SHOR'S ALGORITHM

FAKTORISIERUNG GROßER ZAHLEN MIT EINEM QUANTENCOMPUTER

Ausarbeitung eines Vortrags
im Rahmen des Seminars **Algorithmen für Quantencomputer**
Leitung: Prof. Helmut Alt
SoSe 2002
FU-Berlin

von Till Zoppke & Christian Paul

INHALT:

DAS RSA-VERFAHREN

Das RSA-Verfahren ist das derzeit gängige Verfahren zur geheimen Nachrichtenübermittlung. Wir stellen es im Zusammenhang mit Shors Algorithmus vor, da es die „Beute" darstellt, welche mit einem polynominellen Faktorisierungsalgorithmus „erlegt" werden kann.

Der Name *RSA* leitet sich von seinen Erfindern Rivest, Shamir und Adleman ab, welche das Verfahren 1978 vorstellten. Es beruht darauf, dass ein Primzahltest (zur Generierung eines Schlüssels) auch im Bereich von mehr als 100 Dezimalstellen in Sekunden durchführbar ist (vgl. Bundschuh 1998, S.100), hingegen eine Faktorisierung solcher Zahlen 10^9 Jahre benötigt.

Bei dem Verfahren liegt ein Teil des Schlüsselsatzes offen. Möchte Alice eine Nachricht an Bob verschicken, so nutzt sie dazu seinen öffentlichen Schlüssel (engl. *public key*), den man sich in einer Art Telefonbuch ausgestellt denken kann. Bob entschlüsselt die Nachricht mit seinem privaten Schlüssel (*private key*), welchen nur Bob kennt.

An dieser Stelle einige Erkenntnisse der Zahlentheorie, auf denen RSA beruht.

Definition:

$Z_n^* := \{[a] \subset Z \mid \forall_{x \in [a]}(x \bmod n = a \bmod n \wedge ggT(a,n) = 1)\}$ heißt Menge der primen Restklassen modulo *n*. Repräsentanten sind $\{a < n \mid ggT(a,n) = 1\}$

Definition:

$\varphi(n) := |Z_n^*|$ (Anzahl der primen Restklassen) heißt Eulersche-phi-Funktion. Diese hat folgende zwei wichtige Eigenschaften:

1. $\varphi(n \cdot m) = \varphi(n) \cdot \varphi(m)$ (Multiplikativität)
2. $\varphi(p) = p - 1$ für eine Primzahl *p*.

Satz:

$ggT(a,b) = 1 \Leftrightarrow aZ + bZ = Z$.

Zur Erzeugung eines Schlüssels (*e*, *x*, *m*) wähle man nun zwei Primzahlen *p*, *q*, $p \neq q$, sei $m := p \cdot q$. Dann ist $\varphi(m) = (p-1) \cdot (q-1)$.
Wähle *e* mit $ggT(e, \varphi(m)) = 1$.
Dann $\exists_{x,y \in Z}(x \cdot e = y \cdot \varphi(m) + 1 \Rightarrow x \cdot e \equiv 1 \bmod \varphi(m))$,
und es gilt insgesamt: $\boxed{\forall_{a \in N} \; a^{e \cdot x} \equiv a \bmod m}$

Beweis: Fallunterscheidung.

1. $ggT(a,m) = 1$: Dann gilt $a^{ex} = a^{y \cdot \varphi(m)+1} = \left(a^{\varphi(m)}\right)^y \cdot a \equiv a \bmod m$ wegen $a^{\varphi(m)} \equiv 1 \bmod m \; \forall a,m$ mit $ggT(a,m) = 1$.
2. $ggT(a,m) \neq 1$: $\varphi(m) = (p-1) \cdot (q-1), e \cdot x \equiv 1 \bmod \varphi(m)$
   $\Rightarrow e \cdot x \equiv 1 \bmod (p-1), e \cdot x \equiv 1 \bmod (q-1)$

$\Rightarrow \exists_{l,m\in N}\, e\cdot x = l\cdot(p-1)+1 = m\cdot(q-1)+1$

o. E. $a \neq 0 \mod p$ und $a \neq 0 \mod q$. Dann gilt:

$$\begin{aligned}
&\left[a^{ex}\right]_p = \left[a^{l\cdot(p-1)+1}\right]_p \\
&= [a]_p^{\,l\cdot(p-1)} \cdot [a]_p \\
&= \left[a^p\right]_p^{\,l} \cdot [a]_p^{\,-l} \cdot [a]_p \\
&= [a]_p^{\,l} \cdot [a]_p^{\,-l} \cdot [a]_p \\
&\left(a^p \equiv a \mod p, \text{ denn p ist Primzahl nach dem kleinen Satz von Fermat}\right) \\
&= [a]_p
\end{aligned}$$

Das entsprechende zeigt man für *q*.
Daraus folgt nun (beachte: *p* und *q* sind teilerfremd), dass

$$a^{e\cdot x} \equiv a \mod m = p\cdot q$$

Zur Verschlüsselung einer Nachricht *a*, die wir uns als Bitfolge mit fester Länge vorstellen können, potenziert man *a* mit *e* mod *m* (es gelte a<m): $b = a^e \mod m$. Hierbei bildet (*e*, *m*) den öffentlichen Schlüssel. Zur Entschlüsselung der Nachricht *b* potenziert der Empfänger *b* mit *x* mod *m*, und es gilt: $b^x \mod m = a$, man erhält also wieder die unverschlüsselte Nachricht. Geheimer Schlüssel ist *x*, auch *p*, *q* und *y* müssen geheim bleiben.

Beispiel:
Wähle *p*=3, *q*=5, dann ist *m*=15, $\varphi(m) = 2\cdot 4 = 8$.
Wähle *e*=7 $(ggT(7,8)=1)$
Gesucht *x*, *y* mit $x\cdot 7 = y\cdot 8+1$

$\Rightarrow x = 7, y = 6$ ist eine Lösung

Verschlüsselung von 2, 11:

$2^7 = 128 \equiv 8 \mod 15$

$11^7 = 19.487.171 \equiv 11 \mod 15$

Entschlüsselung:

$8^7 = 2.097.152 \equiv 2 \mod 15$

$11^7 = 19.487.171 \equiv 11 \mod 15$

Wie kann das Verfahren geknackt werden? Es beruht darauf, dass keine effiziente Umkehrung der Eulerschen-phi-Funktion existiert. Sollte jedoch m faktorisiert werden können, so lässt sich auch leicht $\varphi(m)$ finden:

$$m = p\cdot q \Rightarrow \varphi(m) = (p-1)\cdot(q-1)$$

(nach Konstruktion zerfällt *m* in genau 2 Primfaktoren). Nun lassen sich auch einfach drei Zahlen *x*, *y*, *e* mit $x\cdot e = y\cdot\varphi(m)+1$ bestimmen (eine beliebige erfüllende Belegung genügt), und der Kode ist erfolgreich geknackt. In diesem Zusammenhang ist es nachvollziehbar, dass die Veröffentlichung von Shors Algorithmus ein über die Akademische Welt hinaus breites Echo fand und einen Motivationsschub zur Entwicklung von Quantencomputern gab.

ZAHLENTHEORETISCHER HINTERGRUND

Der Algorithmus von Shor ist eigentlich eine für Quantencomputer angepasste Version eines Algorithmus von Rabbin (1976), bei der die Faktorisierung einer Zahl auf die Bestimmung der Ordnung in Z_n zurückgeführt wird.

Sei $n = p \cdot q$, Produkt zweier Primzahlen p und q.
Die Methode sieht nun vor, dass wir eine Zahl x (aber $x \neq \pm 1$) finden, so dass:

x^2	$\equiv 1$	(mod n)	und damit:
$x^2 - 1$	$\equiv 0$	(mod n)	und nach der 3. Binomischen Formel:
$(x-1)(x+1)$	$\equiv 0$	(mod n)	
→ n	$\mid (x-1)(x+1)$		
→ $p \cdot q$	$\mid (x-1)(x+1)$		

Da n ein Teiler von (x-1)(x+1) ist, ist ein Faktor von n auch ein Teiler von (x-1) oder (x+1). Diesen erhalten wir durch die Berechung von ggT (n, x-1) und ggT(n, x+1) und erhalten damit unseren gesuchten Primfaktor von n.

Wie finden wir nun solch ein x ?

Dies geschieht mit Hilfe der Periodenbestimmung der Funktion $f_{a,n}(k) = a^k \bmod n$.
Die Periode r von f(a) ist die kleinste Zahl, für die gilt:

$f_{a,n}(k+r)$	=	$f_{a,n}(k)$	für alle k	also:
$f_{a,n}(k+r)$	=	a^{k+r}	(mod n)	
	=	$a^k \cdot a^r$	(mod n)	
	=	a^k	(mod n)	das ist genau, wenn:

a^r	$\equiv$	1	(mod n)

Die Zahl r wird auch die Ordnung von a in Z^*_n genannt.
Wir wählen zufällig ein a aus und bestimmen die Periode. Danach setzen wir für unser gesuchtes x ein:

x	=	$a^{r/2}$	und damit ist:
x^2	=	$a^r \equiv 1$	(mod n)

Es versteht sich, das dies nicht funktioniert, wenn r ungerade oder $a^{r/2} \equiv +/-1$ ist. Ist dies der Fall, wählen wir ein anders a und bestimmen erneut die Periode. Die Wahrscheinlichkeit, dass wir nicht erneut testen müssen, liegt ungefähr bei 50%.

Mit Hilfe des Euklidischen Algorithmus können wir nun unseren Primfaktor bestimmen:

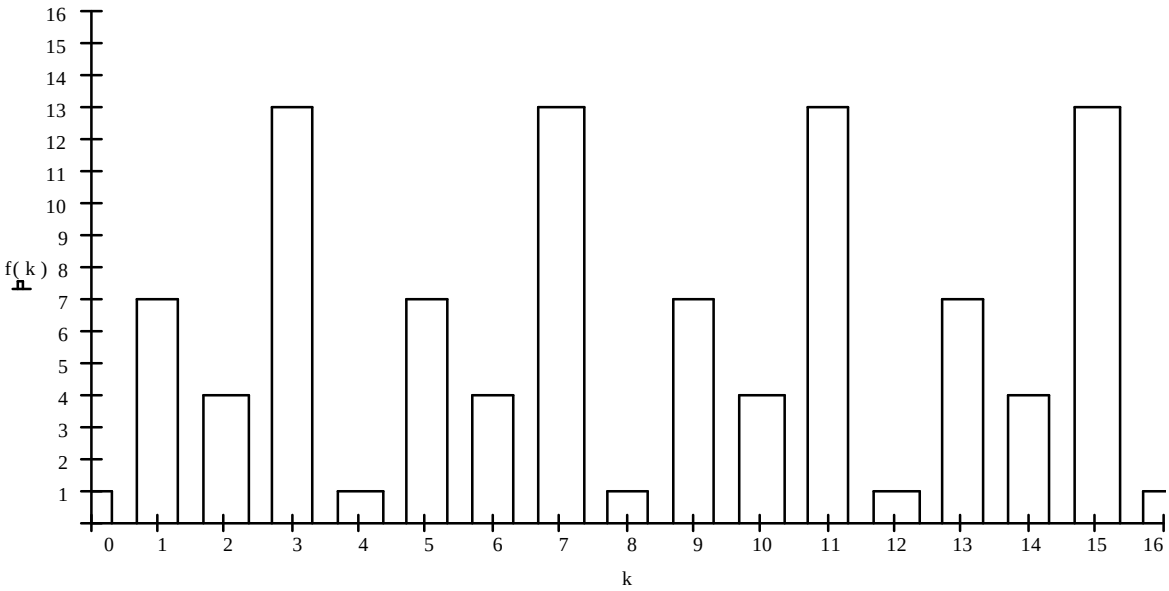
$$ggT(n, a^{r/2}+1) \text{ und } ggT(n, a^{r/2}-1)$$

Wir haben das Problem der Suche nach einem Primfaktor also auf die Bestimmung der Periode einer Funktion reduziert. Der Algorithmus von Shor nutzt die Rechenleistung des

Quantencomputers bei der parallelen Berechnung aller Funktionswerte von a^k mod n und der anschließenden Periodenberechnung mit Hilfe einer angepassten Fourier-Transformation.

Beispiel:

Der Funktionsgraph von a^k mod n mit den Werten **n = 15** und **a = 7**:



Auf der X-Achse befinden sich die steigenden Werte für k und auf der Y-Achse die entsprechenden Funktionswerte von **$f(k) = 7^k \bmod 15$**

Deutlich erkennbar ist die Periode der Funktion.

Periode: r = 4

→	$a^{r/2}$	=	$7^{4/2} = 7^2 = 49$
→	ggT (n, $a^{r/2}$+1)	=	ggT (15, 48) = **3**
→	ggT (n, $a^{r/2}$-1)	=	ggT (15, 50) = **5**

Primfaktorzerlegung: 15 = 5·3

DER ALGORITHMUS AUF DEM QUANTENCOMPUTER

Unser Quantencomputer soll uns die Periode r der Funktion a^k mod n berechnen. Das a haben wir zufällig ausgewählt und steht als Eingabe bereit. Wie wir sehen werden, erhalten wir nicht r direkt am Ende der Messung, sondern einen Wert, den wir noch mit klassischen Mitteln nachbearbeiten müssen.

Wir benötigen zwei Quantenregister R1 und R2 und mit der Länge $\overline{\log_2 n}$ Qubits. Dazu kommt noch etwas "Arbeitsspeicher", welcher aber nach jedem Rechenschritt zurückgesetzt wird, er wird in der folgenden Beschreibung nicht weiter auftauchen.

Sei weiterhin q eine 2er Potenz mit $n^2 \le q \le 2n^2$, q 0 $O(n^2)$

Wir teilen den Ablauf den Algorithmus in fünf Schritte ein:

0. Ausgangszustand

$$\psi 0 \quad = \quad |0\rangle|0\rangle$$

Wir wählen als Beispiel erneut die Werte wie im Beispiel vorhin: n = 15, a = 7, q = 16

1. Superposition

Auf das erste Register wird eine Hadamard-Tranformation H_q angewendet, um die Superposition aller k Zahlen zu erreichen.

$$\psi 1 \quad = \quad \frac{1}{\sqrt{q}} \sum_{k=0}^{q-1} |k\rangle|0\rangle$$

Beispiel:
$$\psi 1 = \frac{1}{\sqrt{16}} \sum_{k=0}^{15} |k,0\rangle = \frac{1}{4} \begin{pmatrix} |0\rangle|0\rangle + |1\rangle|0\rangle + |2\rangle|0\rangle + |3\rangle|0\rangle + \\ |4\rangle|0\rangle + |5\rangle|0\rangle + |6\rangle|0\rangle + |7\rangle|0\rangle + \\ |8\rangle|0\rangle + |9\rangle|0\rangle + |10\rangle|0\rangle + |11\rangle|0\rangle + \\ |12\rangle|0\rangle + |13\rangle|0\rangle + |14\rangle|0\rangle + |15\rangle|0\rangle \end{pmatrix}$$

2. Berechnung von a^k mod n

Nun wird a^k mod n für alle k im ersten Register gleichzeitig berechnet und in das zweite Register geschrieben.

$$\psi 2 \quad = \quad \frac{1}{\sqrt{q}} \sum_{k=0}^{q-1} \left| k \right\rangle \left| a^k \bmod n \right\rangle$$

Nach Shor kann dies auf einem Quantencomputer in $O(\lg^2 n \lg\lg n \lg\lg\lg n)$ Zeit realisiert werden, er selbst schlägt die Realisierung der Exponentiation mit Hilfe des schnellen Algorithmus von Schönhage-Strassen vor.

Beispiel:
$$\psi 2 = \frac{1}{\sqrt{16}} \sum_{k=0}^{15} \left| k, 7^k \bmod 15 \right\rangle = \frac{1}{4} \begin{pmatrix} |0\rangle|1\rangle + |1\rangle|7\rangle + |2\rangle|4\rangle + |3\rangle|13\rangle + \\ |4\rangle|1\rangle + |5\rangle|7\rangle + |6\rangle|4\rangle + |7\rangle|13\rangle + \\ |8\rangle|1\rangle + |9\rangle|7\rangle + |10\rangle|4\rangle + |11\rangle|13\rangle + \\ |12\rangle|1\rangle + |13\rangle|7\rangle + |14\rangle|4\rangle + |15\rangle|13\rangle \end{pmatrix}$$

3. Implizite Messung des zweiten Registers

Als nächstes wird eine implizite Messung des zweiten Register durchgeführt. Nennen wir den Messwert y. Je nach a und n gibt es unterschiedliche Werte für y, jedoch alle mit der gleichen Wahrscheinlichkeit.

Zu jedem y existiert eine bestimmte Sequenz von k_j-Werten: k_0, k_2, ... k_m aus dem ersten Register, so das für jedes k_j gilt:

$$a^{k_i} \equiv y \pmod{n}$$
$$a^{k_0 + rj} \equiv y \pmod{n}$$

$k_j = k_0 + r \cdot j$ wobei r unsere gesuchte Periode ist und
k_0 der kleinste Wert aus dem ersten Register, der Offset.
k_m ist der größte k_j Wert, für diesen gilt $k_m \leq q$
und damit ist $m \approx q/r$

Da das zweite Register nun fest ist, wollen wir es im folgenden vernachlässigen. Unser neuer Zustand ist demnach:

$$\psi 3 \quad = \quad \frac{1}{\sqrt{m+1}} \sum_{j=0}^{m} \left| k_0 + rj \right\rangle$$

Beispiel: Nehmen wir an, es würde die "7" gemessen

$$\psi 3 = \frac{1}{\sqrt{m+1}} \sum_{j=0}^{m} \left| k_0 + rj \right\rangle = \frac{1}{2} \left(|1\rangle + |5\rangle + |9\rangle + |13\rangle \right)$$

4. Quanten Fourier-Transformation

Nun wird die Quanten Fourier-Transformation (QFT) auf das erste Register angewendet: Die QFT ist eine angepasste Variante der Schnellen Fourier-Transformation (FFT). Auf Quantencomputer in $O(q^2)$ realisierbar, FFT auf herkömmlichen Computern in $O(2^q q)$ Zeit.

Die QFT ist eine unitäre Transformation.

$$QFT_q : |a\rangle \rightarrow \frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} e^{2\pi i a c / q} |c\rangle$$

mit der Matrix:

$$F_q = \frac{1}{\sqrt{q}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{(q-1)} \\ 1 & \omega^2 & (\omega^2)^2 & \dots & (\omega^2)^{(q-1)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \omega^{(q-1)} & (\omega^2)^2 & \dots & (\omega^{(q-1)})^{(q-1)} \end{bmatrix}$$

wobei $\omega = e^{2\pi i / q}$ die q-te Einheitswurzel im Komplexen ist.

In unserem Fall bewirkt die QFT, das wir eine periodische Funktion (Periode r) mit Offset (k_0) auf eine periodische Funktion mit der Periode q/r abbilden, ohne einen störenden Offset.

Wir betrachten im folgenden nur den Spezialfall m = q/r -1

$$\psi 3 \quad = \quad \sqrt{\frac{r}{q}} \sum_{j=0}^{\frac{q}{r}-1} |k_0 + rj\rangle$$

Nach Anwendung von QFT sind wir im folgenden Zustand:

$$\begin{aligned} \psi 4 \quad &= \quad \frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} \sqrt{\frac{r}{q}} \sum_{j=0}^{\frac{q}{r}-1} e^{2\pi \cdot i \cdot c \cdot (k_0 + rj)/q} |c\rangle \\ &= \quad \frac{\sqrt{r}}{q} \sum_{c=0}^{q-1} e^{2\pi \cdot i \cdot c \cdot k_0 / q} \left(\sum_{j=0}^{\frac{q}{r}-1} e^{2\pi \cdot i \cdot c \cdot rj / q} \right) c\rangle \\ &= \quad \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} e^{2\pi \cdot i \cdot c \cdot k_0 / q} \left| j \cdot \frac{q}{r} \right\rangle \end{aligned}$$

$$\psi 4 \quad = \quad \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} e^{2\pi \cdot i \cdot c \cdot k_0 / q} \left| j \cdot \frac{q}{r} \right\rangle$$

Beispiel: $$= \frac{1}{\sqrt{4}} \left(\left| 0 \right\rangle + e^{2\pi \cdot i / 4} \left| 4 \right\rangle + e^{2\pi \cdot i \cdot 2 / 4} \left| 8 \right\rangle + e^{2\pi \cdot i \cdot 3 / 4} \left| 12 \right\rangle \right)$$

5. Extraktion der Periode

Nun sind wir soweit fertig, dass wir messen können. Wenn wir nun das erste Register messen, erhalten wir ein Vielfaches von q/r. Sei c der erhaltene Messwert:

$$c = j \cdot \frac{q}{r}$$
$$\rightarrow \frac{c}{q} = \frac{j}{r}$$

wenn ggT (j, r) = 1 (also teilerfremd), dann kann r korrekt bestimmt werden durch:

$$r = \frac{q}{ggT(c, q)}$$

ansonsten erhalten wir eine falsche Periode. Dies zu überprüfen ist allerdings nicht weiter schwer, und wir müssen eventuell dann einen neuen Rechen- und Messvorgang starten.

Beispiel:

Angenommen, wir erhalten c = 8 (j = 2) als Messwert:

ggT (2, 4) = 2 → r = 16 / ggT (8, 16)
→ r = 2
→ Neue Rechnung

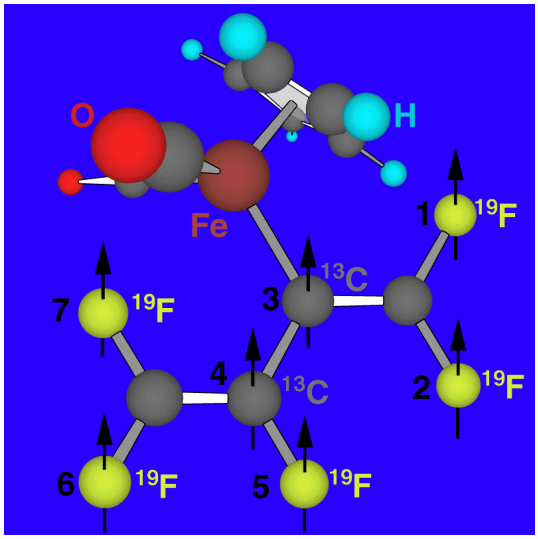
Angenommen, wir erhalten c =12 (j = 3) als Messwert:

ggT (3, 4) = 1 → r = 16 / ggT (12, 16)
r = 16 / 4
r = 4 (Hurra!)

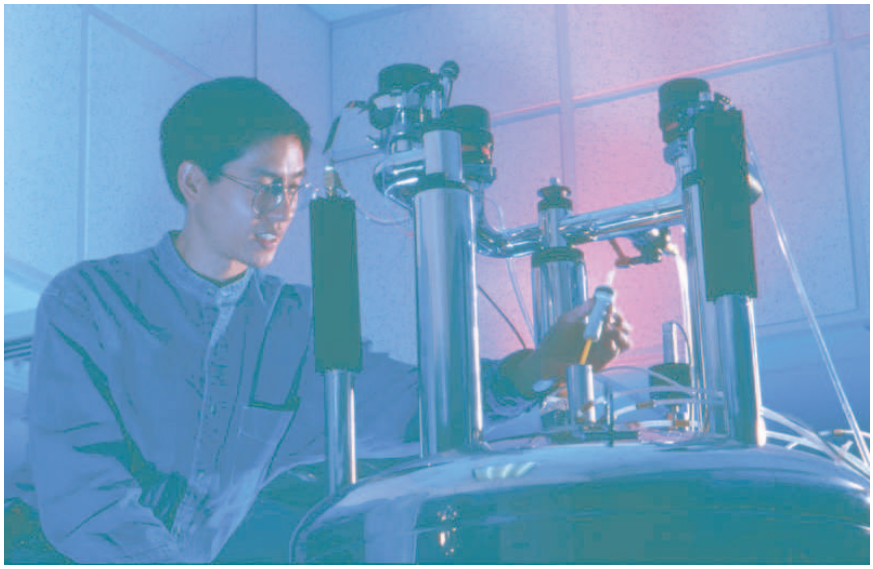
Wir haben aber einen Spezialfall betrachtet! In der Regel können wir nicht annehmen dass Spezialfall m = q/r -1. In Normalfall wird eine weitere (manuelle) Nachbearbeitung mittels Kettenbruchentwicklung nötig sein.

PERSPEKTIVEN

Mit der Veröffentlichung von Shors Algorithmus im Jahre 1994 ließ die praktische Umsetzung in Hardware nur sieben Jahre auf sich warten. Im Jahre 2001 gelang es Wissenschaftlern im IBM Forschungszentrum in Almaden einen Quantencomputer zu bauen, welcher die Zahl 15 faktorisiert. Hierzu wurde eigens ein Molekül entworfen, welches sieben Qbits beinhaltet. Deren Spin ist in der Abbildung durch schwarze Pfeile gekennzeichnet. Die Lösung ist leider nicht auf größere Zahlen skalierbar, denn die Länge der quantenfaktorisierbaren Zahl richtet sich nach der Anzahl der Qbits pro Quantenregister. Flaschenhals ist hierbei die modulare Exponentiation, welche für ein zu faktorisierendes m eine Zahl von m^3 Quantenregistern benötigt.



Überschlägt man die Rechenleistung herkömmlicher Computer nach Moore's law, so ist bei gelegentlicher Verdoppelung der Schlüssellänge die Sicherheit des RSA-Verfahrens auch in ferner Zukunft gewährleistet. Das Verhältnis zwischen dem Rechenaufwand zur Erzeugung der Schlüssel (Primzahltest) und dem Rechenaufwand zur Verschlüsselung/ Entschlüsselung (modulare Exponentiation) verändert sich mit steigender Schlüssellänge noch zu Ungunsten der Faktorisierung. Im Jahre 2042 würde die Faktorisierung einer Zahl mit 4096 bit nach herkömmlicher Methode 2×10^{22} Jahre benötigen. Ganz anders die Situation bei Entwicklung eines Quantencomputer mit genügender Zahl von Qbits und einer hypothetischen Rechenleistung von 100Mhz. Für eine Schlüssellänge von 4096 würde dieser aus mindestens 2×10^{12} Qbits bestehen und könnte die Faktorisierung in 4,8 Stunden durchführen (average case). Beide Hochrechnungen nach Gruska 1999.



QUELLEN- UND LITERATURANGABEN

Shor, Peter W., 1997: *Polynomial-Time Algorithms for Prime Factorisation and Discrete Logarithms on a Quantum Computer* (erweiterte Version der Erstschrift von 1994). http://www.research.att.com/~shor/papers/QCjournal.pdf

Hirvensalo, Mika, 2001: *Quantum Computing*

Gruska, Jozef, 1999: *Quantum Computing*

Peter Bundschuh, ^{4}1998: *Einführung in die Zahlentheorie*

Jürgen Müller, 1999: *Skript zur Vorlesung Lineare Algebra II, Universität Trier*

HYPERLINKS

- Konzeption und Realisation eines Programms zur Simulation des Algorithmus von Shor. Studienarbeit von Thorsten Koch aus dem Jahr 2001. http://www.theoinf.tu-ilmenau.de/ra1/skripte/spi/studienarbeit_koch.pdf
- Das Programm zum Download http://www.theoinf.tu-ilmenau.de/ra1/skripte/spi/ShorDemoIns.zip
- Homepage von Peter Shor http://www.research.att.com/~shor/
- Meldung aus dem Heise Newsticker über den IBM Quantencomputer Prototypen http://www.heise.de/newsticker/data/wst-20.12.01-003/
- Seminar über Quantencomputer an der Universität Darmstadt http://www.cdc.informatik.tu-darmstadt.de/~cludwig/QuantenseminarWS01/Vortraege/
- Applet zur Diskreten Fourier Transformation http://www.educeth.ch/informatik/interaktiv/dft/applet.html

BILDNACHWEIS

- Beide Abbildungen von http://domino.research.ibm.com/comm/bios.nsf/pages/quantum.html