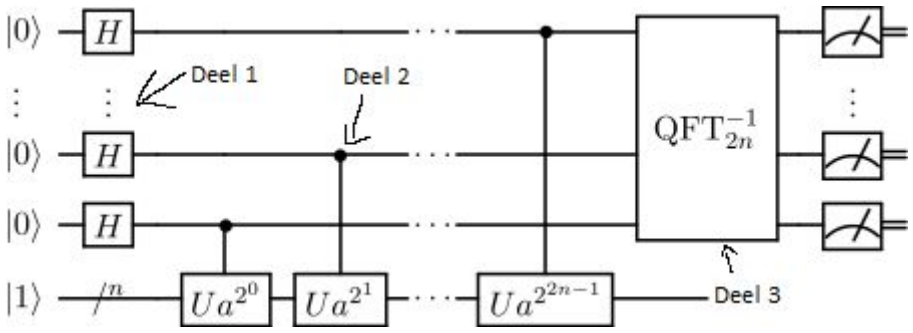
Figuur 2, het schema van Shor's Algoritme (Trenar3, z.d.)

Nu we weten hoe het in elkaar zit, kunnen we uit reeds uitgevoerde onderzoeken halen hoeveel quantum gates er nodig zijn voor iedere stap.
Voor deel 1 heb je simpelweg n Hadamard gates nodig als je n qubits hebt
Voor deel 2 heb je ongeveer n^3 NOT, CNOT en Toffoli gates nodig. Het aantal gates verschilt met andere waardes a en N, dus het precieze aantal is niet vast te stellen.
Voor deel 3 heb je n Hadamard gates nodig, en $(n^2 - n)/2$ Controlled Phase gates.
En voor deel 4 heb je geen gates nodig.

Totaal zijn dit dus $2n$ Hadamard gates, $(n^2 - n)/2$ Controlled Phase gates en n^3 NOT, CNOT of Toffoli gates ("Shor's factoring algorithm", 2015; Rivest et al., 1978; Markov & Saeedi, 2015).

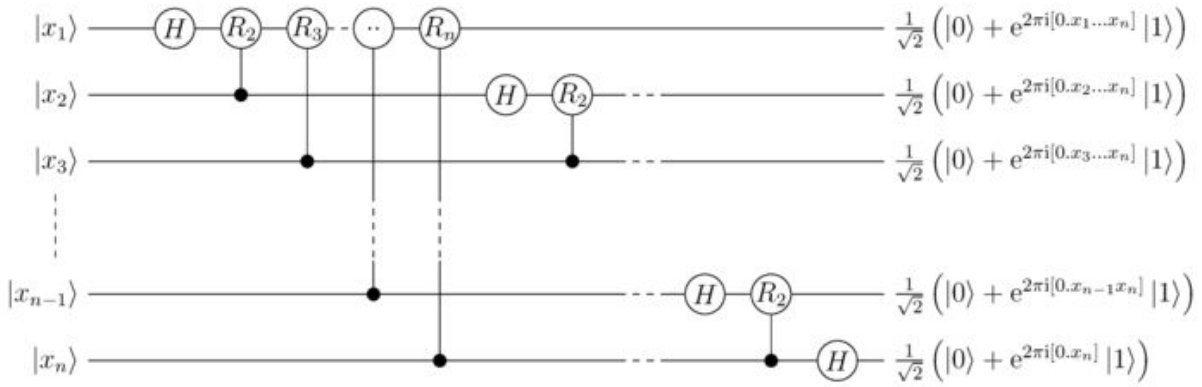
§3.2 De formule voor het aantal gates per bit

Mooi. We weten nu hoeveel quantum gates er nodig zijn. Alleen voer je deze gates niet allemaal achter elkaar uit, maar sommige ook parallel.

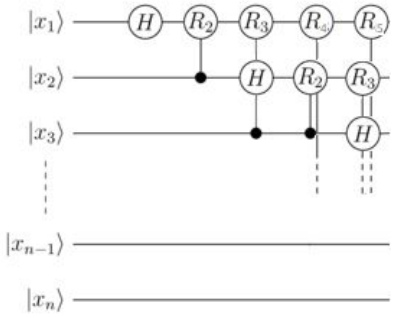


Figuur 2, het schema van Shor's Algoritme (Trenar3, z.d.)

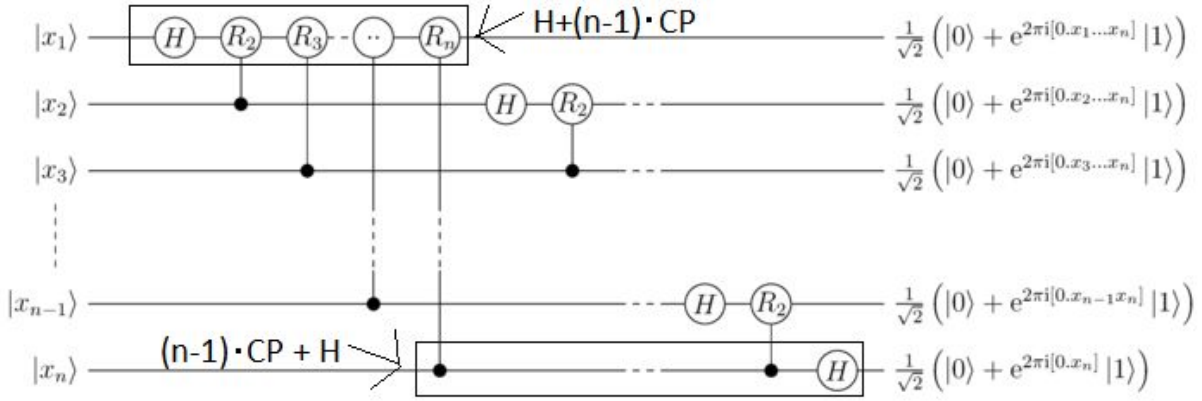
- Deel 1 kan uitgevoerd worden in de tijd die het duurt om 1 Hadamard gate uit te voeren, want ze staan allemaal parallel
- Deel 2 kan variëren tussen $n^3 \cdot$ *de langste duur van* NOT, $CNOT$ *en* $Toffoli$ *gates*, en $n^3 \cdot$ *de kortste duur van die gates*. Het kan zelfs nog sneller als alles altijd parallel uitgevoerd wordt, want dat zorgt namelijk dat het n keer zo snel wordt aangezien het op n lijnen wordt uitgevoerd in plaats van op 1, dus is de totale tijdsduur de tijdsduur van n^2 gates.
- Deel 3 is wat ingewikkelder, dus we nemen even wat meer tijd om dat te bespreken.



Figuur 3. Het schema van een quantum fourier transformatie waarbij de H's Hadamard gates vertegenwoordigen en de R's Controlled Phase gates. (Bender2k14, 2014)



Figuur 4. Een visuele representatie van het horizontaal comprimeren.



Figuur 5. Een visuele representatie van waarom er dit aantal gates is.

Figuur 3 is het schema van een quantum fourier transformatie. Het probleem hiermee voor ons is dat het niet laat zien hoe het het snelst werkt. Dit zou zijn als alle quantum gates zo ver mogelijk horizontaal opgeschoven zouden zijn, zoals in figuur 4. De maximale tijd die dit duurt is de bovenste lijn quantum gates plus de onderste lijn quantum gates min de ene Controlled Phase gate die ze gemeen hebben, zoals te zien in figuur 5. Totaal duurt dit dus: $(H + (n-1)CP) + ((n-1)CP + H) - CP = 2H + (2n-3)CP$ waarbij H staat voor de tijd die het duurt om een Hadamard gate uit te voeren en CP staat voor de tijd die het duurt om een Controlled Phase gate uit te voeren.

De theoretische duur van Shor's Algoritme is dus:

- Voor deel 1: 1 keer de duur van de Hadamard gate
- Voor deel 2: n^2 tot n^3 keer een van de volgende gates: NOT, CNOT en Toffoli
- Voor deel 3: $2n-3$ keer de duur van de Controlled Phase gate, plus 2 keer de duur van een Hadamard gate.

Of als we dit als formule schrijven: $3 \cdot H + n^2 \; tot \; n^3 \; (NOT, CNOT \; of \; Toffoli) + (2n-3) \cdot CP$
En als we dit opsplitsen in 2 formules, een voor de kortste tijdsduur en een voor de langste tijdsduur:
Kortste tijd = $3 \cdot H + n^2 \; (snelste \; van \; NOT, \; CNOT \; en \; Toffoli) + (2n-3) \cdot CP$
Langste tijd = $3 \cdot H + n^3 \; (langzaamste \; van \; NOT, \; CNOT \; en \; Toffoli) + (2n-3) \cdot CP$
Nu hebben we dus formules waarmee we kunnen beschrijven hoe lang het duurt om Shor's algoritme uit te voeren, uitgedrukt in aantal bits en de duur van quantum gates.

§3.3 Tijdsduur van quantum gates

§3.3.1 Tijdsduur van gate-delen

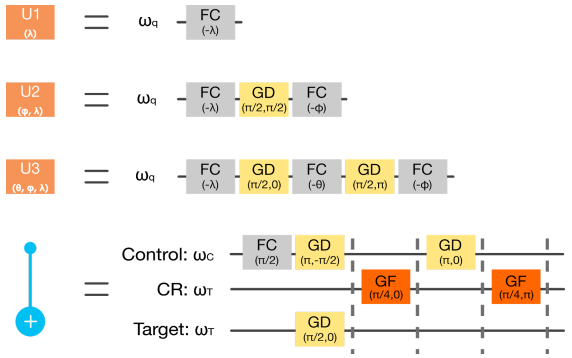
We hebben nu wel die formules, maar om ze goed te kunnen gebruiken moeten we weten hoe lang die quantum gates duren. Hiervoor hebben we de tijdsduur van de quantum gates nodig. Om hierbij te komen moeten we naar quantum gates gaan kijken op een fysiek niveau.
Hiervoor hebben we een bepaalde quantum computer nodig, omdat iedere quantum computer anders is. Hiervoor nemen wij de quantum computer “IBM Q 5 Tenerife” als voorbeeld. Deze quantum computer werkt met de positie en rotatie van de spin van elektronen en bij deze quantum computer zijn quantum gates niets meer dan bepaalde rotaties en veranderingen van de frequentie van de spin van die elektronen.
Die rotaties bestaan uit bepaalde gate-delen, en die gate-delen hebben allemaal (ongeveer) een vaste tijdsduur.

De gates bestaan uit delen genaamd FC (frame change), GD (Gaussian derivative) en GF (Gaussian flattop). FC’s veranderen de leessnelheid van de deeltjes en duren dus 0 ns, omdat ze niets met het deeltje doen, GD’s zijn aanpassingen aan een qubit en duren 60 ns en GFs zijn overdrachten tussen twee qubits. Het bij dit gate-deel moeilijker uit te vinden hoe lang ze duren. Bij deze quantum computer hebben de GF’s een kortste tijdsduur van 110 ns en een langste tijdsduur van 400 ns. Dit is variabel omdat het afhangt van welke twee qubits met elkaar werken. Laten we voor nu uitgaan van het gemiddelde van alle GF’s van deze quantum computer, namelijk 210 ns.
Totaal duren de gate-delen dus:

- FC: 0 ns
- GD: 60 ns
- GF: (gemiddeld) 210 ns

(J. Gambetta, 2018, 7 juli).

§3.3.2 opbouw van categorieën



Figuur 6. De opbouw van quantum gate categorieën (Gambetta, z.d.).

In figuur 6 is de opbouw van de drie groepen quantum gates te zien, namelijk u1, u2 en u3, net als de opbouw van een CNOT gate. u1, u2 en u3 gates zijn gates die werken op een enkele qubit, en de categorie waar gates in vallen hangt af van hoeveel eigenschappen van een qubit ze aanpassen. Voor nu is het belangrijk om te kijken hoelang al deze gates duren, want dat hebben we nodig om de tijdsduur van onze gates uit te vinden.

- u1 gates: 0 ns want een FC duurt 0 ns
- u2 gates: 60 ns want het is een GD die 60 ns duurt met twee FCs van 0 ns.
- u3 gates: 120 ns want het zijn 2 GD's van 60 ns ieder en 3 FC's van 0 ns.
- CNOT gate: 530 ns, want het zijn drie GD's van 60 ns en twee GF's van 210 ns, maar twee van de GD's lopen parallel dus die kunnen als één worden geteld.

§3.3.3 opbouw van gates en invullen tijden

Hiermee kunnen we de tijdsduur van alle gates die we nodig hebben bepalen. Laten we beginnen met de single-qubit gates. Een Hadamard gate is een u2 gate dus die duurt 60 ns en een NOT gate is een u3 gate dus die duurt 120 ns (IBM QX team, 2017). Dan de multi-qubit gates. De CNOT weten we al, die duurt 530 seconden. De Controlled Phase gate is niet heel duidelijk, maar aangezien hij ongeveer hetzelfde doet als de CNOT gate en dus waarschijnlijk ook hetzelfde in elkaar zit gaan we ervan uit dat hij even lang duurt als een CNOT gate, dus duurt deze gate ook 530 ns Van de Toffoli gate is de tijdsduur onbekend,, maar aangezien we wel weten dat hij opgebouwd kan worden uit 5 twee-qubit gates (Barenco et al., 1995) en we weten dat een CNOT gate 530 ns duurt, gaan wij ervan uit dat hij 5 keer zo lang duurt, 2650 ns dus.
Dus om het even samen te vatten:

- Tijdsduur Hadamard gate: 60 ns
- Tijdsduur NOT gate: 120 ns
- Tijdsduur CNOT gate: 530 ns
- Tijdsduur Controlled Phase gate: 530 ns
- Tijdsduur Toffoli gate: 2650 ns

We kunnen dit invullen in onze formules van eerder, namelijk:
Kortste tijd = $3 \cdot H + n^2\ (snelste\ van\ NOT,\ CNOT\ en\ Toffoli) + (2n-3) \cdot CP$ en
Langste tijd = $3 \cdot H + n^3\ (langzaamste\ van\ NOT,\ CNOT\ en\ Toffoli) + (2n-3) \cdot CP$
Na invullen en herleiden krijgen we:
Kortste tijd = $210n^2 + 1060n - 1410$ ns en
Langste tijd = $2650n^3 + 1060n - 1410$ ns, waar n het aantal input qubits is en dus ook het aantal bits van het te factoriseren getal is.

Methode

§4 Tijdsduur klassieke computer

§4.1 Hoeveel tijd heeft een klassieke Computer nodig om RSA te kraken?

De tijd die een klassieke computer nodig heeft om RSA te kraken hangt af van de lengte van n. Des te groter de n, des te meer tijd nodig is om hem te factoriseren. We gaan, door onze PCs meerdere lengtes n te laten kraken met een programma (te downloaden vanaf https://github.com/jurrejelle/RSACrack), een verband leggen tussen de lengte van n en de tijd die nodig is om die n te factorizeren.
Met dit verband kunnen we een functie inschatten waarmee we kunnen bereken hoe lang een klassieke computer in theorie nodig zal hebben voor grotere waardes van n.
We berekenen alleen de waardes voor $64 + 4k$ bits voor nu omdat RSA maar per 2 bits van de sleutel omhoog kan en we tijd willen besparen voor nu
Verder nemen we de middenwaarde van 100 pogingen tot kraken voor alle sleutels onder 150 bits, voor alle sleutels boven de 150 bits nemen we maar 20 pogingen omdat het anders per aantal bits 5 minuten $\cdot$ 100 pogingen = 500 minuten = 8 uur en 20 minuten per dataset was, en dat is niet haalbaar voor de tijd die we hebben.

§4.2 Wat gebruiken we om RSA te kraken op een klassieke computer?

We gebruiken 5 bestanden, crack.py, analyze_results.py, data.html, formula.py en generate_datafile.py. Naast dat heb je ook nog de 2 bestanden my_math.py en calc.py die in de folder imports staan, die als benodigdheden werken voor crack.py. Zonder deze bestanden werkt crack.py niet.

Crack.py

Wat crack.py doet is het probeert met de Multiple Polynomial Quadratic Sieve sleutels te kraken van verschillende lengtes. Hij maakt eerst een folder aan genaamd results om de resultaten in kwijt te kunnen, en daarna maakt hij in die folder bestanden aan voor ieder aantal bits, in de vorm $64 + 4k$ dus uiteindelijk staat in results de bestanden 64.txt, 68.txt, 72.txt etc.
Elk bestand heeft per lijn een gekraakte sleutel, in het volgende formaat:
[tijd die het duurde om hem te kraken], [priemgetal 1], [privesleutel n]

Voor alle aantal bits onder 130 genereert hij 100 testcases, en voor elk aantal bits erboven maar 20, omdat het anders te lang zou duren.

Generate_datafile.py

Dit programma genereert, nadat crack.py wat data heeft gegenereerd, een javascript bestand, namelijk datafile.js, in het mapje results aan de hand van alle files in de map results, en dat bestand wordt gebruikt in data.html om de grafiek goed weer te kunnen geven.

Aproximate_formula.py

Dit bestand creëert een formule van de dataset gegenereerd door generate_datafile.py met als eerste punt het grootste aantal bits en als tweede punt dat aantal bits min 40. Dat genereert een formule, die je kan zien als je het programma uitvoert, en een dataset/javascript bestand genaamd estimated_datafile.js, met daarin de waardes voor de tijd in seconden van de bits van 64 tot en met 256. Die wordt later in Data.html ook weergegeven.

Data.html

Wanneer je generate_datafile.py hebt uitgevoerd, kan je dit bestand openen in je browser (bijvoorbeeld chrome) om de data uit de results map weer te geven in een grafiek, zodat het overzichtelijk is. Verder wordt de dataset die voorspeld is uit de huidige dataset ook weergegeven, om te laten zien hoe de echte test data en de grafiek overeenkomen.

Analyze_results.py

Dit programma is niet nodig om de formule te genereren maar is een programma wat wordt gebruikt om meer info te krijgen over een bepaalde dataset/file. Wanneer je het opstart krijg je een lijst met alle files in results, en daardoor dus alle datasets, bijvoorbeeld 64.txt, 68.txt etcetera. Dan kun je de naam van één van de files invoeren, bijvoorbeeld 128.txt, en dan geeft het het aantal testcases, het gemiddelde van de tijden, de middenwaarde van de tijden en het kortste en langste dat een sleutel heeft geduurd om gekraakt te worden in die dataset.

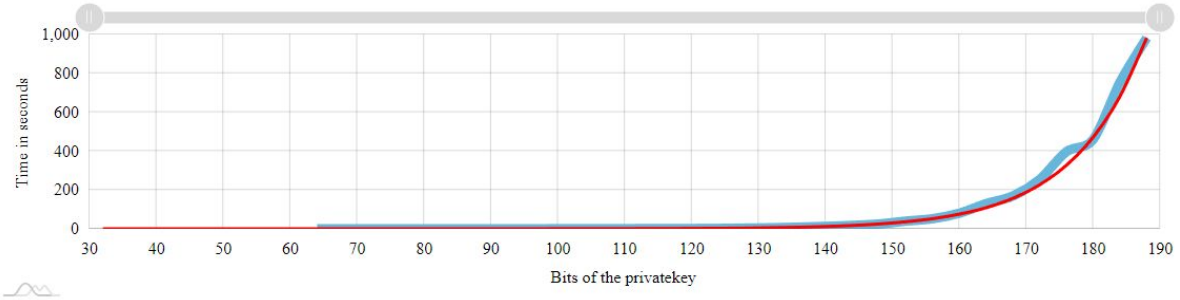
Een gedetailleerde beschrijving van hoe je de files moet gebruiken, samen met de files zelf, is beschikbaar op https://github.com/jurrejelle/RSACrack

Resultaten

§5 Resultaten klassieke computer

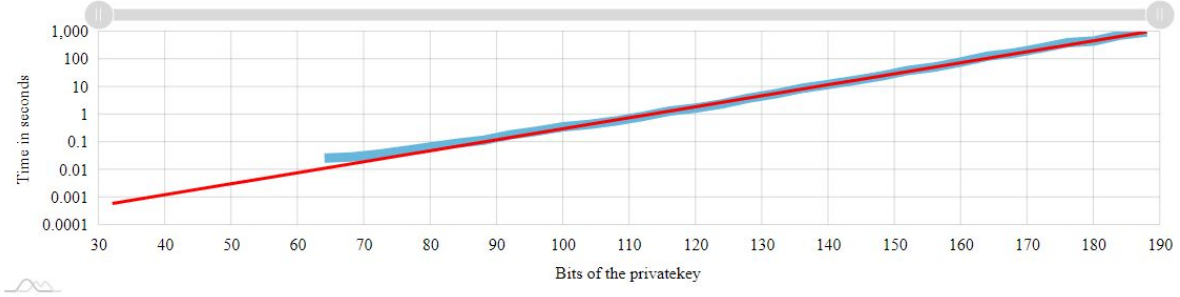
§5.1 Praktische resultaten klassieke computer

De middenwaardes voor alle $64 + 4k$ bits tot 164 maken de volgende grafiek (zie figuur 7)
De blauwe lijn is de data die we zelf gekregen hebben, de rode lijn is de data die we interpoleren en extrapoleren met behulp van de formule die we maken.



Figuur 7. De data geplot op een normale schaalverdeling bron: data.html

Waaruit we aan kunnen nemen dat de formule voor tijd tegen het aantal bits, inderdaad exponentieel is.
Om dit te verifiëren hebben we de data geplot op logaritmisch papier (zie figuur 8)



Figuur 8. De data geplot op een logaritmische schaalverdeling bron: data.html

Hieruit kunnen we niet alleen zien dat de formule inderdaad exponentieel is, maar hieruit kunnen we ook de formule zelf afleiden.

§5.2 Opstellen formule klassieke computer

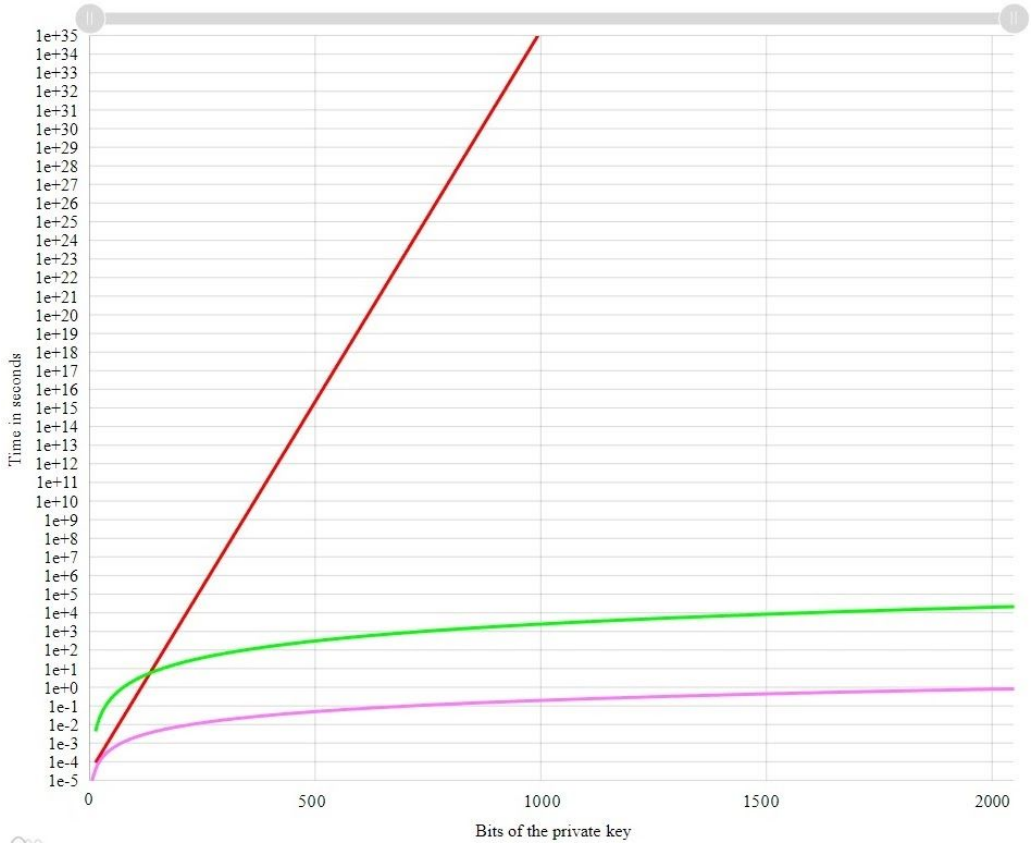
Om de formule van een exponentieel verband op te stellen, nemen we twee punten op de grafiek (in het geval van ons programma de laatste bit, en de bit 40 bits voor de laatste bit), om daarvandaan het grondgetal $grondgetal$ uit te rekenen, en dan pluggen we één van de twee punten terug in de formule om de $basis$ uit te rekenen, waardoor de uiteindelijke formule in de vorm $t = basis \cdot grondgetal^n$ ook in het bestand data.html staat, en in ons geval is dat:

$t = 3.20387527935 \cdot 10^{-5} \cdot 1.09603071613^n$

Met t als de tijd in seconden, en n als aantal bits

Conclusie

Nu we dus die drie formules hebben, kunnen we ze gaan vergelijken.



Figuur 9. de formules afgebeeld op een logaritmische schaalverdeling. Bron: data2.html, Jurre Groenendijk, 2019

De rode lijn is de tijd voor klassieke computers, de groene lijn is de langzaamste tijd voor quantum computers en de roze lijn is de optimale tijd voor quantum computers

De formules zijn:

- Voor een normale computer: $t = 3.20387527935 \cdot 10^{-5} \cdot 1.09603071613^{n}$
- Voor een theoretische trage quantum computer: $t = (2650n^3 + 1060n - 1410) \cdot 10^{-9}$
- Voor een theoretische snelle quantum computer: $t = (210n^2 + 1060n - 1410) \cdot 10^{-9}$

Uit figuur 9 kunnen wij opmaken dat een quantum computer sneller is, of langzamer is tot de 132 bits, afhangend van welke formule je kiest. Dit maakt echter niet veel uit want met 132 bits duurt het maar zes seconden. Daarna is de quantum computer echter veel sneller. Op 500 bits is de quantum computer al 10^{13} of zelfs 10^{17} keer zo snel. Dus om onze hoofdvraag te beantwoorden: Een quantum computer is tot de 132 bits mogelijk langzamer dan een normale computer, maar aangezien dat tot maximaal 5 seconden is maakt dat ook geen significant verschil. Daarna wordt een quantum computer de enige realistische manier om het te kraken, aangezien je met een klassieke computer al snel even lang nodig hebt als het universum tot nu toe bestaat.

Discussie

We hadden graag willen kijken naar hoe lang een normale computer in theorie erover zou doen om RSA te kraken en hier, net als bij onze praktische opdracht, een formule van willen maken. Dit lukt alleen niet omdat de formules die nodig zijn om sleutels, groter dan dat we met onze praktische opdracht hebben gekraakt, te kraken, vele malen ingewikkelder zijn en omdat dit veel moeilijker is dan met quantum computers door de complexiteit van klassieke computers. Hierdoor kunnen wij geen theoretisch verband leggen tussen de grootte van het getal en hoe lang een computer er over doet om te kraken, maar alleen een praktisch verband. We wilden ook graag kijken of onze formule voor quantum computers klopt door het in te voeren in een online quantum computer en te kijken naar hoe lang het zou duren, maar helaas kan je bij alle online quantum computers niet zien hoelang het experiment duurt, en hebben we geen quantum computer tot onze beschikking.

Zoals we eerder zeiden kan een quantum computer fouten maken, alleen is dat hier geen groot probleem aangezien we het gewoon opnieuw kunnen proberen. Ook moeten we zeggen dat Shor's algoritme niet altijd werkt, dus daarom is je antwoord soms ook niet goed en moet je het soms ook opnieuw proberen. Maar het is zeer waarschijnlijk dat na 10 keer proberen de quantum computer wel het goede antwoord geeft, wat wij gelukkig heel simpel kunnen checken door n te delen door q of p.

Het quantum algoritme voor het toepassen van de functie $f(x) = a^x \, mod \, n$ kan ook sneller gemaakt worden door gebruik te maken van quantum fourier transformaties, maar dat kan alleen als je algoritmes hebt met meer dan 600 qubits. Ook is het handig om te weten dat de formule is voor het aantal bits van het te kraken getal, niet het aantal qubits van de quantum computer. De quantum computer heeft twee keer zoveel qubits nodig omdat hij dit getal moet kunnen opslaan in de input en output registers (Markov & Saeedi, 2015).

Slotwoord

In dit slotwoord willen wij vertellen waarom wij dit onderwerp hebben gekozen, een voorspelling laten zien over de toekomst van RSA en willen wij wat mensen bedanken, omdat dit profielwerkstuk zonder hen niet mogelijk zou zijn.

Wij hebben dit onderwerp gekozen omdat Jelle en Jurre al een cursus Quantum Computing hebben gedaan (deze was aangeboden door school) en zij daarom al wat voorkennis hadden. Ook was Jurre al best bekend met RSA, en wilde hij wat gaan doen met RSA en de rest vond het ook wel een interessant onderwerp.

Als we niets doen aan de huidige methode van RSA en onze gegevens op blijven slaan met een RSA sleutel van bijvoorbeeld 4096 bits, dan zou het in het slechtste geval volgens Rose's law nog drie jaar duren voordat ze dat (in minder dan twee dagen zelfs) kunnen kraken. Rose's law zegt dat quantum computers ieder jaar twee keer zo veel qubits hebben, en als we kijken naar Dwave Systems dan hebben ze nu al 2000 qubits, dus laat het einde van encrypties op basis van factorisatie niet lang meer op zich wachten (Griffin, 2017).

Verder willen we specifiek bedanken:
Floor Terra (@flooter op twitter) voor het beantwoorden van onze vragen over Quantummechanica,
Feodore, Nicky Troost, Melanie Moerkerken, Floor Terra, Sylvia de Raat, Jilles Groenendijk, @Dave_von_S en @PyroBatNL voor het proeflezen
En meneer Kustermans voor het begeleiden, het vinden van fouten, het geven van vrijheid en het helpen in het algemeen.

Bronvermelding

Barenco, A., Bennett, C. H., Cleve, R., DiVincenzo, D. P., Margolus, N., Shor, P., . . . Weinfurter, H. (1995). Elementary gates for quantum computation. Geraadpleegd van https://arxiv.org/pdf/quant-ph/9503016.pdf

Bender2k14. (2014, 29 juli). Quantum subroutine for order finding in Shor's algorithm [Foto]. Geraadpleegd op 28 januari 2019, van https://upload.wikimedia.org/wikipedia/commons/6/6b/Shor%27s_algorithm.svg

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2011). Introduction to Algorithms (17e ed.). Cambridge, Verenigde Staten: MIT Press.

Hayward, J. (z.d.). Quantum Parallelism. Geraadpleegd op 16 oktober 2018, van https://quantum-algorithms.herokuapp.com/299/paper/node16.html

Gambetta, J. (z.d.). [Opbouw Quantum Gates] [Foto]. Geraadpleegd op 28 januari 2019, van https://raw.githubusercontent.com/Qiskit/ibmq-device-information/master/backends/tenerife/images/gatedef_U1U2U3_CNOT.png

Gambetta, J. (2018, 7 juli). IMBQ device information. Geraadpleegd op 29 januari 2019, van https://github.com/Qiskit/ibmq-device-information/blob/master/backends/tenerife/V1/version_log.md

Gidney, C. (2017). Shor's Quantum Factoring Algorithm. Geraadpleegd op 16 oktober 2018, van http://algassert.com/post/1718

Griffin, M. (2017, 19 december). Rose's Law is Moore's Law on steroids. Geraadpleegd op 1 februari 2019, van https://www.fanaticalfuturist.com/2016/08/quantum-computing-roses-law-is-moores-law-on-steroids/

IBM QX team. (2017). Advanced Single-Qubit Gates. Geraadpleegd op 31 januari 2019, van https://quantumexperience.ng.bluemix.net/proxy/tutorial/full-user-guide/002-The_Weird_and_Wonderful_World_of_the_Qubit/004-advanced_qubit_gates.html

Introduction to the D-Wave Quantum Hardware | D-Wave Systems. (z.d.). Geraadpleegd op 17 oktober 2018, van https://www.dwavesys.com/tutorials/background-reading-series/introduction-d-wave-quantum-hardware

Kelly, J. (2018, 5 maart). A Preview of Bristlecone, Google's New Quantum Processor. Geraadpleegd op 18 oktober 2018, van https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html

Markov, I., & Saeedi, M. (2015). Constant-Optimized Quantum Circuits for Modular Multiplication and Exponentiation. Geraadpleegd van https://arxiv.org/pdf/1202.6614.pdf.

Nielsen, M. A., & Chuang, I. L. (2010). Quantum Computation and Quantum Information: 10th Anniversary Edition (20e ed.). Cambridge, Engeland: Cambridge University Press.

Rivest, R., Shamir, A., & Adleman, L. (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Geraadpleegd van https://people.csail.mit.edu/rivest/Rsapaper.pdf

Shor's factoring algorithm. (2015, 26 oktober). Geraadpleegd op 29 januari 2019, van https://www.quantiki.org/wiki/shors-factoring-algorithm

Shor, W. (1996). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. Geraadpleegd van https://people.csail.mit.edu/rivest/Rsaphttps://arxiv.org/pdf/quant-ph/9508027.pdf

Trenar3. (z.d.). Quantum circuit for Quantum-Fourier-Transform with n Qubits [Foto]. Geraadpleegd op 28 januari 2019, van https://upload.wikimedia.org/wikipedia/commons/6/61/Q_fourier_nqubits.png

Bijlages

(Alle bijlages kunnen gedownload worden vanaf https://github.com/jurrejelle/RSACrack/)