Quantencomputer

Der Vorteil des Quantencomputers demonstriert am RSA-Verfahren, Kryptoanalyse mit Shor ‘94

von Manuel Grill

Betreuer: Univ.-Prof. Dr.rer.nat. Enrico Arrigoni

1. Einführung[1]

Überall hört undliest man neuerdings von so einem ominösen Konstrukt namens Quantencomputer. Hier ein Durchbruch, da ein Durchbruch. Selbst im akademischen Betrieb hat so mancher Student Schwierigkeiten mit dem genauen Verständnis des Konzeptes Quantencomputer.

Diese Bakkalaureatsarbeit soll die grundlegendsten Sachverhalte des Quantencomputers explizieren und anhand eines Demonstrationsbeispiels (**Kryptoanalyse beim RSA-Verfahren**) zeigen, dass Quantencomputer effizienter sind als klassische Systeme. Dazu wird zunächst das RSA-Verfahren genauer behandelt sowie der Algorithmus, der das RSA-Verschlüsselungsverfahren knackt. Weiters werden die Grundlagen dieser Methode (**Shor-Algorithmus**) genauer betrachtet, und illustriert welche Vorteile dieser Algorithmus gegenüber klassischen Systemen hat (**Quantenparallelismus**).

Die Realisierung des Quantencomputers erweist sich als sehr komplexes Thema, denn so vielfältig wie die Verwirrung und Unwissenheit in diesem Bereich sind, so zahlreich sind die Arbeitsgruppen und noch zahlreicher sind die Lösungsansätze zur Realisierung des Quantencomputers.
Was allerdings gewiss ist: Ein Quantencomputer kann gewisse, geschickt gestellte Fragestellungen **‚schneller'** berechnen als ein klassisches System, das von nur 2 Zuständen abhängt, den berühmten Bits:

1 und *0* oder *true* und *false* oder *Strom fließt* und *tut es nicht* (oder eben weniger)...

Das folgt aus dem **Quantenparallelismus** und dem als 'spukhaft' berühmt gewordenen Prinzip der **Verschränkung**. Diesen enormen Vorteil der Quantenmechanik, mehrer Zustände verwenden zu können, ermöglicht eine Auswertung einer allgemeinen Funktion, für die sonst mehrere Rechenschritte notwendig wären, mit nur **EINEM** Schritt. Eben genau dieser Vorteil bewegt die Wissenschafter dazu, die Forschung am Quantencomputer stark zu forcieren. Beim Auslesen allerdings kann nur wie beim klassischen Computer vorgegangen werden, weil beim Messen die verschränkten Zustände zerstört werden. Genauer betrachtet gibt es mit diesem ‚schneller' Probleme, wenn man die Fehlerkorrektur miteinbezieht. Denn allein schon die Wechselwirkungen mit der Umgebung können diese empfindlichen, verschränkten Zustände zerstören. Trotz dieser Problematik erweist sich der Quantencomputer jedoch letztendlich als das schnellere und damit für gewisse Problemstellungen zu bevorzugende System. Den Quantencomputer zu realisieren ist allerdings kein leichtes Vorhaben, denn so verschränkte Teilchen zu erzeugen und dann wirklich exakt zu wissen, dass diese verschränkt sind, gemeinsam mit der Tatsache, dass Messungen und eben die Wechselwirkungen mit der Umgebung diese Zustände wieder zerstören und dass hier immer wieder Unsicherheiten vorkommen, das alles wird die Wissenschaft noch Jahre beschäftigen.

N.B.
Es kann aus Gründen des Umfangs nicht jeder Beweis näher ausgeführt werden, ein paar kleinere sind dennoch enthalten.

[1] Motiviert vom Artikel „Quant für Quant zum Megarechner" der Tageszeitung „Der Standard" vom Mittwoch, 18.6. 2008, Forschung Spezial, S. 17

2. Grundlagen Kryptographie, Public-Key-Kryptographie[2]

In diesem Kapitel werden die Grundlagen der Kryptographie und vor allem der Public-Key-Kryptographie erarbeitet. Das RSA-Verfahren, eine Public-Key-Verschlüsselungsverfahren, hängt zusammen mit der Faktorisierung großer Zahlen. Dieses Kapitel hat mit dem Quantencomputer eher weniger zu tun, doch wird hier auf die Hintergründe hingewiesen, die zu dem Problem der Faktorisierung geführt haben. Da sich diese Arbeit mit dem Thema Quantencomputer auseinandersetzt, stellt sich auch die Sinnfrage „Wozu Quantencomputer?“. Diese Frage lässt sich am besten durch ein Demonstrationsbeispiel klären, das die Effizienz eines Quantencomputers aufzeigt: Faktorisierungen kann man mit dem Quantencomputer schneller berechnen. Also ergibt sich eine von vielen Anwendungen des Quantencomputers im Bereich der **Kryptographie**.

Was genau heißt eigentlich Kryptographie?
Wenn man dieses Wort genauer betrachtet, fällt einem auf, dass es sich aus den griechischen Wörtern für 'geheim' und 'schreiben' zusammensetzt. Das Thema Kryptographie beschäftigt die Menschheit schon seit Anbeginn der Verschriftlichung. Berühmte Beispiele wären zum Beispiel der Cäsar-Code, Polybios-Code, Freimaurer-Code usw...

All diese Verfahren hatten eines gemeinsam, nämlich einer fremden Person (Angreifer) den Zugang zu wichtigen Nachrichten zu verhindern. Dies kann geschehen mittels eben der Kryptographie und der dazugehörigen mathematischen Methoden oder der Steganographie, die in dieser Arbeit keine Rolle spielt (z.Bsp.: Geheimtinte). Um zu zeigen, dass das System des Quantenparallelismus gegenüber klassischen Systemen Vorteile bringt, muss man sich hier mit der **Public-Key-Kryptographie** auseinandersetzen, weil es notwendig sein wird, eine große Zahl zu faktorisieren und dabei der **Shor-Algorithmus** helfen wird.

Der **Schlüssel**, die allgemeinen Vorschriften (Algorithmus), wie verschlüsselt werden muss, spielt eine große Rolle in der Kryptographie, denn je länger und kompizierter so ein Schlüssel ist, desto sicherer wird die zu verschlüsselnde Information.

...**Klartext** ⇒ **Sender**, verschlüsselt Text ⇒ **Geheimtext** (den **Angreifer** lesen will, aber ohne Schlüssel ist das schwierig) wird gesendet ⇒ **Empfänger**, entschlüsselt ⇒ Klartext...

Da es aber intelligentere Angreifer gibt als jene, die 'nur' durch Ausprobieren dechiffrieren wollen, ist es zwingend notwendig die Schlüssel immer komplizierter werden zu lassen (intelligente Angreifer betreiben: 'Kryptoanalyse') . Jetzt muss man nur weiterüberlegen, was passiert, wenn mehrere Leute miteinander geheim kommunizieren (gemeinsame Schlüssel, neue Teilnehmer, usw.). Was zur Public-Key-Kryptographie führt:

Anders als bei **symmetrischen Verschlüsselungsverfahren** (Sender und Empfänger besitzen den gleichen Schlüssel), stellt sich hier die Frage, ob es überhaupt notwendig ist, dass der Sender die selben Informationen hat wie der Empfänger.

[2] Konzepte, Beweise aus Quelle 2, Kapitel 7, ab Seite 112

⇒ **asymmetrische Kryptographie oder Public-Key-Kryptographie**.

Public-Key-Verschlüsselungsschema

Jeder Teilnehmer *T* hat 2 Schlüssel, den **öffentlichen** E_T (**'public-key'**) und den **privaten** D_T mit folgenden Eigenschaften:

Public-Key-Eigenschaft: privater Schlüssel kann aus öffentlichem nicht berechnet werden!
Entschlüsselungseigenschaft: für Botschaft *b* gilt: $D_T(E_T(b))=b$

Folgendes Prinzip:
Sender *A* holt sich E_B von Empfänger *B* und verschlüsselt Botschaft *b* zu $c = E_B(b)$.
Der geschützte Geheimtext wir nun zu Empfänger geschickt und dieser kann mit $D_B(c)=D_B(E_B(b))=b$ wieder entschlüsseln.
Gilt nun weiters $E_T(D_T(b))=b$ (**Signatureigenschaft**), so kann *A* mit $D_A(b) = Signatur$ sein Dokument signieren. Und wenn er nun *Signatur* und *m* veröffentlicht, kann jeder andere Teilnehmer *T* mit E_T nachschauen, ob ursprüngliche Botschaft *b* herauskommt und somit kontrollieren, ob diese Signatur von *A* stammt (**Signaturschema**).

Diese **Public-Key-Verschlüsselungs- und Signaturschemata** wurden durch R. Rivest, A. Shamir, L. Adleman im Jahre 1978 realisiert. Der **RSA-Algorithmus** ist bis jetzt einer der wichtigsten **Public-Key-Algortihmen**.

Der RSA-Algorithmus

Hint: ggT... größte **g**emeinsame **T**eiler

Satz von Euler ... 1. mathematische Voraussetzung für RSA
Sei n das Produkt zweier verschiedener Primzahlen p ,q . Dann gilt für jede natürliche Zahl m > n und jede natürliche Zahl k:

$$b^{k\varphi(n)+1} \bmod n = b.$$

Wobei $\varphi(n)$ die **eulersche φ – Funktion** ist und es gilt : $\varphi(n) = (p-1)(q-1)$.
$\varphi(n)$ ist für natürliche Zahlen *n* definiert und *p,q* sind Primzahlen.

Auf den **Beweis**[3] wird an dieser Stelle verzichtet, dieser ist ohnehin leicht in der Literatur zu finden, kleine Anregung: den **kleinen Satz von Fermat** mittles Induktion beweisen. Da es hier ja eigentlich um den Quantencomputer geht, und der Umfang nicht explodieren soll, wird nicht auf das bewährte Definition-Satz-Beweis-Schema zurückgegriffen.

[3] Siehe Quelle 2, Kapitel 7, Seite 130

Satz ...2.mathematische Vorraussetzung
Seien a,b ganze Zahlen, wobei a≠0. Mit q und r soll gelten: b = qa + r. Dann gilt:

$$ggT(b,a) = ggT(a,r).$$

Satz (Euklidscher Algorithmus) ... 3. mathematische Voraussetzung
Seien a,b ganze Zahlen und a > 0. Dann kann man den ggT(a,b) auf folgende Art bestimmen:
Schritt 1: Berechne q, r mit b = aq + r und $0 \leq r < a$.
Schritt 2: Wenn $r \neq 0$, dann setze b := a und a := r. Führe Schritt 1 aus, solange bis r = 0, denn dann ist a der gesuchte ggT.

Bsp: *a* = 35, *b* = 101

$$101 = 35 \cdot 2 + 31$$
$$35 = 31 \cdot 1 + 4$$
$$31 = 4 \cdot 7 + 3$$
$$4 = 3 \cdot 1 + 1$$
$$3 = \mathbf{1} \cdot 3 + 0$$

Also ist ggT(35, 101)=1.

Beweis

Nach der 2.mathematischen Voraussetzung gilt ggT(*b*,*a*) = ggT(*a*,*r*) in jedem Schritt.
Bei Abbruch *r* = 0 ist wegen ggT (a,0) = *a* der letzte Wert von *a* der gesuchte ggT.

Satz (Vielfachsummendarstellung) ... 4. mathematische Voraussetzung
Seien a und b ganze Zahlen und d = ggT (a,b) . Dann gibt es ganze Zahlen x,y für die gilt:
$d = x \cdot a + y \cdot b$.
Insbesondere gilt: wenn a,b teilerfremd, dann d = 1.

x, y bestimmbar über erweiterten euklidschen Algorithmus. Dieser besteht im ersten Schritt aus dem euklidschen Algorithmus und im zweiten aus dem Finden der Zahlen *x,y* durch die aus Schritt 1 bekannten Gleichungen, ausgehend von der vorletzten, bei unserem Bsp. war das: $4 = 3 \cdot 1 + 1$.

Schlüsselerzeugung

Für **jeden** Teilnehmer wählt man 2 **große** Primzahlen *p,q*.
Man bildet nun das Produkt : *n = pq*
und die eulersche φ – Funktion: $\varphi(n) = (p-1)(q-1)$.
Man wählt eine natürliche Zahl *f*: ggT(*f*, $\varphi(n)$) = 1.

Weiters bildet man mit dem **erweiterten euklidschen Algorithmus**[4] eine natürliche Zahl *d* :
$fd \bmod \varphi(n) = 1$.
Aus Vielfachsummendarstellung folgt: $d \cdot e + \varphi' \cdot \varphi = 1$.

Das heißt $(fd - 1)$ teilt $\varphi(n)$, oder $fd = k\,\varphi(n) + 1$ für eine natürliche Zahl *k*.

[4] Siehe im Anhang: Quellcodes, Beispiel dazu in Quelle 2, Seite 70

Der öffentliche Schlüssel ist das Paar ***(f,n)***, der private Schlüssel ist ***d***.
Die Nachricht wird als natürliche Zahl dargestellt $b < n$.
Der Sender verschlüsselt nun ***b*** mit ***(f,n)***, dem öffentlichen Schlüssel des Empfängers, und sendet alles als Geheimtext ***c*** zu Empfänger.

$$c := b^f \ mod \ n$$

Dieser entschlüsselt nun mit seinem privaten Schlüssel *d* den Geheimtext *c* zu Botschaft *b*, mittels **Satz von Euler**[5].

$$c^d \ mod \ n = b^{fd} \ mod \ n = b^{k\varphi(n)+1} \ mod \ n = b$$

So lässt sich auch das Signaturschema realisieren.

$$signatur := b^d \ mod \ n$$

Sicherheit des RSA-Algorithmus

Kennt Angreifer $\varphi(n)$, so kann er aus dem öffentlichen Schlüssel den privaten berechnen.
Also muss die Bestimmung von $\varphi(n)$oder die Faktorisierung von *n* schwierig sein.
Kennt Angreifer $\varphi(n)$ so kann er *n* faktorisieren und umgekehrt, kennt er die Faktorisierung kann er $\varphi(n)$ berechnen.

Satz

Sei n das Produkt zweier verschiedener Primzahlen p und q. Dann ist es gleich schwierig $\varphi(n)$ zu bestimmen oder n zu faktorisieren.

Beweis

Kennt man $\varphi(n)$ so kann man mit $\varphi(n) = (p-1)(q-1)$ und $n = pq$
p , q bestimmen. Also *n* faktorisieren.
Und kennt man *p* und *q*, so kann man mit $\varphi(n) = (p-1)(q-1)$
$\varphi(n)$ bestimmen. #

Also reicht es zu sagen: Je komplizierter die Faktorisierung von *n*, desto sicherer ist das RSA-Verfahren.
(Noch nicht bewiesen nach dem Stand von 2004, also sind komplizierte Faktorisierungen nur ein **notwendiges** Kriterium für die Sicherheit). [6]

[5] Siehe Quelle 2, Kapitel 7, Seite 130
[6] Vgl. Quelle 2, Seite 132

Hier setzt man nun mit dem Quantencomputer an, denn eine Faktorisierung mit klassischen Verfahren ist ein *NP*, ein unberechenbares Problem.

Maß für 'Schnelligkeit' von Algorithmen[7]

effiziente Algorithmen = berechenbare Algorithmen, Komplexitätsklasse ***P***
ineffiziente Algortihmen = unberechenbare Algorithmen, Komplexitätsklasse ***NP*** oder ***N***

T... Anzahl der elementaren Rechenschritte
L... Größe des Inputs, Anzahl der Binärstellen, Anzahl Bits

Polynomiale Probleme: *T* wächst ungefähr wie alle Polynome des Inputs $\Rightarrow$ *P*

$$T \sim L^3$$

Exponentielle Probleme: T wächst schneller als Polynome $\Rightarrow$ *NP*

$T \sim e^L$ ...exponentiell

$T \sim e^{L^{\frac{1}{2}}}$ ... subexponentiell

Jetzt ist man mit den Grundlagen des RSA-Verfahrens, den grundlegenden Prinzipien der Public-Key-Kryptographie und mit essentiellen Sätzen der diskreten Mathematik vertraut. Dieses Vorwissen ist natürlich wichtig, um eine vernünftige Kryptoanalyse durchzuführen. Der Begriff der Schnelligkeit wurde eingeführt, um die Vorteile des Quantencomputers gegenüber dem klassischen System darstellen zu können.

[7] Entnommen aus Quelle 3, Seite 73

3. Grundlagen und Gates des Quantencomputers[8]

Für die Kryptoanalyse (Shor-Algorithmus) wird es notwendig sein, sich das Konzept des Quantencomputers zu verinnerlichen. Dieses Kapitel widmet sich den Grundlagen des Quantencomputers.
Um mit quantenmechanischen Methoden ansetzen zu können, muss man mit Begriffen wie **Diracdarstellung**, **Operatoren** in Diracschreibweise, **Observablen**, **Messungen** in der Quantenmechanik, **Zeitentwicklung**, **Hilbertraum**, **Schrödingeroperator**, ... usw. Vertraut sein. Vom **Stern-Gerlach-Experiment** oder **EPR-Paradoxon** sollte man auch schon einmal gehört haben. Diese Vorkenntnise werden **VORRAUSGESETZT**.

Grundsätzlich ...

...interessant sind Fragen bezüglich der **Unschärferelation** und die Auswirkungen in der Quantenwelt:

Ort und Impuls sind gleichzeitig nicht beliebig genau messbar (**Komplementarität**). Aber sind sie nur nicht messbar im Sinne von: ‚wir sind technisch nicht fähig sie zu messen, sie haben aber trotzdem ständig eine gewisse Geschwindigkeit und sind an einem bestimmten Punkt zu einer gewissen Zeit (**Realismus**)?'

Diese Frage ist nicht leicht zu beantworten, aber es stellt sich heraus, dass man diese Zustände mit **Wellen** beschreiben kann. Nicht nur das, man beschreibt sie mit einer **Überlagerung** von mehreren Wellen (also verschiedene Wellenlängen, das bedeutet verschiedenen Impulsen). Diese Zustände haben also eine Vielzahl von möglichen Impulsen, die man je nach Wahrscheinlichkeitsverteilung mit einer gewissen Wahrscheinlichkeit messen kann. Je mehr mögliche Impulse, desto schmaler kann man dieses Wellenpaket machen, also desto bestimmter ist der Ort (Wellenpaket beschreibt **Aufenthaltswahrscheinlichkeit**). Aber durch die Vielzahl der addierten möglichen Impulse ist der Impuls unscharf. Und je breiter dieses Wellenpaket ist, desto unbestimmter ist der Ort und desto weniger Impulse benötigt man, also desto bestimmter ist der Impuls (vgl. **Fouriertransformation, Zerfließen von Wellenpaketen**). Diese Beziehungen gelten für einzelne Teilchen, also ordnet man einem Teilchen ein Wellenpaket zu (vgl. **Doppelspaltexperiment)**.

Observable besitzen demnach statistische Eigenschaften.

Messungen haben in der Quantenwelt generell einen größeren Einfluss als in der makroskopischen Welt. Zustände werden in der makroskopischen Welt nur vernachlässigbar klein beeinflusst, während eine Messung in der Quantenwelt verheerende Auswirkungen auf Zustände haben kann (wie wir später sehen werden spielt das eine große Rolle bei **verschränkten Zuständen**).
Um gewisse Informationen weiterzuleiten oder Operationen auszuführen, muss man diese Zustände allerdings nicht messen.

[8] Konzepte, Beweise hauptsächlich aus Quelle 3
Ergänzungen zum Thema quantum circuit model aus Quelle 4

Im Folgenden wird das mathematische Modell ‚**quantum circuit model**' behandelt.

Analog zum klassischen Computer, der ja zum Bsp. mit Transistoren realisiert und über zum Bsp. eine **'positive' Logik** definert ist, die 2 Zustände kennt:

5V...High, True, 1

0V...Low, False, 0

muss man hier **Quantenbits** definieren, die sogenannten **qubits**.

Realisiert werden diese qubits mit jedem beliebigen 2-Niveau-System, das genauer zu behandeln, würde gemeinsam mit dem Shor-Algorithmus den Rahmen dieser Arbeit sprengen, somit folgt nur ein kleiner Überblick:

qubits

Spin $\frac{1}{2}$ Teilchen

zB.: Stern-Gerlach-Experiment mit Magnetfeld in z-Richtung (Konvention, Darstellung in z-Basis).

$|0\rangle = |S_x\,; +\rangle = \ |+\rangle = |\uparrow\rangle = \binom{1}{0}$ Entspricht also dem klassischen ‚false'.

$|1\rangle = |S_x\,; -\rangle = \ |-\rangle = |\downarrow\ \rangle = \binom{0}{1}$ Und eben das ‚true'.

Photonen

Mit Zuständen zum Beispiel rechts- und linkspolarisiert.

Zwei metastabile Zustände

Zustand $|0\rangle$, man wählt den Grundzustand.
Zustand $|1\rangle$, man wählt den ersten metastabilen Zustand.
Es gibt **keinen** Dipolübergang zwischen $|0\rangle, |1\rangle$.

Beim Shor-Algorithmus ist die Darstellung in der z-Basis der Spin $\frac{1}{2}$ -Teilchen vorteilhaft.

Grundlegende Gates[9]

NOT…

Entspricht einem Spin-Flip oder auch einer Matrixmultiplikation in der z-Basis mit der Paulimatrix σ_x .

Hadamard-Gate... *H*

Hierfür gibt es kein klassisches Äquivalent.
Entspricht einer Drehung um 90°.
Zweifache Anwendung ergibt den Einheitsoperator.

$$H|\uparrow\rangle \rightarrow \frac{1}{\sqrt{2}}(|\uparrow\rangle + |\downarrow\rangle) = |\rightarrow\rangle$$
$$H|\downarrow\rangle \rightarrow \frac{1}{\sqrt{2}}(|\uparrow\rangle - |\downarrow\rangle) = -|\leftarrow\rangle$$

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

In z-Basis:

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

In Matrixdarstellung:

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

Diese Multiplikation setzt sich zusammen aus einer Drehung um die *y*-Achse (um -90°) und aus einer „Phasenrotation“.

Phase-Shift-Gate... U_ϕ

Aus |x⟩ wird $e^{i\phi x}|x\rangle$. Also: $U_\phi|x\rangle = e^{i\phi x}|x\rangle$.
In Matrixdarstellung:

$$U_\phi = \begin{pmatrix}1 & 0\\0 & e^{i\phi x}\end{pmatrix}$$

Mit Hadamard und Phase-Shift-Gates kann man **allgemeine 1-qubit-Transformationen** erzeugen.
⇒ **universeller Satz von 1-qubit-Transformationen**.

[9] Vgl. hierzu Quelle 3, Seite 65

$$U_{\left(\frac{\pi}{2}+\phi\right)}\, H\, U_{2\alpha} H|0\rangle = const. = \cos\alpha\,|0\rangle + \; e^{i\boldsymbol{\phi x}} \sin\alpha\,|1\rangle$$

Die rechte Seite der Gleichung ist aus der Quantenmechanik bekannt (Spin $\frac{1}{2}$ Teilchen!).

Man muss allerdings beachten:

Es gibt natürlich eine Vielzahl von Phase-Shift-Gates (kontinuierlicher Winkel). Man wählt jetzt einfach ein $\boldsymbol{U_{\overline{\phi}}}$ **,** bei dem das Verhältnis Winkel zu 2 π irrational ist und mit diesem speziellen $\boldsymbol{U_{\overline{\phi}}}$ lassen sich alle anderen Phase-Shift-Gates erzeugen (einfach *n* mal $\boldsymbol{U_{\overline{\phi}}}$ anwenden). [10]

Versucht man nun Gates auf 2 qubits anzuwenden, so entspräche das den klassischen Gates ‚and' und ‚or'. Diese sind aber irreversible Operationen (reduzieren den Hilbertraum). Das gibt Probleme in der Quantenmechanik, da für die Zeitentwicklung unitäre Operatoren vorausgesetzt werden (wichtig für Hermitizität des Hamiltonoperators). Also muss man mittels eines Tensorproduktes einzelner qubit-Zustände Korrelationen zwischen den einzelnen Spins einführen.

$|b_1 b_2\rangle\; mit\; b_i \;\in\; \{0,1\}.$ .... $|b_1\rangle \otimes |b_2\rangle$

2-qubit-Gates[11]

Controlled – NOT – Gate … *CNOT*

Wenn das erste qubit true ist, dann wende *NOT* auf das zweite qubit an.
Also tauscht dieses Gate $|10\rangle$ mit $|11\rangle$. Das führt zu einer einfachen Matrixdarstellung:

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

$C_{NOT} = P_1 \otimes Not + P_0 \otimes \mathbb{I}$ ... wobei P_1, P_0 die folgenden Projektoren sind:

$$P_{0,1} \;=\; \tfrac{1}{2}(\,\mathbb{I} \;\pm\; \sigma_z\,)$$

$$P_1 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \ldots \text{Projektor auf } |1\rangle$$

$$P_0 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \ldots \text{Projektor auf } |0\rangle$$

Ganz allgemein gilt Folgendes:

$$C_{NOT}\,|b_1 b_2\rangle \;=\; |b_1\,(b_1 + b_2)_{mod2}\rangle$$

[10] Mehr dazu Quelle 3, Seite 36/37
[11] Vor allem aus Quelle 3, Kapitel 4

Controlled-U-Gate

Macht das gleiche wie das C_{NOT} –Gate, nur für beliebige 1-qubit-Gates *U*.
Also C_U wendet *U* auf 2. qubit an, falls das 1. qubit true ist:
$C_U = P_1 \otimes U + P_0 \otimes \mathbb{1}$

Dieses Gate kann man mit $CBA = \mathbb{1}$ und $C\sigma_x B\sigma_x A = U$, wobei *A,B,C* geeignete 1-qubit-Gates sind, und mit 2 C_{NOT} -Gates ersetzen.

($U_{\overline{\phi}}$, H, C_{NOT}) ist ein universeller Satz für beliebige unitäre Transformationen[12]

Beliebige 1-qubits können mit $U_{\overline{\phi}}$ und H erzeugt werden und fügt man zu diesen 2 noch das C_{NOT} -Gate hinzu, so kann man mit diesen drei Operatoren **beliebige unitäre Transformationen** durchführen. Und das nicht nur für 2-qubits, sondern auch für *n*-qubits (also für einen beliebigen Quantencomputer). Quantennetzwerke bestehen demnach aus 1- und 2-qubit-Gates.

Verschränkte Zustände

Mittels dieser bisher beschriebenen Operationen lassen sich verschränkte Zustände herstellen. Für diese gilt:

1. Können nicht als Produktzustand geschrieben werden.
2. Getrennte, auch wiederholte, Messungen an einzelnen Teilchen können Zustand nicht aufschlüsseln.
3. Können nur durch Wechselwirkungen zwischen Teilchen entstehen
   (2-qubit-Gates, wie C_{NOT}).
   Wechselwirkung darf auch schon in Vergangenheit geschehen sein.

zB.: $C_{NOT}\left(H^{(1)}\,\mathbb{1}^{(2)}\right)|00\rangle = C_{NOT}\,\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\,|0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ ... verschränkter Zustand

Mit diesem Rüstzeug ist man nun in der Lage, sich weiter zu vertiefen und sich ganz spezielle Anwendungen dieser Gates anzusehen. Im nächsten Kapitel wird es um das Auswerten einer Funktion gehen, was wiederum eine Grundlage für den Shor-Algorithmus darstellt, weil es darin um das Finden einer Periode einer periodischen Funktion gehen wird.

[12] Beweis siehe Quelle 5

4. Quantenarithmetik, Quantenparallelismus[13]

Dieses Kapitel behandelt die mathematischen Grundlagen für das effiziente Auswerten von periodischen Funktion (wiederholter Verweis auf den Shor-Algorithmus).

Wichtig für das Auswerten von Funktionen sind sogenannte Register:

$$|b_{n-1}b_{n-2}\dots b_0\rangle$$

also eine Reihe von *n* qubits.

Eine nette Darstellungsmöglichkeit ergibt sich daraus für Dezimalstellen:

$$|1110\rangle = |(13)_{10}\rangle$$

Man kann jetzt zum Bsp. mit einem 3-qubit-Register das ‚and'-Gate reversibel machen (**Toffoli-Gate**). Und das wiederum ist als 3-qubit-Gate laut obigem Satz als 1-qubit und 2-qubit-Gates darstellbar. Was jetzt aber viel interessanter ist:

Auswerten von Funktionen[14]

Ziel: für Input *x*, eine beliebige Funktion *f(x)* berechnen.

Man stellt *x* nun als Register mit *n* qubits dar (Binärdarstellung): $|x\rangle$ und stellt sich vor, dass dieser Zustand durch ein geeignetes Quanennetzwerk zu einem Output $|f_{(x)}\rangle$ führt.
Dass so ein Quantennetzwerk existiert, ist aus dem vorigen Kapitel bekannt, aber nur falls die Transformation unitär ist. *f(x)* muss jetzt in der Regel nicht invertierbar sein, das wiederum würde aber Probleme mit der geforderten Unitarität bringen. Dieses Problem wird analog zu dem vorherigen Fall ‚and' gelöst, indem man ein Input-Register und ein Output-Register einführt. Da in der Regel die Funktion jetzt aber nicht gleich viele bits haben muss wie *x*, sind diese Register von unterschiedlicher Länge.
Die Transformation nennen wir U_f.

$Input - Register\ \ mit\ n\ qubits$: $|x\rangle$
$Output - Register\ der\ Länge\ m$: $|y\rangle$

Im Anfangszustand sei *x* der Input und *y* wird auf *y* = 0 präpariert.
U_f wirkt jetzt nur auf das Output-Register und schreibt ihm den Wert *f(x)* zu. Das Input-Register bleibt ‚unberührt', also:

$$U_f\,|x,y\rangle = |x\,,(y + f_{(x)})_{mod2^m}\rangle$$

Dies hier ist also invertierbar und unitär, es existiert übrigens auch ein klassisches Analogon.
mod_{2^m} ist selbstverständlich bei *m* qubits, also wird es in Zukunft weggelassen.

[13] Inhalte, Beweise aus Quellen 1 und 3
[14] Ausführlicher siehe Quelle 3, Kapitel 4, Seite 40

Alles schön und gut, doch einen Vorteil gegenüber eines klassischen Systems sieht man hier immer noch nicht. Erst jetzt ist man in der Lage sich den **einzigen**, aber dafür umso **wichtigeren** Vorteil näher anzusehen, den ein Quantencomputer gegenüber dem klassischen hat:

QUANTENPARALLELISMUS

Das Schöne an der Quantenmechanik ist, dass sich Zustände als Überlagerung von mehreren Basiszuständen beschreiben lassen (Linearität).
Man wählt nun einen Anfangszustand $|\psi\rangle$ und stellt ihn mit den Basiszuständen des ersten Registers dar:

$$|\psi\rangle = \sum_{x=0}^{2^n-1} |x, y = 0\rangle$$

$2^n - 1$ deshalb, weil n bits im Input- Register sind, daher hat man mit 2 Zuständen 0,1 genau 2^n Möglichkeiten und da wir von 0 an begonnen wird, ziehen man noch 1 ab.

Jetzt sieht man den Vorteil gegenüber klasischen Systemen, denn wendet man auf $|\psi\rangle$ U_f an, so wird f(x) für alle Basiszustände des ersten Registers mit nur **EINEM SCHRITT** ausgewertet.

$$U_f|\psi\rangle = \sum_{x=0}^{2^n-1} |x, f_{(x)}\rangle \equiv |Lösung\ für\ alle\ Zustände\ mit\ nur\ einem\ Schritt\rangle$$

Ein Bsp.[15]

n = 100 bits und die benötigeten Operationen des klassischen Computers:

$$x = \{0,1, \dots, 2^{100} - 1\}$$

⇒ 2^{100} Operationen, also etwa 10^{30}.
Vergleicht man dazu die Rechenoperationen des Quantencomputers: 1 (!).

Definiert man für ein klassisches System etwa 10^9 Rechenoperationen pro Sekunde, so bräuchte er dafür rund 10^{14} Jahre.
Also selbst wenn die Entwicklung des Quantencomputers noch Jahre dauert und er dann auch noch extrem langsam wäre (ist auch unplausibel, davon auszugehen; aber sagen wir er bräuchte ein Jahr pro Rechenoperation), so stünden gegenüber: 10^{14} Jahre (klassisch) vs. (Entwicklungsjahre+1) (quantenmechanische System).

⇒ exponentiell große Rechenleistungssteigerung!!

[15] Aus Quelle 3, Seite 42

„Probleme“

Mit der Überlagerung benötigt man zwar nur eine Rechenoperation, aber messen kann man, wie beim klassisches System, nur die einzelnen Konfigurationen $x = \bar{x}$ (alle Werte von *x* gleich wahrscheinlich). Die wirklichen Probleme des Quantencomputers liegen aber vor allem bei der Fehlerkorrektur. Ein paar Stichworte: Dekohärenz, Phasenfehler[16], ...

Messung

Der Zustand $|Lösung\rangle$ wird nun mit $P_{\bar{x}}$ projiziert.
$P_{\bar{x}} = |\bar{x}\rangle\langle\bar{x}| \otimes \mathbb{I}$ ... der Einheitsoperator wirk auf das Output-Register.

$$P_{\bar{x}} \sum_{x} |x, f_{(x)}\rangle = \sum |\bar{x}\rangle\langle\bar{x}|x\rangle \otimes \mathbb{I}\, |f_{(x)}\rangle = \sum \delta_{x,\bar{x}}\, |\bar{x}\rangle \otimes |f_{(x)}\rangle = |\bar{x}, f_{(\bar{x})}\rangle$$

Jetzt kann man *y* problemlos messen und man bekommt zu 100% $f(\bar{x})$.

Mit diesen Grundlagen kann man sich nun auf die Suche nach speziellen Algorithmen machen, man muss nur geschickt die Eigenschaften der Interferenz ausnutzen, um diese Algorithmen effizient zu gestalten. Im nächsten Kapitel geht es um die Quantenfouriertransformation und diese zu verstehen ist unerlässlich, will man sich mit dem Shor-Algorithmus beschäftigen. Denn es geht in der Quantenfouriertransformation ja auch um das geschickte Spielen mit Interferenzen.

[16] Auf Fehlerkorrekturen dieser Art geht diese Arbeit nicht ein, mehr dazu in den Quellen 1 und 3

5. Quantenfouriertransformation QFT[17]

Bei der Faktorisierung großer Zahlen ist es, wie später ausgeführt wird, von Vorteil Perioden in einer Funktion finden zu können. Das allerdings geht leichter, falls man die Quantenfouriertransformation beherrscht, eine diskrete Fouriertransformation, die nun in diesem Kapitel illustriert wird.

Allgemeines zur Quantenintegraltransformation

Sei n eine natürliche Zahl und $S_n = \{0,1, \dots 2^n - 1\}$.
So benötigt man folgende Abbildungen:

$$K: S_n \times S_n \to \mathbb{C}$$
$$f: S_n \to \mathbb{C}$$

Also für jede Abbildung f ist $\tilde{f}: S_n \to \mathbb{C}$ mit dem Kern K eine diskrete Integral Transformation definiert:

$$\tilde{f}(y) = \sum_{x=0}^{2^n-1} K(y,x) f(x)$$

Schreiben man nun K als Matrix:

$$K = \begin{pmatrix} K(0,0) & \cdots & K(0,N-1) \\ K(1,0) & & K(1,N-1) \\ \vdots & \ddots & \vdots \\ K(N-2,0) & & K(N-2,N-1) \\ K(N-1,0) & \cdots & K(N-1,N-1) \end{pmatrix}$$

Und die Funktion f als Vektor:

$$f = (f(0), f(1), \dots f(N-1))^t$$

Wobei $N \equiv 2^n$.
Dann vereinfacht sich diese Transformation zu:

$$\tilde{f} = Kf$$

Angenommen, der Kern sei unitär, d.h.: $K^\dagger = K^{-1}$. Dann existiert eine inverse Transformation:

[17] Inhalte, Beweise aus Quelle 1, Kapitel 6, ab Seite 109

$$f(x) = \sum_{y=0}^{2^n-1} K^{\dagger}(x,y)\tilde{f}(y)$$

(ohne Beweis).

Man definiert U als eine $N \times N$ unitäre Matrix.

$|x\rangle = |x_{n-1}x_{n-2} \dots x_0\rangle$ sei dann eine binäre Standardbasis vom Hilbertraum mit $x_k \epsilon \{0,1\}$, also $x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_0 2^0$.

Dasselbe soll gelten für ein zweites *n*-qubit-Register $|y\rangle$. Der Hilbertraum hat folgende Form: $H = (\mathbb{C}^2)^{\otimes n}$, also $|x\rangle \otimes |y\rangle \epsilon H$.
So kann man schreiben:

$$U|x\rangle = \sum_{y=0}^{N-1} |y\rangle\langle y|U|x\rangle = \sum_{y=0}^{N-1} U(y,x)|y\rangle$$

$U(x,y) = \langle x|U|y\rangle$ ist eine komplexe Zahl und Element der Matrix U.

Satz

Ist nun

$$U|x\rangle = \sum_{y=0}^{N-1} K(y,x)|y\rangle$$

dann vollzieht U die diskrete Inegral Transformation (DIT):

$$\tilde{f}(y) = \sum_{x=0}^{2^n-1} K(y,x)f(x)$$

Und zwar im folgenden Sinne:

$$U\left[\sum_{x=0}^{N-1} f(x)|x\rangle\right] = \sum_{y=0}^{N-1} \tilde{f}(y)|y\rangle$$

Die DIT wird wieder in nur einem Schritt für alle Variablen *y* vollzogen!
Quantenparallelismus mit der Superposition $\sum_{x=0}^{N-1} f(x)|x\rangle$.

Beweis

$$U\left[\sum_{x=0}^{N-1} f(x)|x\rangle\right] = \sum_{x=0}^{N-1} f(x)U|x\rangle = \sum_{x=0}^{N-1} f(x)\left[\sum_{y=0}^{N-1} K(y,x)|y\rangle\right]$$

$U|x\rangle = \sum_{y=0}^{N-1} K(y,x)|y\rangle$ laut Annahme.

$$= \sum_{y=0}^{N-1}\left[\sum_{x=0}^{N-1} K(y,x)f(x)\right]|y\rangle = \sum_{y=0}^{N-1} \tilde{f}(y)|y\rangle \ \#$$

QFT... Quantenfouriertransformation

Die QFT ist nun eine DIT mit dem Kern:

$$K(x,y) = \frac{1}{\sqrt{N}}{\omega_n}^{-xy}$$

mit

$$\omega_n = e^{\frac{2\pi i}{N}} \ und \ N \equiv 2^n$$

Sie wird auch als diskrete Fouriertransformation (DFT) bezeichnet.

$$\tilde{f}(y) = \sum_{x=0}^{2^n-1} K(y,x)f(x) = \frac{1}{\sqrt{N}}\sum_{x=0}^{2^n-1} {\omega_n}^{-xy} f(x)$$

Satz *Der Kern ist unitär.*

Beweis

$$\left(KK^{\dagger}\right)(x,y) = \langle x|K\sum_{z}|z\rangle\langle z|K^{\dagger}|y\rangle = \sum_{z} K(x,z)K^{\dagger}(z,y) = \frac{1}{N}\sum_{z}{\omega_n}^{-xz}{\omega_n}^{zy}$$
$$= \frac{1}{N}\sum_{z}{\omega_n}^{z(y-x)} = \frac{1}{N}\sum_{z} e^{\frac{-2\pi iz(x-y)}{N}} = \delta_{x,y} = \mathbb{1}(x,y) \ \#$$

Weitere Eigenschaften

Der Kern für *n* = 1 entspricht dem Hadamard-Gate!!

$$K_1 = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & e^{\frac{2\pi i}{2}}\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} = H$$

Die inverse DFT ist:

$$f(x) = \frac{1}{\sqrt{N}}\sum_{y=0}^{2^n-1} {\omega_n}^{xy}\,\tilde{f}(y)$$

Interessant ist :

$$U_{QFTn}|0\rangle = \frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}|y\rangle$$

Also eine Anwednung von U_{QFTn} hat aus $|0\rangle$ eine Superposition der Basisvektoren: $\frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}|y\rangle$ erzeugt!!

Anwendungsbeispiel QFT, finden einer Periode[18]

Deswegen interessant, weil bei der Faktorisierung wieder eine Periode gesucht wird.

Grob gesagt wird hier auf $|x\rangle|y=0\rangle$ eine beliebige Transformation U_f (siehe ‚Auswerten von Funktionen' Kap. 4) angewendet und einmal eine QFT durchgeführt (siehe oben) und dann kann man sehen, dass, falls eine Periode existiert, Messungen ganz interessante Eigenschaften aufweisen.

Hier: *n* = 3, also ein 3-qubit-System
$S_3 = \{0,1,\dots 2^3-1\} = \{0,1,\dots 7\}$
$f\colon S_3 \rightarrow S_3\,, f(x) = f(x+P), P = 2$
$|Input-Register\rangle = \frac{1}{\sqrt{2^3}}(|000\rangle + \dots + |111\rangle) = \frac{1}{\sqrt{2^3}}(|0\rangle + |1\rangle + \dots + |7\rangle)$

$$|\psi\rangle = \frac{1}{\sqrt{2^3}}\sum_{x=0}^{7}|x\rangle|0\rangle = \frac{1}{\sqrt{8}}(|0,0\rangle + |1,0\rangle + \dots + |7,0\rangle)$$

Einmal unitäre Transformaton liefert:

$$|\psi'\rangle = \frac{1}{\sqrt{2^3}}\,U_f\sum_{x=0}^{7}|x\rangle|0\rangle = \frac{1}{\sqrt{8}}\sum_{x=0}^{7}|x\rangle|f(x)\rangle = \frac{1}{\sqrt{8}}(|0,f(0)\rangle + \cdots + |7,f(7)\rangle)$$

N.B. Nur Diagonalelemente, wenn man Koeffizient $\frac{1}{\sqrt{8}}$ in Matrix ($|i\rangle, |f(j)\rangle$) betrachtet, sind nicht 0. Siehe Abbildung unten.

[18] Aus Quelle 1, ab Seite 111

$\|f(7)\rangle$	0	0	0	0	0	0	0	$\frac{1}{\sqrt{8}}$
$\|f(6)\rangle$	0	0	0	0	0	0	$\frac{1}{\sqrt{8}}$	0
$\|f(5)\rangle$	0	0	0	0	0	$\frac{1}{\sqrt{8}}$	0	0
$\|f(4)\rangle$	0	0	0	0	$\frac{1}{\sqrt{8}}$	0	0	0
$\|f(3)\rangle$	0	0	0	$\frac{1}{\sqrt{8}}$	0	0	0	0
$\|f(2)\rangle$	0	0	$\frac{1}{\sqrt{8}}$	0	0	0	0	0
$\|f(1)\rangle$	0	$\frac{1}{\sqrt{8}}$	0	0	0	0	0	0
$\|f(0)\rangle$	$\frac{1}{\sqrt{8}}$	0	0	0	0	0	0	0
	$\|0\rangle$	$\|1\rangle$	$\|2\rangle$	$\|3\rangle$	$\|4\rangle$	$\|5\rangle$	$\|6\rangle$	$\|7\rangle$

Abbildung 1 - Koeffizienten vor der QFT

Jetzt wird eine QFT vorgenommen, und man sieht, dass die vorher betrachtete Matrix nun scheinbar komplizierter geworden ist. Beachtet man aber die aus der Messung (Register1) gefundene Periode *P*(hier = 2), so ergibt sich eine einfachere Matrix.

$$QFT\ am\ 1.Register\colon |x\rangle \ \rightarrow \frac{1}{\sqrt{8}} \sum_{z=0}^{7} e^{\frac{-2\pi i x z}{8}} |z\rangle$$

$$\Rightarrow |\psi''\rangle = \frac{1}{8} \sum_{z,x} e^{\frac{-2\pi i x z}{8}} |z, f(x)\rangle$$

Hier stehen die imaginären Koeffizienten wild durcheinander[19]:

$\|f(7)\rangle$	→	↘	↓	↙	←	↖	↑	↗
$\|f(6)\rangle$	→	↓	←	↑	→	↓	←	↑
$\|f(5)\rangle$	→	↙	↑	↘	←	↗	↓	↖
$\|f(4)\rangle$	→	←	→	←	→	←	→	←
$\|f(3)\rangle$	→	↖	↓	↗	←	↘	↑	↙
$\|f(2)\rangle$	→	↑	←	↓	→	↑	←	↓
$\|f(1)\rangle$	→	↗	↑	↖	←	↙	↓	↘
$\|f(0)\rangle$	→	→	→	→	→	→	→	→
	$\|0\rangle$	$\|1\rangle$	$\|2\rangle$	$\|3\rangle$	$\|4\rangle$	$\|5\rangle$	$\|6\rangle$	$\|7\rangle$

Abbildung 2 – Koeffizienten nach der Fouriertransformation ↗ bedeutet $e^{i\pi/4}$.

[19] Genauer in Quelle 1, Seite 113-115

Was hat die QFT also gebracht?
Wenn man von der gemessenen Periode *P*=2 ausgeht, so kann man sagen:

$$f(0) = f(2) = f(4) = f(6) = a$$
$$f(1) = f(3) = f(5) = f(7) = b$$

Bei $a, b \in S_3$ und $a \neq b$.

$$\Rightarrow |\psi''\rangle = \frac{1}{8} \sum_{z,x \in S_3} e^{\frac{-2\pi i x z}{8}} |z, f(x)\rangle$$

Der Zustand $|\psi''\rangle$ hat sich also reduziert.

$\|b\rangle$	⟶	0	0	0	⟵	0	0	0
$\|a\rangle$	⟶	0	0	0	⟶	0	0	0
	$\|0\rangle$	$\|1\rangle$	$\|2\rangle$	$\|3\rangle$	$\|4\rangle$	$\|5\rangle$	$\|6\rangle$	$\|7\rangle$

Abbildung 3 - Koeffizienten nach QFT und Berücksichtigung der Periode *P* = 2.

Somit verbleiben nur die Vektoren: $|0, a\rangle$,$|0, b\rangle$, $|4, a\rangle$, $|4, b\rangle$.
Die Amplitude der sich nicht auslöschenden Koeffizienten ist $\frac{1}{2}$.

$$\Rightarrow |\psi''\rangle = \frac{1}{2}(|0, a\rangle + |0, b\rangle + |4, a\rangle + e^{-i\pi}|4, b\rangle)$$

Misst man jetzt also $|Input - Register\rangle$, dann bekommt man entweder 0 oder 4.
Und dieses Messergebnis hängt direkt mit der Periode *P* = 2 zusammen.
Misst man also 0 oder 4, so kann man auf *P* = 2 Rückschlüsse ziehen.

Später wird der folgende Satz von Bedeutung sein:

Satz[20]

Angenommen: es sind n-qubit-Register, $Q = 2^n$, P ... Periode, m = Q/P sei eine natürliche Zahl, f(x) sei periodisch, dann ist der gemessene Wert des Registers z einer der folgenden:

$$0, \frac{1 \cdot 2^n}{P}, \frac{2 \cdot 2^n}{P}, \frac{3 \cdot 2^n}{P}, \ldots \frac{(P-1) \cdot 2^n}{P}$$

[20] Quelle 1, Seite 407, Übungsaufgabe aus Kapitel 6, Beispiel 3

Beweis

$$|\psi'\rangle = \frac{1}{Q}\sum_{z,x=0}^{Q-1} e^{-2\pi i x z/Q}|z\rangle|f(x)\rangle \; nach \; der \; QFT$$

$$\sum_{x=0}^{Q-1} h(x) \; \rightarrow \sum_{l=0}^{P-1}\sum_{k=0}^{m-1} h(kP+l)$$

$$|\psi'\rangle = \frac{1}{Q}\sum_{l=0}^{P-1}\sum_{k=0}^{m-1}\sum_{z=0}^{Q-1} e^{-2\pi i l z/Q} e^{-2\pi i k z/m}|z\rangle|f(kP+l)\rangle$$

$$|\psi'\rangle = \frac{1}{Q}\sum_{z=0}^{Q-1}\sum_{k=0}^{m-1} e^{-2\pi i k z/Q} \sum_{l=0}^{P-1} e^{-2\pi i l z/Q}|z\rangle|f(l)\rangle, weil \; ja \; f(l) = f(kP+l)$$

Annahme: $z \; = \; q \cdot m \; (f\ddot{u}r \; 0 \leq q \; \leq P-1), dann$:

$$\sum_{k=0}^{m-1} e^{-\frac{2\pi i k z}{m}} = \sum_{k=0}^{m-1} e^{-\frac{2\pi i k q m}{m}} = m$$

Falls $z \; \neq \; q \cdot m$:

$$\sum_{k=0}^{m-1} e^{-\frac{2\pi i k z}{m}} = \frac{1-e^{-2\pi i z}}{1-e^{-\frac{2\pi i z}{m}}} = 0$$

$$|\psi'\rangle = \frac{m}{Q}\sum_{l=0}^{P-1}\sum_{q=0}^{P-1} e^{-2\pi i l q/P}|qm\rangle|f(l)\rangle$$

Und man erhält:

$$qm = \frac{qQ}{P} = \frac{q2^n}{P} \; f\ddot{u}r \; (0 \; \leq q \; \leq P-1). \#$$

Weil ja *m = Q/p* eine natürliche Zahl ist.

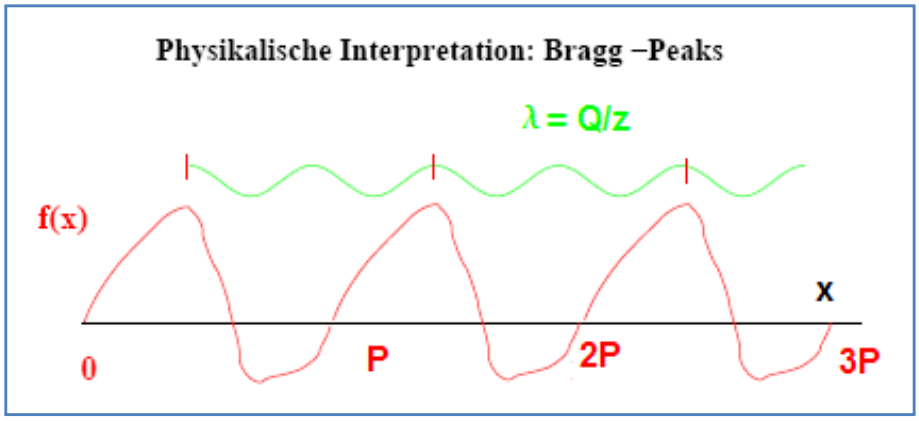

Abbildung 4 - Physikalische Interpretation bei der QFT von periodischen Funktionen.

z = (dQ)/P, also $\lambda d = P$. Es entsteht konstruktive Interferenz bei *Q/z*.
f(x) ist eine periodische Funktion.

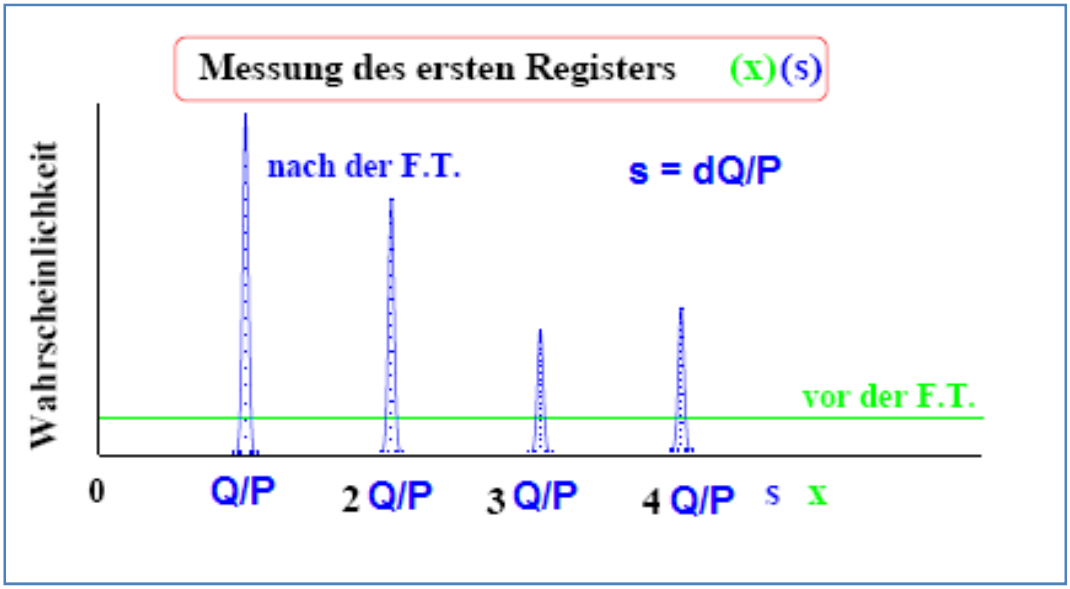

Abbildung 5 - Gegenüberstellung der Wahrscheinlichkeiten vor und nach der QFT. Hier ist *s* die Fouriertransformierte.

In den Büchern und Skripten, die hierfür durchgearbeitet wurden, stand an dieser Stelle, wie und mit welchen Gates man die QFT realisiert. Für den Zweck dieser Arbeit reicht es allerdings, auf den universellen Satz für beliebige, unitäre Transformationen hinzuweisen. [21]

Jetzt ist man in der Lage auf Perioden in einer Funktion Rückschlüsse zu ziehen, was ungeheuer wichtig für den Shor-Algorithmus ist. Dieser soll nun endlich im nächsten Kapitel genauer behandelt werden. Generell beruhen alle bisher bekannten Quantenalgorithmen auf dem geschickten Ausnutzen von Interferenzen.

[21] Für Beschreibung von Realisierungen siehe unter anderem Quelle 1

6. Der Shor-Algorithmus (Shor '94)[22]

Mit den bisherigen erarbeiteten Grundlagen hat man nun das Rüstzeug, sich mit dem Faktorisierungsalgorithmus von Shor zu beschäftigen. Wie man aus Kapitel 2 weiß, geht es um das Problem der Faktorisierung einer großen Zahl *N*. Für das bessere Verständnis und für den Überblick befassen sich das Kapitel 8 und 9 mit Anschaungsbeispielen. Der 2.Schritt wird im nächsten Kapitel genauer betrachtet, in diesem 2. Schritt wird es notwendig sein das ganze Wissen aus den vorigen Kapiteln anzuwenden.

N=pq, wobei *p* und *q* Primzahlen sind.

Klassische Algorithmen

Trivialer kl. Algorithmus

Alle Zahlen von $r = 1, \dots, \sqrt{N}$ probieren und $\frac{N}{r}$ bilden.
$T \sim e^{\frac{L}{2}}$ ... Anzahl Iterationen

Bester kl. Algorithmus

$T \sim e^{\sqrt[3]{L \cdot \log(L)^2}}$ ... schneller als exponentiell, aber immer noch ineffizient, also *NP*

Gegenüberstellung Rechenzeit bei Faktorisierung[23]

Dezimalstellen	Klass. Algorithmus (1GHz)	Shor '94 (1MHz)
100	**150 Tage**	**2.5 Stunden**
300	**6 Millionen Jahre**	**2.5 Tage**
L	$\boldsymbol{T \sim exp(\sqrt[3]{L})}$	$\boldsymbol{T \sim L^3}$

Abbildung 6 - Gegenüberstellung der Rechenzeiten

[22] Grundkonzept, Beweise aus Quelle 1, Seite 137; Ergänzungen zur Rechenzeit aus Quelle 3
[23] Entnommen aus Quelle 3, Seite 12

Der Shor- Algorithmus

Der S.-A. besteht aus 5 Schritten und alle bis auf einen sind klassischer Natur. Schritt 2 ist der kleine, aber feine Unterschied. Beim Finden der Periode im Schritt 2, dieser hier ist der zeitaufwändigste Schritt, hilft der Quantencomputer und macht aus dem klassischen *NP*- Problem ein *P*-Problem.

Die 5 Schritte, genauer betrachtet[24]:

Schritt 1

Eine natürliche Zahl $m < N$ wird zufällig gewählt (Nicht jedes m erfüllt die 5 Schritte!!).
Berechne ggT(m,N) (Euklidscher Algorithmus vgl. Kap. 2).
Der Euklidsche Algorithmus ist ein polynomiales, berechenbares Problem[25]. $T \sim \log(N)$
Falls jetzt ggT$\neq 1$, ist man fertig: *m* ist entweder *p* oder *q*.
Sonst weiter mit Schritt2.

Schritt 2

Defintion:
$f_{m,N} : \mathbb{N} \rightarrow \mathbb{N}$
$x \longmapsto m^x \, mod \, N$ ... $f_{m,N(x)} = m^x \, mod \, N$

Finde die **kleinste** natürliche Zahl *P*, sodass gilt: $\boldsymbol{m^P \equiv 1 \, mod \, N}$.
P... Periode.

Dieser Schritt 2 ist der aufwändigste dieser 5 Schritte und kann nur mit quantenmechanischen Mitteln effizient berechnet werden. Alle anderen Schritte können auch mit klassischen Systemen in polynomialen Schritten berechnet werden.

Schritt 3

Ist *P* ungerade, dann gehe zu Schritt 1, weil unbrauchbar für Schritt 4 und 5. Wiederhole Schritte bis *P* and dieser Stelle gerade.
Ist *P* gerade, weiter mit Schritt 4.

Schritt 4

Mit geradem *P* gilt:

$$\left(m^{\frac{P}{2}} - 1\right)\left(m^{\frac{P}{2}} + 1\right) = m^P - 1 \equiv 0 \, mod \, N$$

Falls $\left(m^{\frac{P}{2}} + 1\right) \equiv 0 \, mod \, N$, dann ist $ggT\left(\left(m^{\frac{P}{2}} - 1\right), N\right) = 1$.
Das wiederum bedeutet zurück zu Schritt 1.

[24] Quelle 1, Seite 140
[25] Vgl. Quelle 4, Seite 10

Falls $\left(m^{\frac{P}{2}}+1\right) \not\equiv 0\ mod\ N$, dann weiter mit Schritt 5.

Ein paar wichtige Bemerkungen zu Schritt 4:
$\left(m^{\frac{P}{2}}-1\right)$ enthält nun *p* oder *q* (falls Bedingung für Schritt 5 erfüllt ist).
$\boldsymbol{\left(m^{\frac{P}{2}}-1\right)}$ **kann kein Vielfaches von N sein, sonst würde** $\boldsymbol{m^{\frac{P}{2}} \equiv 1\ mod\ N}$ **gelten und das wiederum bedeutet, dass *P* nicht die kleinste Zahl ist für die gilt:** $\boldsymbol{m^{P} \equiv 1\ mod\ N}$**.**
Das wäre ein Widerspruch zur Annahme!

Schritt 5

Die Zahl $k = ggT\left(\left(m^{\frac{P}{2}}-1\right), N\right)$ ist nun entweder *p* oder *q*. Fertig.

Bsp.:

N = 91
Zufällig gewähltes *m* = 3 (bei *m* = 7 wäre man sofort fertig! Vergleiche: $ggT(91{,}7\ oder\ 13) \neq 1$)

Schritt 1
Also $ggT(91{,}3) = 1$

Schritt 2

f(x)	**1**	3	9	27	81	61	**1**	3
x	**0**	1	2	3	4	5	**6**	7

Schritt 3
Daraus folgt die Periode *P* = 6, ist gerade, weiter mit Schritt 4.

Schritt 4
$(3^3+1) = 28 \not\equiv 0\ mod\ 91$, also weiter mit Schritt 5.

Schritt 5
$d = ggT(26{,}91) = 13 = q\ oder\ p$, fertig.
$Check$: $13 \cdot 7 = 91$, ok. ☺

Bestimmt man nun die **Periode *P* von der Funktion** $\boldsymbol{f_{m,N}}$ in **effizienter**[26] Art und Weise, so kann man auch ***N* effizient faktorisieren**. Schritt 1,3,4,5 sind klassische, polynomiale Problemstellungen[27].

[26] Hinweise auf Effizienz in jedem Schritt, Quelle 4, Seite 10
[27] Siehe Quelle 3, Seite74,75

Dass das für einen klassischen Computer ein *NP*-Problem ist, kann man sich folgender Maßen überlegen:

Die Funktion berechnet man sich ja schrittweise, also: *x*, *x*+1, *x*+2, ... bis zu dem Zeitpunkt, an dem sie sich wiederholt (Periode *P*).

Man muss die Funktion also $\sim P$- mal ausrechnen. P ist aber $\sim N^{Potenz} \sim e^{L \cdot Potenz}$.

Dieses grundlegende Konzept ermöglicht ein einfaches Lösen verschiedenster Problemstellungen zur Kryptoanalyse des RSA-Verfahrens. Der 2. Schritt muss allerdings noch genauer betrachtet werden, hier spielen ja die Komponenten der Quantenmechanik eine große Rolle, wie man im nächsten Kapitel sehen wird.

7. Genaue Betrachtung des 2. Schrittes im Shor-Algorithmus[28]

In dieses Kapitel gehen nun alle Überlegungen der Kapitel über den Quantencomputer, Quantenparallelismus, Quantenarithmetik und der Quantenfouriertransformation ein. Aus Gründen des Überblicks sei hier nochmal erwähnt, dass der Shor-Algorithmus 5 Schritte umfasst und weiters sei nochmal zur Wiederholung betont, dass man mit eben diesen Schritten eine große Zahl faktorisiert, was wiederum das Knacken des RSA-Verfahrens ermöglicht. Das ist nur mit einem Quantencomputer effizient möglich.

Man definiert: $0 \leq x \leq 2^n - 1$ und sucht die Periode der Funktion :

$f_{m,N} : \mathbb{N} \to \frac{\mathbb{Z}}{\mathrm{N}\mathbb{Z}}$ (also für *x* und *x+kN*), $x \longmapsto m^x \, mod \, N$
$f_{m,N(x)} = m^x \, mod \, N$

Voraussetzungen für genauere Behandlung des 2. Schrittes

Eine natürliche Zahl *N=pq* soll mit *p,q* Primzahlen faktorisiert werden.
Man benötigt eine natürliche Zahl *n*:

$$N^2 \leq 2^n \leq 2N^2$$

Man schreibt $Q = 2^n$ und $S_n = \{0,1, \dots Q-1\}$, was wiederum $(f: a \longmapsto m^a \, mod \, N)$ einschränkt: $(f: S_n \to \mathbb{Z}/\, \mathrm{N}\mathbb{Z})$.

Man benötigt zwei *n*-qubit-Register:

$$|REG1\rangle \, |REG2\rangle = |a\rangle|b\rangle = |a_{n-1}a_{n-2} \dots a_0\rangle|b_{n-1}b_{n-2} \dots b_0\rangle$$

Dezimalzahlen *a, b* $\epsilon \, S_n$ werden binär dargestellt:

$$a = \sum_{j=0}^{n-1} a_j 2^j \, ; b = \sum_{j=0}^{n-1} b_j 2^j$$

Eine QFT wird an einem Register angewendet:

$$|x\rangle \; \to \; U_{QFTn}|x\rangle = \frac{1}{\sqrt{Q}} \sum_{\mathrm{z}=0}^{\mathrm{Q}-1} {\omega_n}^{-zx} \, |z\rangle$$

und U_{QFTn} als $\mathcal{F}$ geschrieben; *x,z* $\epsilon \, S_n$ und $\omega_n = e^{\frac{-2\pi i}{Q}}$.

[28] Aus Quelle 1, ab Seite 141

Der 2. Schritt, Finden einer Periode

2.0 Anfangszustand

$$|\psi_0\rangle = |REG1\rangle\,|REG2\rangle = |00\ldots0\rangle\,|00\ldots0\rangle$$

2.1 Anwenden von $\mathcal{F}$ am ersten Register...

$$|\psi_0\rangle = |REG1\rangle\,|REG2\rangle = |0\rangle|0\rangle \overset{\mathcal{F}\otimes\mathbb{I}}{\longmapsto} |\psi_1\rangle = \frac{1}{\sqrt{Q}}\sum_{\mathrm{x}=0}^{\mathrm{Q}-1}|x\rangle\,|0\rangle$$

... erzeugt eine Überlagerung der Zustände $|x\rangle$ $(0 \leq x \leq Q-1)$ des ersten Registers.

2.2 Anwenden von U_f

$$U_f|x\rangle\,|0\rangle = |x\rangle|f(x)\rangle;\ U_f|\psi_1\rangle = |\psi_2\rangle = \frac{1}{\sqrt{Q}}\sum_{\mathrm{x}=0}^{\mathrm{Q}-1}|x\rangle\,|f(x)\rangle$$

Man beachte die Verschränktheit beider Register.

2.3 Nochmaliges Anwenden von $\mathcal{F}$ am ersten Register

$$|\psi_3\rangle = \mathcal{F}\otimes\mathbb{I}\,|\psi_2\rangle = \frac{1}{Q}\sum_{\mathrm{x}=0}^{\mathrm{Q}-1}\sum_{z=0}^{Q-1}{\omega_n}^{-zx}\,|z\rangle\,|f(x)\rangle = \frac{1}{Q}\sum_{z=0}^{Q-1}|z\rangle\,|g(z)\rangle$$

$$= \frac{1}{Q}\sum_{z=0}^{Q-1}\||g(z)\rangle\|\,|z\rangle\ \frac{|g(z)\rangle}{\||g(z)\rangle\|}$$

Wobei:

$$|g(z)\rangle = \sum_{x=0}^{Q-1}{\omega_n}^{-zx}\,|f(x)\rangle$$

2.4 Messen des ersten Registers und Wahrscheinlichkeiten

Durch Messung an $|REG1\rangle$ erhält man $z \in S_n$ mit der Wahrscheinlichkeit:

$$W(z) = \frac{\||g(z)\rangle\|^2}{Q^2}.$$

Und jetzt wird klar, warum man nur einzelne Werte auslesen kann, denn zur selben Zeit kollabiert der Zustand zu (vgl.: Polarisationsfilter, Pojektionsoperatoren):

$$|\psi_4\rangle = |z\rangle \frac{|g(z)\rangle}{\||g(z)\rangle\|}$$

W(z) ist noch genauer zu betrachten:

Genauer gesagt handelt es sich hier um eine klassische Wahrscheinlichkeitsverteilung S über S_n in der eben *z* mit *W(z)* auftreten. Würde man mehrere Messungen machen, so erhält man folgende Verteilung[29]:

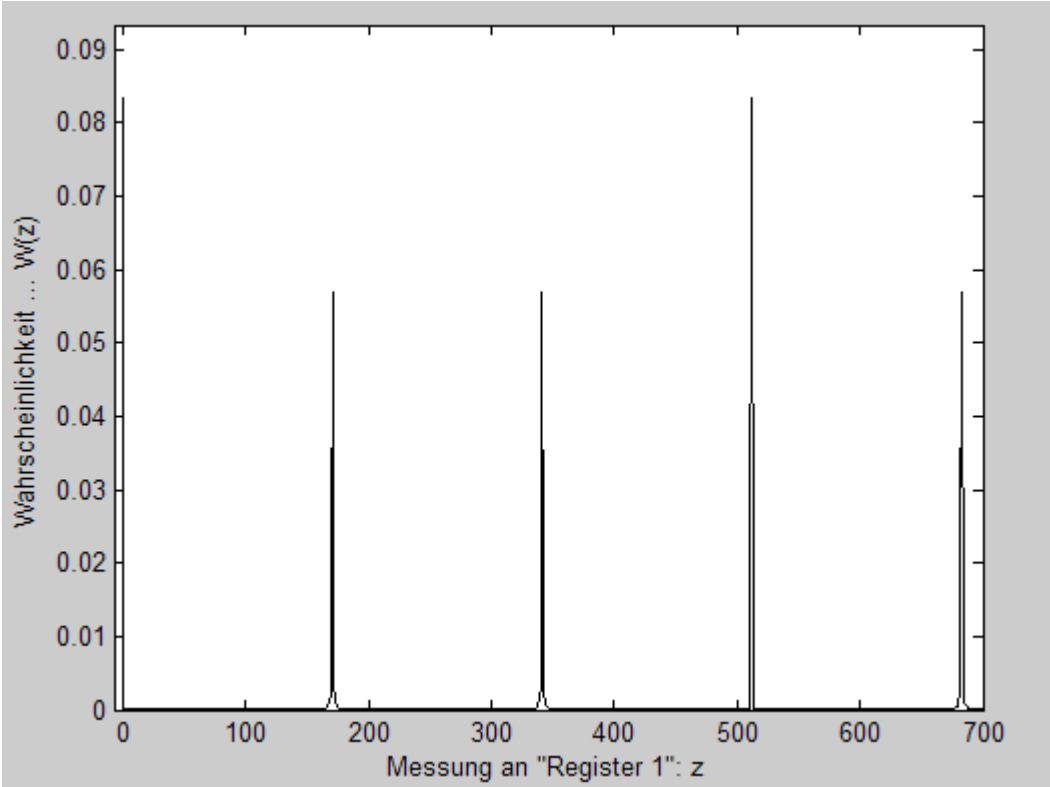

Abbildung 7 - Simulierte Verteilung der Messungen eines Quantencomputers für *Q* = *P* **q* + *r*, *q* = 170, *r* = 8 und *P* = 12.

Einschub : Simulation eines Quantencomputers, Wahrscheinlichkeitsverteilung W(z)

Da man leider noch keinen Quantencomputer zur Verfügung hat, muss man sich mit folgenden Überlegungen die Wahrscheinlichkeitsverteilung von Messungen am Register 1 simulieren (vgl. Abbildung 7):

Sei $Q = 2^n = Pq + r; (0 \leq r < P)$; *q,r* sind natürliche Zahlen und $Q_0 = Pq$.
Dann ergibt sich aus den Definitionen und nach längerer Rechnung[30]:

[29] Siehe Quellcodes
[30] Beweis Quelle1 , Seiten 145,146

$$W(z) = \begin{cases} \dfrac{r\sin^2\left(\frac{\pi Pz}{Q}\left(\frac{Q_0}{P}+1\right)\right) + (P-r)\sin^2\left(\frac{\pi PzQ_0}{Q\ P}\right)}{Q^2\sin^2\left(\frac{\pi Pz}{Q}\right)} & \dots für\ Pz \not\equiv 0\ mod\ Q \\ \dfrac{r(Q_0+P)^2 + (P-r){Q_0}^2}{Q^2P^2} & \dots für\ Pz \equiv 0\ mod\ Q \end{cases}$$

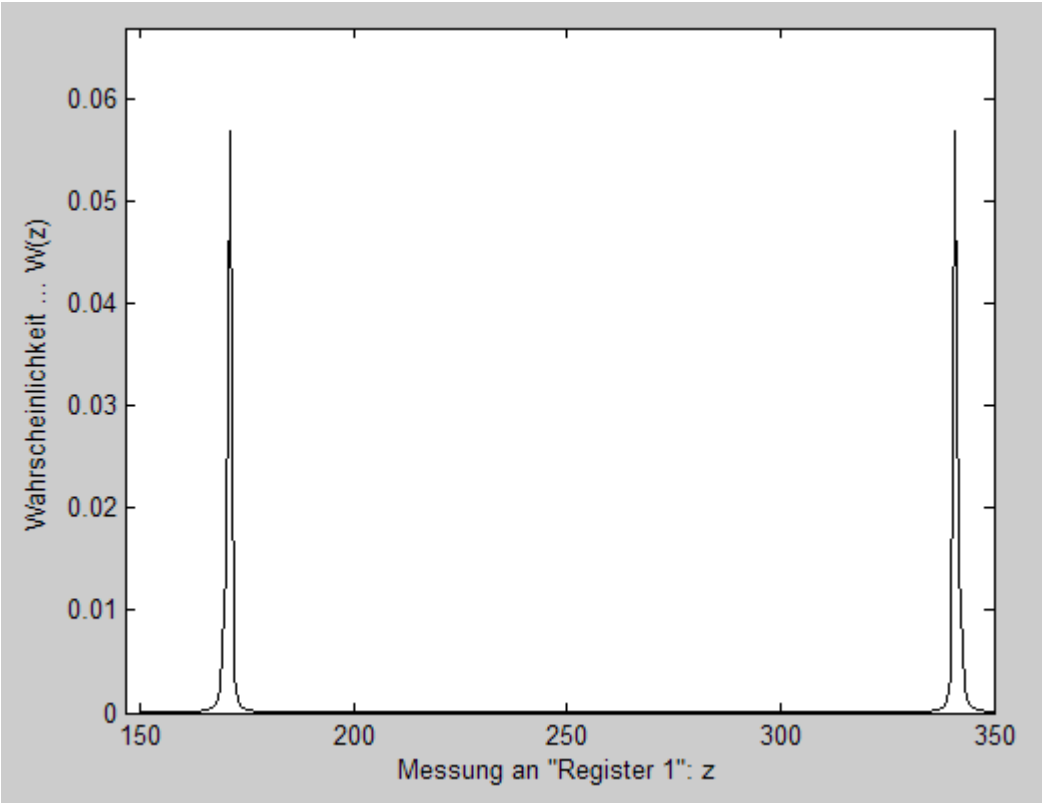

Abbildung 8 - Wahrscheinlichkeitsverteilung für *Q/P* ist nicht ganzzahlig.

Messungen können also auch falsche *z* liefern, allerdings mit einer kleineren Wahscheinlichkeit. Das bedeutet auch eine falsche Distanz *q* oder Periode *P*.

Nimmt man weiters an, dass $Q/P \in \mathbb{Z}$, also $Q_0 = Q$, dann ergibt sich:

$$W(z) = \begin{cases} 0 & \dots für\ Pz \not\equiv 0\ mod\ Q \\ P^{-1} & \dots für\ Pz \equiv 0\ mod\ Q \end{cases}$$

Beweis

$$Für\ Pz \not\equiv 0\ mod\ Q, r = 0, also\ Q = Pq\ ergibt\ sich\ W(z) = \frac{P\sin^2(\pi z)}{Q^2\sin^2\left(\frac{\pi z}{q}\right)} = 0.$$

$$Und\ für\ Pz \equiv 0\ mod\ Q, ergibt\ sich\ W(z) = \frac{0\cdot(Q_0+P)^2+(P-0){Q_0}^2}{Q^2P^2} = \frac{P\,Q^2}{Q^2P^2} = P^{-1}. \#$$

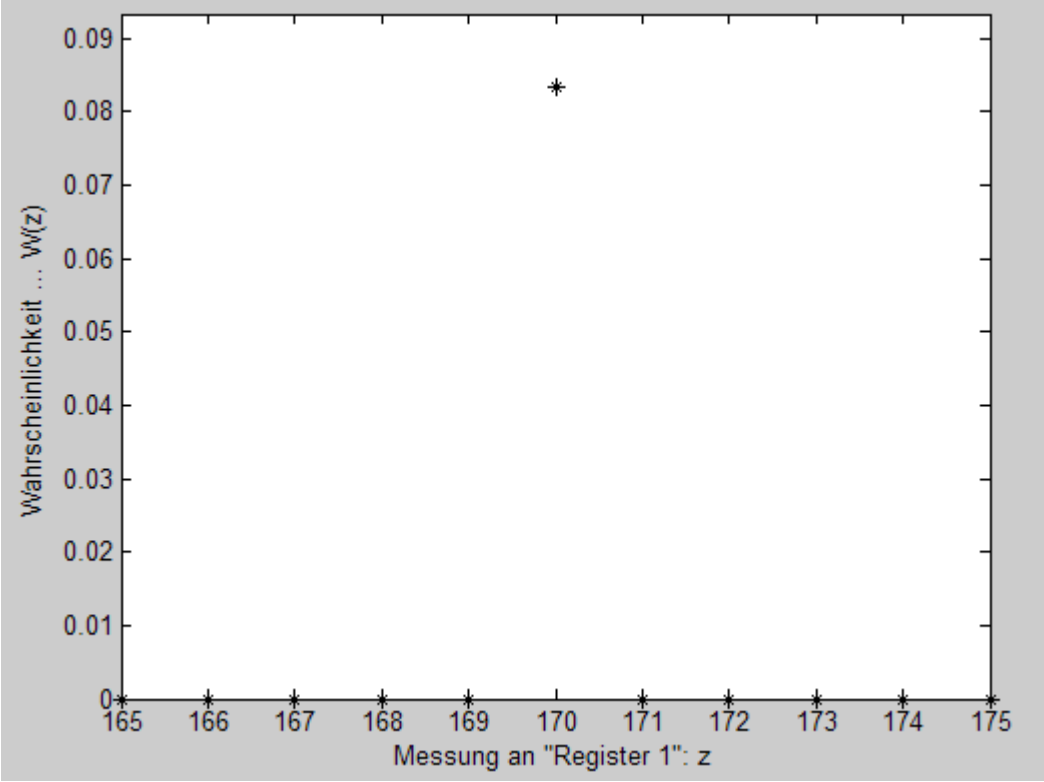

Abbildung 9 - Peak für *Q/P* ϵ $\mathbb{Z}$. Hier gibt es keinen Streubereich, es erfolgt sofort eine brauchbare Messung, aus der man mit CFE ein richtiges *P* folgern kann. Siehe Schritt 2.5c.

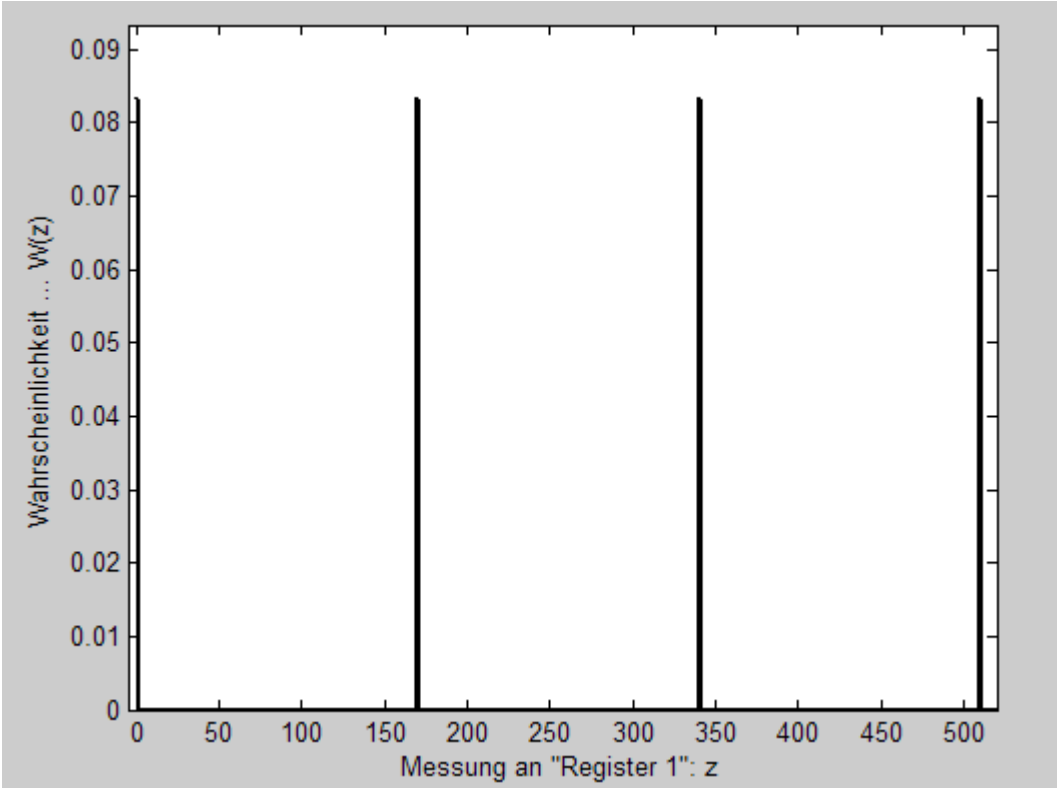

Abbildung 10 - Hier erkennt man *P* Peaks mit der Wahrscheinlichkeit *W(z) = 1/P*.

2.5a ... Indirekte Bestimmung von *P* über die Distanz *q*[31]

Begrenzt man nun $0 \leq z \leq Q - 1$. Und **wiederholt die Messungen**, dann ergibt sich die **minimale Distanz zwischen den Peaks** von ungefär $\acute{q}$. So kann man eine genäherte Periode $\acute{P}$ schätzen:

$$\acute{P} = {}^{Q}/_{\acute{q}} .$$

Man weiß, dass, falls das Verhältnis (*Q* zu eigentlicher Periode) keine natürliche Zahl ist, sich zu einer kleineren Wahrscheinlichkeit falsche Messungen ergeben (vgl. Abbildung 11).
Diese genäherte, probabilistische Periode muss nun auf Plausibilität überprüft werden , indem man Schritt 3 – 5 nochmal ausführt. Für große *N* ist diese Metode allerdings nicht sonderlich geschickt.
Es gibt aber auch eine bessere Methoden die Periode zu finden. In den nächsten Punkten wird eine solche behandelt. Diese ist effizient und benötigt nur **EINE** Messung am Register 1, um eine mögliche Periode *P* zu bestimmen.

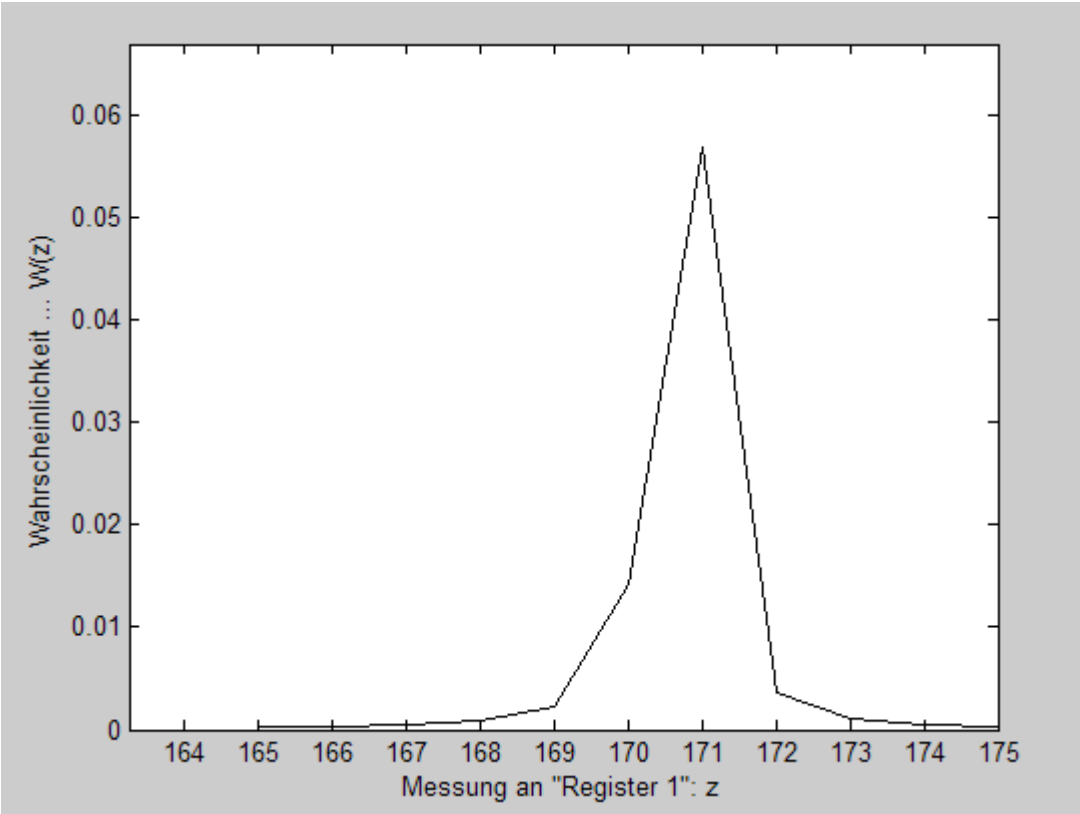

Abbildung 11 - Streubereich $\Delta z \sim \log(Q)$. Die höchste Wahrscheinlichkeit hat aber immer noch der Peak *Q/z*, der die kleinste Differenz mit *P/d* im Bereich bildet, wobei *d* = *zP/Q*.
Paramter dieser Simulation: *Q* = 16384 = *Pq* + *r* = 6*2730 + 4.

[31] Vollständige Beispiele in Quelle 1, Kapitel 8

2.5b ... Bestimmung der Periode *P* mittels wiederholter Einzelmessungen[32]

Diesmal nimmt man nur eine Messung vor und mit einer großen Wahrscheinlichkeit bekommt man $z = (dQ)/P$. Mit d ist eine natürliche Zahl.
Da ja *Q* bekannt ist, kann man daraus *P* berechnen, nur kann es sein, dass dieser Bruch noch reduziert werden kann. Also zum Beispiel *P* = 8, dann kann man Peak 1 und 2 vergleichen. Einmal hat man $z/Q = 1/P$ und einmal $z/Q = 2/P$, so wird man einmal glauben *P* = 8 und einmal *P*=8/2 = 4. Betrachtet man noch die Peaks 3 bis 8, so wird man bei *P* = 8/3, 8/5, 8/7 auf *P* =8 schließen und bei P = 8/4, 8/6,8/8 auf P = 2, 4 (aus 4/3) und 1. So schließt man bei nur einer Messung zu 50 % richtig (dies gilt nur bei Q/P ist eine ganze Zahl).

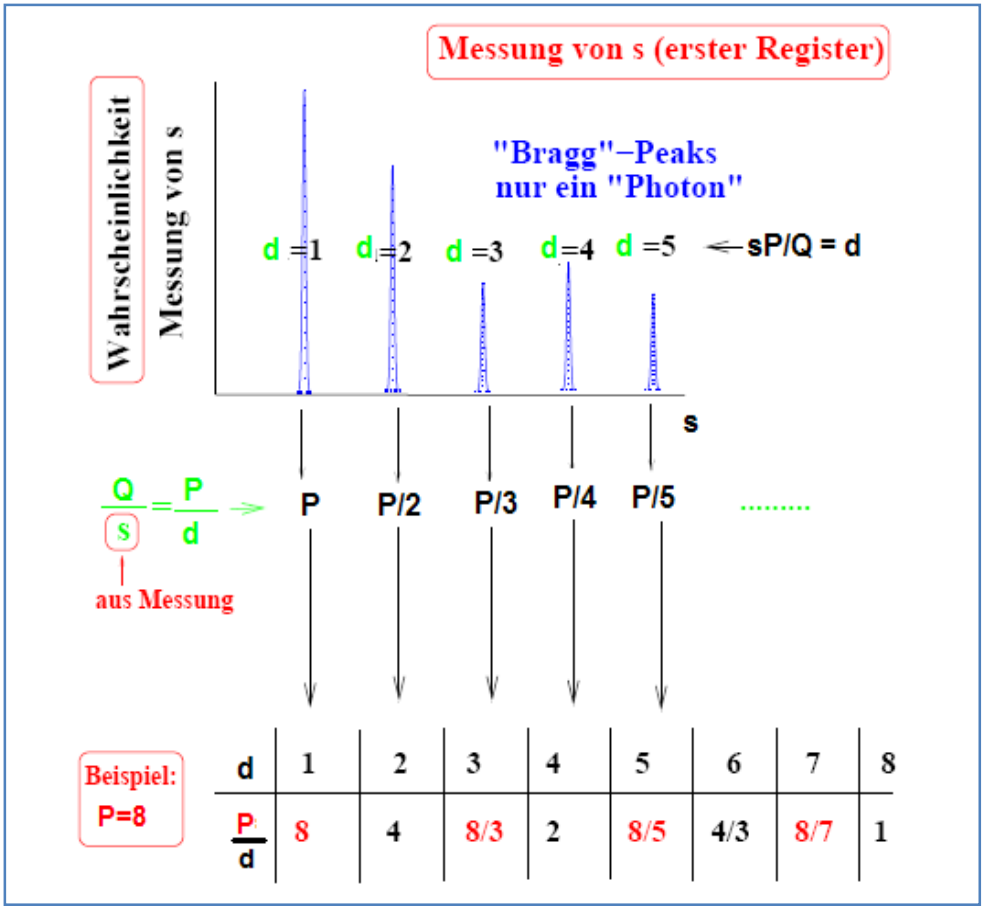

Abbildung 12 – reduzierbare Brüche und die Folgen daraus für die Bestimmung der Periode.

Ist nun *Q/P* nicht ganzzahlig, so misst man immer wieder *z*, die auf ein falsches *P* folgern lassen (siehe Abbildung 11, man bemerke den **Streubereich**).

[32] Aus Quelle 3, Kapitel 7, Seite 80 bis 83

Die Wahrscheinlichkeit für sofortigen Erfolg sei nun ε. Also liegt man (1- ε) daneben.

Die Wahrscheinlichkeit nach ω Wiederholungen noch immer kein richtiges Ergebnis zu haben ist $(1 - \varepsilon)^{\omega}$.

Also ist $P(\omega) = 1 - (1 - \varepsilon)^{\omega}$ die Wahrscheinlichkeit für mindestens eine richtige Messung bei ω Wiederholungen.

Setzt man sich ein Limit für eine Sicherheit von $P_0 = 99\,\%$,

so muss man $\omega = \frac{-\log(1-P_0)}{\log(1-\varepsilon)} \approx \frac{-\log(1-P_0)}{\varepsilon} \sim \frac{1}{\varepsilon}$ Messungen vornehmen.

Man kann zeigen, dass $\frac{1}{\varepsilon}$ immer noch proportional zu Polynom von log(*N*) ist, also bleibt der Algorithmus effizient trotz öfterer Wiederholung[33].

Ist *Q/P* keine ganze Zahl, so kann man zeigen, dass der Streubereich $\Delta z \sim \log(Q)$ ist.
Die Ordnung dieses Streubereichs ändert sich auch nicht, wenn man auf Kosten der Phasengenauigkeit die Effizienz bewahrt. [34]

Nimmt man die benachbarten Werte von der falschen Messung und berechnet daraus *P*, so ändert sich die Komplexitätsklasse dabei nicht und man ist immer noch im effizienten Bereich[35].

2.5c Bestimmung der Periode *P* mit CFE (continued fraction expansion)

Oben wird bereits darauf hingewiesen, dass man sich im effizienten Bereich befindet, selbst wenn man mehrmals wiederholen müsste. Da man jetzt aber nach einer Messung *z/Q* mit CFE die Periode berechnen (es sei denn der Bruch kann noch reduziert werden), und weiters eine Bedingung erhalten kann, mit der man falsche *z* (aus dem Streubereich) filtern kann, so gilt für die neue Anzahl der Wiederholungen $\widetilde{\omega} \leq \omega$. Erfolgt die Realisierung der CFE im polynomialen Bereich, so ist man effizient[36].
Ein paar Wiederholungen:

$$\forall\, x \in \mathbb{R} : \lceil x \rceil = \inf\{\, n \in \mathbb{Z} \mid x \leq n\}$$
$$\forall\, x \in \mathbb{R} : \lfloor x \rfloor = \sup\{\, n \in \mathbb{Z} \mid n \leq x\,\}$$

[33] Siehe Quelle 4, Seite 10; Quelle 5, ab Seite 296
[34] Eigentlich müsste man in der exakten Fouriertransformation noch Phasen kontrollieren, dann geht aber die Effizienz verloren. Man kann aber zeigen, dass es reicht die Phasen mit geringerer Genaugikeit zu kontrollieren. Beweis siehe Quelle 6, vgl. hierzu auch Quelle 1, Kapitel 6, Seite 122 und Quelle 3, Seite 83
[35] Vgl. Quelle 5, ab Seite 296
[36] Vgl. Quelle 4, Seite 12

CFE...continued fraction expansion[37]

$$\forall\, x \in Q: x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ldots + \cfrac{1}{a_q}}}}}$$

Wobei a_i natürliche Zahlen sind. *x* wird auch als $x = [a_0, a_1, \ldots, a_n]$ geschrieben.
Allgemein:

1. *x* sei eine rationale Zahl. $a_0 = \lfloor x \rfloor$ und $r_0 = x$.
2. $a_1 = \left\lfloor \frac{1}{r_0 - a_0} \right\rfloor, r_1 = \frac{1}{r_0 - a_0}$.
3. $a_n = \left\lfloor \frac{1}{r_{n-1} - a_{n-1}} \right\rfloor,\ r_n = \frac{1}{r_{n-1} - a_{n-1}} \rightarrow a_n = \lfloor r_n \rfloor$,
4. $Wiederhole\ bis\ \ Nenner\ von\ r_M = 0, f\ddot{u}r\ x \in \mathbb{R}$ ist M immer endlich.

Begriffe:
Sei $x = [a_0, a_1, \ldots, a_n]$, dann heißt $[a_0, a_1, \ldots, a_n]$ Kettenbruch und a_j die *j*-te Konvergente.

Zurück zum Shor-Algorithmus[38]

Man misst *z* aus Register 1, dann ist *z/Q* eine rationale Zahl, die ungefähr bei *d/P* liegt, *d* ist wieder eine natürliche Zahl. (Es sei denn man misst ein falsches *z*).
So findt man *P* von $f(x) = m^x \mod N$:

1. Finde $[a_0, a_1, \ldots, a_n]$ von *z/Q*. Ist $\frac{z}{Q} < 1$ so folgt: $a_0 = 0$.
2. Wir setzen $a_0 = p_0\ und\ q_0 = 1$.
3. Dann $p_1 = a_1 p_0 + 1$ und $q_1 = a_1 q_0$.
4. $p_i = a_i p_{i-1} + p_{i-2}$ und $q_i = a_i q_{i-1} + q_{i-2},\ (2 \leq i \leq M)$.
5. So erhalten wir Paare (p_i, q_i). Dann ist p_i/q_i die *i*-te Konvergente.
6. Finde das **<u>kleinste</u>** $k\ (0 \leq k \leq M)$ so dass $\left|\frac{p_i}{q_i} - \frac{z}{Q}\right| \leq \frac{1}{2Q}$. *k* ist eindeutig.
7. $p_k = d$ und noch viel wichtiger: $q_k = P\ die\ Periode$!!

(ohne Beweis).

[37] Quelle 1, Seite 152; Quelle 8
[38] Methode, Beweise, Beispiele für Bestimmung der Periode mittels CFE siehe Quelle 1, Seite 151 bis 156

Jetzt muss man sich nur noch vergewissern, ob die Periode auch richtig ist.
Ist *P* richtig, so ist Messung $z \epsilon C$.

$$C = \left\{ z | \exists d \epsilon \{1,2, \dots P-1\}, \left| \frac{d}{P} - \frac{z}{Q} \right| \leq \frac{1}{2Q}, ggT(P,d) = 1 \right\}$$

$$C \neq \{\}$$

Das heißt es existiert immer ein $z \epsilon \{1,2, \dots, Q-1\}$ falls $P < Q$.
Aus $-{}^{P}/_{2} < Q - zP \leq {}^{P}/_{2}$ folgert man:

$$\left| \frac{1}{P} - \frac{z}{Q} \right| \leq \frac{1}{2Q}$$

So hat die Menge *C* zumindest ein Element *z* für das gilt:

$$\frac{d}{P} = \frac{1}{P}$$

(ohne Beweis)

N.B. : Man kann vor der Bestimmung von *P* nicht wissen, ob $z \epsilon C$ ist.

Wiederholung

Mit Schritt 2 wurde Periode *P* **effizient** bestimmt.

Der angewandte Algorithmus (Shor`94) ist ein *P*-Problem, also **effizient** im Gegensatz zu klassischen Algorithmen im 2. Schritt.

Unter Anwendung von Schrit 3 bis 5 kann man *N* faktorisieren und man ist am Ziel der langen Reise angelangt!

Zusätzliche Informationen[39]

Um nun eine Periode P von $f(x) = m^x mod\ N$ zu bestimmen, verwendet man einen Quantencomputer mit $\mathcal{F}$ und U_f, mit dem man diesen Schritt 2 berechnen kann. Die restlichen Schritte erfolgen mit klassischen, effizienten Algorithmen.

1. Ein *n*-qubit QFT circuit wird mit $\boldsymbol{\theta(n^2)}$ elementaren Bausteinen aufgebaut.

2. Die Funktion $f(x) = m^x mod\ N$ wird mit einer Anzahl von Bausteinen **polynomialer** Ordnung gebaut.

3. Wenn man noch die Schritte für die benötigten Wiederholungen und Berechnungen im Schritt 2 beachtet (**polynomial**),

so folgert man dennoch:
Die Primzahlen-Faktorisierung einer Zahl $N \sim 2^n$ erfolgt also mit einer polynomialen Anzahl von Schritten mit dem Quantencomputer.
Wie schon mehrmals erwähnt, benötigt der klassische Computer eine exponentielle Anzahl an Schritten.

Jetzt sind alle 5 Schritte des Shor-Algorithmus komplett und man kann in effizienter Weise große Zahlen faktorisieren, also beliebige RSA-verschlüsselte Wörter decodieren. Mit diesem 5-Schritte-Schema werden nun in den nächsten 2 Kapiteln Beispiele gelöst.

[39] Vgl. Quelle 1, Seite 171 und vorige; siehe auch Kapitel 6, Seite 122

8. Demonstration an einem Beispiel[40]

RSA-Teil

Um zu zeigen, dass das alles hier durchaus Relevanz hat in der realen Welt und nicht nur einen theoretischen Humbug darstellt, wird in diesem Kapitel auf ein Anschaungsbeispiel näher eingegangen.

Man begibt sich in die Rolle eines ‚Kryptoanalytikers', eines Angreifers, der unbedingt eine Nachricht abfangen will, um naja, lassen Sie Ihre Fantasie spielen, z.Bsp. Marschrouten einer Militäroffensive, Passwörter, was auch immer zu erlangen. Es sei *m*(essage) =„TOPSECRET".

Zwei Personen, sie sollen heißen: *A*(lice) und *B*(ob). Diese zwei kommen manchem sicher bekannt vor. ☺

B will *A* eine Nachricht zusenden, die sie mittels dem RSA-Verfahren verschlüsseln um sicherzugehen, dass sonst niemand davon mitbekommt, die Nachricht ist ja immerhin „TOPSECRET".

Also *B* will *A* eine Nachricht schicken. Zuallererst muss die Nachricht mit Hilfe der Quellcodierung (vorzugsweise optimal) codiert werden, um sie überhaupt verschicken zu können. Meist bekommt sie noch Korrekturbits hintenangestellt, weil ja reale Kanäle alles andere als rauschfrei sind.

Man ordnet der Nachricht einfach eine Dezimalzahl zu (vgl. ASCII,...).

B codiert sein Wort in „TOPSECRET" → 12 (soll für diese Zwecke reichen, hier soll ja nur demonstriert werden, wie das alles ungefähr funktionieren soll).

A wird nun ihren öffentlichen Schlüssel an *B* schicken, ihren privaten, zum decodieren der Nachricht, den hält sie geheim. Die muss sie allerdings erst berechnen.

A bereitet nun 2 Primzahlen „ungeheurer Größe" vor:

$$p = 13, \qquad q = 7$$

Deren Produkt *N* gemeinsam mit der Zahl f

$$ggT(\,(p-1)(q-1), f) = ggT(72, f) = \;\; 1$$

den öffentlichen Schlüssel darstellen. Es sei $f = 5$.
Also ist der „extrem sichere" öffentliche Schlüssel $(N, f) = (91, 5)$.

Nun muss *A* noch ihren privaten Schlüssel *d* erzeugen, für den gilt:

$$d \cdot f \equiv 1 \; mod \; (p-1)(q-1)$$

Also die modulare Inverse von $e \; mod \; (p-1)(q-1)$.

[40] Starke Anlehnung an Quelle 1 im Shor-Algorithmus-Teil, ab Seite 141

Es sei $d = \frac{k(p-1)(q-1)+1}{f}$ eine ganze Zahl, k ist eine natürliche Zahl (d wird über den erweiterten euklidschen Algorithmus bestimmt)[41].

Und man bekommt für den privaten Schlüssel ebenfalls $d = 29$.

B verschlüsselt seine Nachricht $b = 12$ zu Geheimtext *c*.

$$c = b^f mod\ N = 12^5 mod\ 91 = 38$$

A empfängt nun (einen rauschfreien Kanal, oder richtige Kanalkodierung vorausgesetzt) *c* = 38.
Sie wendet ihren privaten Schlüssel $d = 29$ an um *c* wie folgt zu decodieren:

$$b = c^d mod\ N = 38^{29} mod 91 = 12$$

Und erhält die richtige Botschaft *b* = 12 = „TOPSECRET".

Nun kommt der Kryptoanalytiker ins Spiel, man will diese „ungeheuer große Zahl" N=91 faktorisieren.
Warum eigentlich? – Weiß man p und q so erhält man automatisch φ und daraus d.
Man erhält automatisch den privaten Schlüssel *d*, mit dem jedes Codewort dekodiert werden kann.

Shor-Teil

Für die Kryptoanalyse folgt man den 5 Schritten(Shor):

Schritt 1

Man wählt ein zufälliges *m* = 3. ggT(3,91)=1, also zu Schritt 2.

Schritt2

$N^2 \leq 2^n \leq 2N^2;\ 91^2 \leq 2^{14} \leq 2 \cdot 91^2;\ 8281 \leq (Q = 16384) \leq 16562$

2.0

$$|\psi_0\rangle = |REG1\rangle\,|REG2\rangle = |00\ldots0\rangle\,|00\ldots0\rangle$$

2.1 $\mathcal{F}$

$$|\psi_1\rangle = \frac{1}{\sqrt{16384}} \sum_{x=0}^{16384-1} |x\rangle\,|0\rangle$$

[41] Siehe Quellcodes und Kapitel 2 dieser Arbeit, weiters Quelle 5, Seite 311

2.2 U_f

$$U_f|\psi_1\rangle = |\psi_2\rangle = \frac{1}{\sqrt{16384}} \sum_{x=0}^{16384-1} |x\rangle\,|f(x)\rangle$$
$$= \frac{1}{\sqrt{16384}} (|0\rangle\,|1\rangle + |1\rangle\,|3\rangle + |2\rangle\,|9\rangle + \dots)$$

$f(x) = 3^x mod\ 91$. Hier ergeben sich die Zustände mit nur einem Schritt, zum Vergleich die Werte im klassischen System, diese hat allerdings benötigen wir nur zur Veranschaulichung.

f(x)	1	3	9	27	81	61	1	3
x	0	1	2	3	4	5	6	7

Bei späterer Analyse wird einem auffallen, dass nur *P* verschiedene Zustände existieren.

2.3 Nochmaliges Anwenden von $\mathcal{F}$ am ersten Register

$$|\psi_3\rangle = \mathcal{F}\otimes\mathbb{I}\,|\psi_2\rangle = \frac{1}{16384} \sum_{x=0}^{16383} \sum_{z=0}^{16383} \omega_n{}^{-zx}\,|z\rangle\,|3^x mod\ 91\rangle = \frac{1}{216384} \sum_{z=0}^{16383} |z\rangle\,|g(z)\rangle$$

$$= \frac{1}{16384} \sum_{z=0}^{16383} \||g(z)\rangle\|\,|z\rangle\ \frac{|g(z)\rangle}{\||g(z)\rangle\|}$$

Wobei:

$$|g(z)\rangle = \sum_{x=0}^{16383} e^{-\frac{2\pi i z x}{16384}} |3^x mod\ 91\rangle$$
$$= (|1\rangle + \omega_n{}^{-z}|3\rangle + \omega_n{}^{-2z}\,|9\rangle + \dots + \omega_n{}^{-16383z}|letzter\ Wert\rangle) =$$
$$(1 + \omega_n{}^{-Pz} + \omega_n{}^{-2Pz} + \cdots)|1\rangle +$$
$$\left(\omega_n{}^{-z} + \omega_n{}^{-(P+1)z} + \omega_n{}^{-(2P+1)z} + \cdots\right)|3\rangle +$$
$$\left(\omega_n{}^{-2z} + \omega_n{}^{-(P+2)z} + \omega_n{}^{-(2P+2)z} + \cdots\right)|9\rangle +$$
$$\dots$$
$$(\qquad\qquad \dots + \omega_n{}^{-16383z})\,|letzter\ Wert\rangle\,.$$

Hier gibt es *P* Ket-Vektoren. Die zusammengefassten Koeffizient des Vektors werden messbar, wenn *z* ein näherungsweise Vielfaches von *q* ist (deswegen, weil sich in der Nähe unseres höchsten Peaks kleinere Peaks befinden, die auch noch messbar sind, vergleiche Abbildung 11), sie interferieren konstruktiv.

$$\sum_{k=0}^{q} \omega_n{}^{-Pkz\,(=h*q)} = c_1 \in \mathbb{C};\ \left|\sum_{k=0}^{q} \omega_n{}^{-Pk(z\,=h*q)}\right| = r_1 \in \mathbb{R}; z_1\ ist\ \sim Vielfaches\ von\ q$$

$$\sum_{k=0}^{q} \omega_n{}^{-Pk(z\,\neq h*q)} = c_2 \in \mathbb{C};\ \left|\sum_{k=0}^{q} \omega_n{}^{-Pk(z\,\neq h*q)}\right| = r_2 \in \mathbb{R}; z_2\ ist\ \neq Vielfaches\ von\ q$$

$$r_1 \gg r_2 \;\rightarrow W(z_1) \gg W(z_2)\,;\; \frac{P \cdot {r_1}^2}{Q^2} \gg \frac{P \cdot {r_2}^2}{Q^2}$$

Der Faktor *P* entsteht, weil der Vektor $|g(z)\rangle$ aus einer Überlagerung von *P* Vektoren besteht, deren Koeffizienten bei einem *z* von $d \cdot \frac{Q}{P} = d \cdot q$ und $d \in \mathbb{N}$ konstruktiv interferieren. Also jeder Ket-Vektor hat einen Koeffizienten mit r_1, aufgrund der periodischen Eigenschaften der Koeffizienten.

2.4 Messung

Durch Messung an $|REG1\rangle$ erhält man $z \in S_n$ mit der Wahrscheinlichkeit:

$$W(z) = \frac{\||g(z)\rangle\|^2}{Q^2}.$$

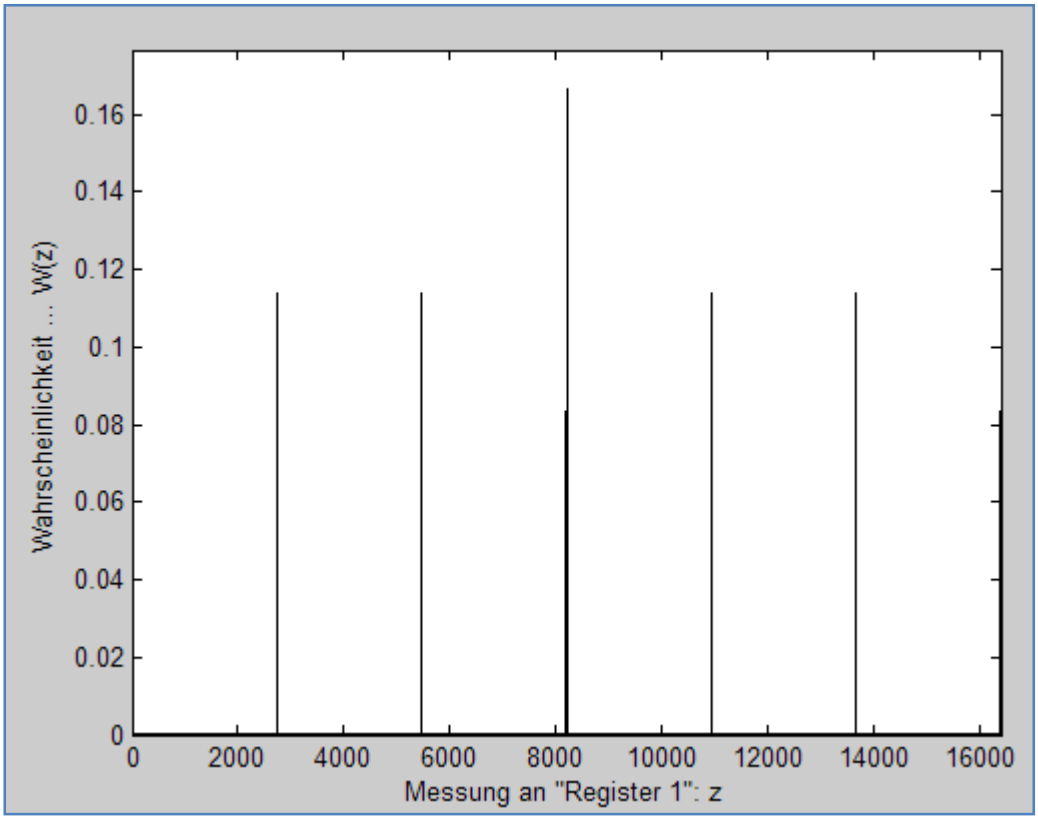

Abbildung 13 - *W(z)* ... Wahrscheinlichkeitsverteilung in Abhängigkeit von *z*

Nun wertet man die Verteilung der Simulation aus. Das wäre bei einem Quantencomputer ebenfalls ineffizient. Denn wollte man eine Wahrscheinlichkeitsverteilung haben, müsste man eine Vielzahl von Messungen durchführen. Man muss immer wieder mit dem Messprozess von Anfang an beginnen, weil ja die verschränkten Zustände mit jeder Messung zerstört werden.

Hier werden dieselben Werte wie für die Simulation für Abbildung 13 verwendet.
So erhält man bei dem Fall *Q/P* ist eine natürliche Zahl die Wahrscheinlichkeit *W* = $0.1\dot{6} \sim \frac{1}{P}$.
Das wiederum führte zu einer Periode *P* = 6, die ja in der Simulation als Inputparameter vorgegeben war.
Bei dem Fall wie in Abbildung 14, also *Q/P* ist keine natürliche Zahl, erhalten wir *P'* = 12.0015 = *Q/q* wobei $Q \neq Q_0 = P \cdot q$, weil ja $r = 4 \neq 0$.
$Q(= 16384) \equiv r \; mod \; P = 4 \; mod \; 6; \quad Q_0 = Q - r = 16384; \quad q = \frac{Q_0}{P} = 2730$.
q = 2730 stimmt ungefähr mit der Abbildung 13 überein. $P' \cong {}^{Q}/_{q} = 6.0015$.

Nun vergleicht man die durch die Simulation erstellte Periode mit der durch einen klassischen Algorithmus bestimmte Periode:

!!![Achtung, beide hier verwendete Methoden, die Simulation (analog mehrere Messungen) und der klassische Algorithmus, sind nicht Bestandteil des effizienten Shor-Algorithmus, sie dienen lediglich der Veranschaulichung und sind bei höheren Zahlen ohnehin nicht mehr durchfürbar.]!!!

f(x)	**1**	3	9	27	81	61	**1**	3	9	27	81	61	**1**	3
x	**0**	1	2	3	4	5	**6**	7	8	9	10	11	**12**	13

Wunderbar, stimmt überein. Man kann also davon ausgehen, dass ein Quantencomputer richtige Ergebnisse liefert, man muss aber geschickt vorgehen, um effektiv zu bleiben.

Nun aber zur Methode der Bestimmung von *P* mittels CFE.[42]
Hier misst man *z* nur einmal und schaut dann, ob die errechnete Periode richtig ist, es kann allerdings sein, dass sich ein *z* aus dem Fehlerbereich $\Delta z \sim \log(Q)$ (vgl. Abbildung 14) eingeschlichen hat. Oder der Bruch kann sich weiter reduzieren.

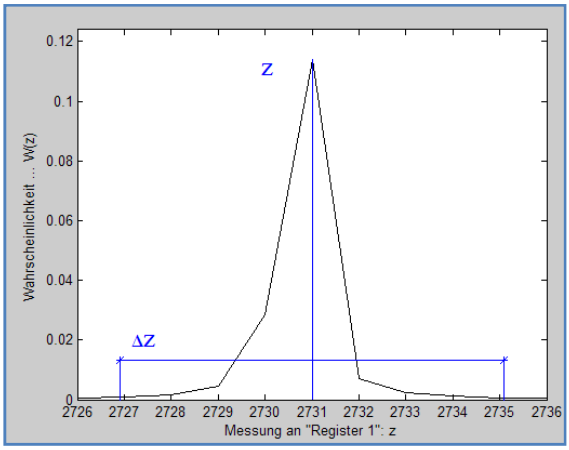

Abbildung 14 – Streubereich Δ *z* ~ log(Q), Simulationsinputparameter aus Beispiel oben.

[42] Siehe Kapitel 7, 2.5c

Mit Schritt 2.5c

Fall 1...
Angenommen, man hätte einen Quantencomputer und man messe nun *z* = 2730
(vgl. Abbildung 14, und beachte Fehlerbereich Δz).
Das liefert mit CFE: *z/Q* = [0,6,682,1,1].

Und die Paare: $\{(p_0, q_0), (p_1, q_1), (p_2, q_2), (p_3, q_3), (p_4, q_4)\}$ =
$\{(0,1), (1,6), (682,4093), (683,4099), (1365,8192)\}$.

Die ergeben sich zu den Brüchen $\{\dots, {p_i}/{q_i}, \dots\} = \left\{0, \frac{1}{6}, \frac{682}{4093}, \frac{683}{4099}, \frac{1365}{8192}\right\}$.

Vergleiche nun ob die Bedingung $\left|\frac{d}{P} - \frac{z}{Q}\right| \leq \frac{1}{2Q}$ erfüllt ist.

Für die Brüche ergibt der logische Vektor *logivec*(i = 0 bis 4)[43] = (0,0,1,1,1).
Man erhält *P* = 4093. Prüfe ggT(*d,P*) = ggT(682,4093) =1. Ok. Dieses *P* führt allerdings bei Schritt 3 und übrigens auch bei vier zur Rückkehr zu Schritt 1. Man sucht ja die kleinste Periode, also widerspricht *P* = 4093, dass
$m^P - 1 \equiv 0\ mod\ N, aber\ kein\ Vielfaches\ von\ N.$
Abgesehen davon ist sie ungerade. Man nun zu Schritt 1 zurückkehren oder man probiert unter Berücksichtigung des Streubereichs ein anderes z, z.B.:

Fall 2...
Mit einer ziemlich hohen Wahrscheinlichkeit erhält man *z* = 2731(Peakspitze).
z/Q = [0, 5,1].

Das bringt wieder:
$\{(p_0, q_0), (p_1, q_1), (p_2, q_2)\}$ = $\{(0,1), (1,5), (1,6)\}$
Der logische Vektor $logivec_{(0\ bis\ 2)}$ bringt (0, 0, 1) und siehe da,
das liefert ein Ergebnis mit *i* = 2.

Und weiters folgert man *P* = 6 aus $(p_2, q_2) = (1,6)$. Check: ggT(1,6) = 1. Und *d* = 1.

Schritt 3

P=6, *P* ist gerade, weiter zu Schritt 4.

Schritt4

Falls $\left(m^{\frac{P}{2}} + 1\right) \not\equiv 0\ mod\ N$, dann weiter mit Schritt 5.

$$\left(m^{\frac{P}{2}} + 1\right) = 3^3 + 1 = 28 \not\equiv 0\ mod\ 91$$

[43] Siehe Quellcodes, -CFE-

Schritt 5

Die Zahl $k = ggT\left(\left(m^{\frac{P}{2}} - 1\right), N\right)$ ist nun entweder *p* oder *q*.

$$k = ggT(26{,}91) = 13 = q, und\ daraus\ folgt:\ \ N = 91 = 7 \cdot 13$$

Jetzt ist man in der Lage, den privaten Schlüssel von Alice zu knacken.

$$d \cdot f \equiv 1\ mod\ (p-1)(q-1)$$

Die einzigen unbekannten waren ja p, q, weil f ist ja mit N Bestandteil des öffentlich zugänglichen Schlüssels. Mit Hilfe des erweiterten Euklidschen Algorithmus bestimmt man d = 5.

Man decodiert das abgefangene Wort $c = 38$ mittels: $c^d mod\ N = 38^{29} mod\ 91 = 12$ = *b* wieder zu dem „TOPSECRET“ = 12. Das mittlerweile gar nicht mehr so streng geheim ist.

Wie man sieht, kann man auf diese Art und Weise ein, klassisch gesehen, unberechenbares Problem mit einem Quantencomputer knacken. Es sei nochmals darauf hingewiesen, dass das RSA-Verfahren im Alltag oft Anwendung findet (Banken, Internet, ... es ist zu beachten, dass in der Anwendung GRÖßERE Zahlen verwendet werden), werden nun Quantencomputer realisiert, so ist dieses Verfahren nicht mehr sicher, auch nicht mit größeren Zahlen.
Aus diesem Grund entwickelt man parallel neben den verschiedenen Realisierungen des Quantencomputers auch spezielle Quantenverfahren (**Quantenkryptographie**).

9. Beispiele mit größeren Zahlen

Bsp 1-RSA

Anno 2125, der Quantencomputer ist längst erfunden.
Message *b* sei *b* = 5123, man nehme diese Zahl bedeutet im TU-Graz-Code den Satz: „Juhu, fertig mit der Bakk.-Arbeit!!“, Manuel will aber nicht, dass der Professor schon davon erfährt und will sie seinem Freund Lukas schicken. Der Professor ist extrem neugierig und lauscht ständig am Kanal nach Botschaften, um immer über den Zustand seiner Studenten im Klaren zu sein. Manuel verwendet also mit Lukas das RSA-Verfahren.

Lukas bereitet nun 2 Primzahlen vor:

$$p = 101, \qquad q = 137$$

Deren Produkt *N* gemeinsam mit der Zahl *f*

$$ggT(\,(p-1)(q-1), f) = ggT(13600, f) = \ 1$$

den öffentlichen Schlüssel darstellen.

Lukas findet ein passendes $f = 9753$.

Der öffentliche Schlüssel von Lukas ist also $(N, f) = (13837, 9753)$.

Diesen kennt der Professor als User des TU-Graz-Pseudo-Nets natürlich.

Lukas muss noch seinen privaten Schlüssel *d* erzeugen, für den gilt:

$$d \cdot f \equiv 1 \ mod\ (p-1)(q-1)$$

Also die modulare Inverse von $e\ mod\ (p-1)(q-1)$.

Lukas bestimmt *d* durch den erweiterten euklidschen Algorithmus[44]: *d* = 4617.

Durch die Verschlüsselung wird jetzt *b*, der Message , die Manuel schickt, durch:

$$b^f mod\ N = \ 5123^{9753} mod\ 13837 = 8317$$

zu dem verschlüsselten Codewort *c* = 8317 = „Juhu, Osterferien!“.

[44] Quellcodes, privatekey()

Für hohe Zahlen ist in Matlab der Befehl *mod*() nicht mehr zu gebrauchen, aber man kann mit **Restklassenmultiplikation** die Zahlen verkleinern[45].

Sollte dies noch nicht ausreichen, kann man die Basis als Dualzahl darstellen, oder das N vereinfachen mit dem chinesischen Restsatz, oder mit dem Satz von Lagrange die Potenz, hier f, vereinfachen[46].

Es sind aber noch locker ein halbes Jahr bis zu den nächsten Osterferien. Der Herr Professor kratzt sich natürlich am Kopf und fragt sich, ob Manuel verrückt sei. Doch dann fällt ihm ein, dass er das RSA-Verfahren in der Vorlesung durchgenommen hat. Und versucht sein Glück mit dem Shor-Algorithmus (siehe Bsp 1-Shor).

Der Herr Professor hat Glück, denn Manuel und Lukas haben tatsächlich das RSA-Verfahren verwendet und sein Shor-Algorithmus liefert ihm p = 101 und q = 137.

Damit bestimmt er sich mit dem erweiterten euklidschen Algorithmus (der als Inputparamter f,p,q benötigt) den privaten Schlüssel d = 4617.
Er wendet nun privaten Schlüssel $d = 4617$ an um c wie folgt zu decodieren[47]:

$$b = c^d mod\ N = 8317^{4617} mod 13837 = 5123$$

Und erhält die richtige Botschaft b = 5123 = „Juhu, fertig mit der Bakk.-Arbeit!!“.

Der Professor freut sich natürlich über diese Neuigkeit und Manuel ist ganz verwundert am nächsten Tag in der Vorlesung, woher der Herr Professor das nun weiß. Doch im Laufe der Vorlesung wird im so einiges klar:

Bsp 1-Shor

Erstellt mit function *example*(), siehe Quellcodes: q_N_m_P = [101, 13837, 100, 4].

N = 13837 , dies ist die Zahl, die der Herr Professor faktorisieren musste, um Manuels Botschaft richtig lesen zu können. Das zeigt er natürlich seinem wissbegierigen Studenten.

Um diese Zahl zu faktorisieren, geht er strikt nach einem bestimmten Schema vor.

Schritt 1

Eine natürliche Zahl m = 100 < N wird zufällig gewählt.
Berechne ggT(m,N). ..Euklidscher Algorithmus vgl. Kap. 2, oder *gcd*() in Matlab ☺.
ggT(100, 13837) = 1.

[45] Siehe function high() im Anhang bei den Quellcodes
[46] Als Anregungen siehe: http://de.wikipedia.org/wiki/Chinesischer_Restsatz und http://de.wikipedia.org/wiki/Satz_von_Lagrange
[47] Siehe function high()

Schritt 2

$N^2 \leq 2^n \leq 2N^2;\ 191462569 \leq 2^{28} = 268435456 \leq 382925138;$

Man benötigt *n*(=28)-qubit-Register.

Es ergibt sich bei der Messung *z* = 67108864.
z'/Q = [0,4], also *z/Q* = 4 (Delta-Peak).

(Andere mögliche Messung, welche letztendlich wieder zu Schritt 1 führt:
Es ergibt sich bei der ersten Messung *z* = 33554432.
z/Q = [0,2], also *z/Q* = 0.5, woraus man die Periode *P* = 2 folgert.
Am Schluss führt das zu einer Faktorisierung *N*=*N*.)

In diesem Fall gibt es keinen Streubereich, weil hier der Delta-Peak-Fall eingetreten ist.

Schritt 3

P = 4 ... gerade

Schritt 4

$\mathrm{mod}(10001, 13837) = 10001 \neq 0$

Schritt 5

$$\text{Die Zahl } k = \mathrm{ggT}(9999, 13837) = p = 101.$$

$$\frac{N}{p} = \frac{13837}{101} = 137 = q.$$

$$N = p \cdot q = 101 \cdot 137.\ Fertig.$$

Bsp 2-Shor (ohne RSA-Teil)

N = 53467. Selbses Schema, man sucht wieder *p* und *q*.

Schritt 1

Eine natürliche Zahl *m* = 20 < *N* wird zufällig gewählt.
Berechne ggT(*m,N*). ..Euklidscher Algorithmus vgl. Kap. 2, oder *gcd*() in Matlab ☺.
ggT(20,53467) = 1.

Schritt 2

$N^2 \leq\ 2^n \leq 2N^2;\ 2858720089 \leq\ 2^{32} (= Q = 4294967296) \leq 5717440178;$

Das heißt man benötigt *n*(= 32)-qubit-Register.
Man überlagert die Anfangszustände, führt die unitäre Tranfsormation für eine periodische Funktion durch, abschließend führt man nochmals eine Fouriertransformation an einem Quantencomputer durch, um Interferenzen geschickt zu nutzen. Man misst ein *z* und mit bekannten *Q* mussen man nun *z /Q* auswerten.

Ein Fall für reduzierbare Brüche wäre:
z = 2 · 715827883-1. Mit *P*=3 (ungerade Periode). *logivec* = (0,1,nicht bestimmt)

z = 715827882. Ist aus dem Streubereich und man folgert ein falsches *P*
(viel zu groß für die Bedingungen der 5 Schritte).

Man wählt einen seiner Nachbarn:
z =715827883. *z/Q* = [0,5,1]. Und folgert mit *logivec* = (0,0,1) die Periode *P* = 6 aus {(0,1),(1,5),(1,6)}. ggT(1,6)=1.

In den weiteren Schritten erweist sich *P* = 6 als richtig.

Schritt 3

P = 6 ... gerade

Schritt 4

$\mathrm{mod}(20^3 + 1, 53467) = 8001\ \neq 0$

Schritt 5

$$\text{Die Zahl } k = ggT(20^6 - 1{,}53467) = p = 421.$$

$$\frac{N}{p} = \frac{53467}{421} = 127 = q.$$

$$N = p \cdot q = 421 \cdot 127 \; Fertig.$$

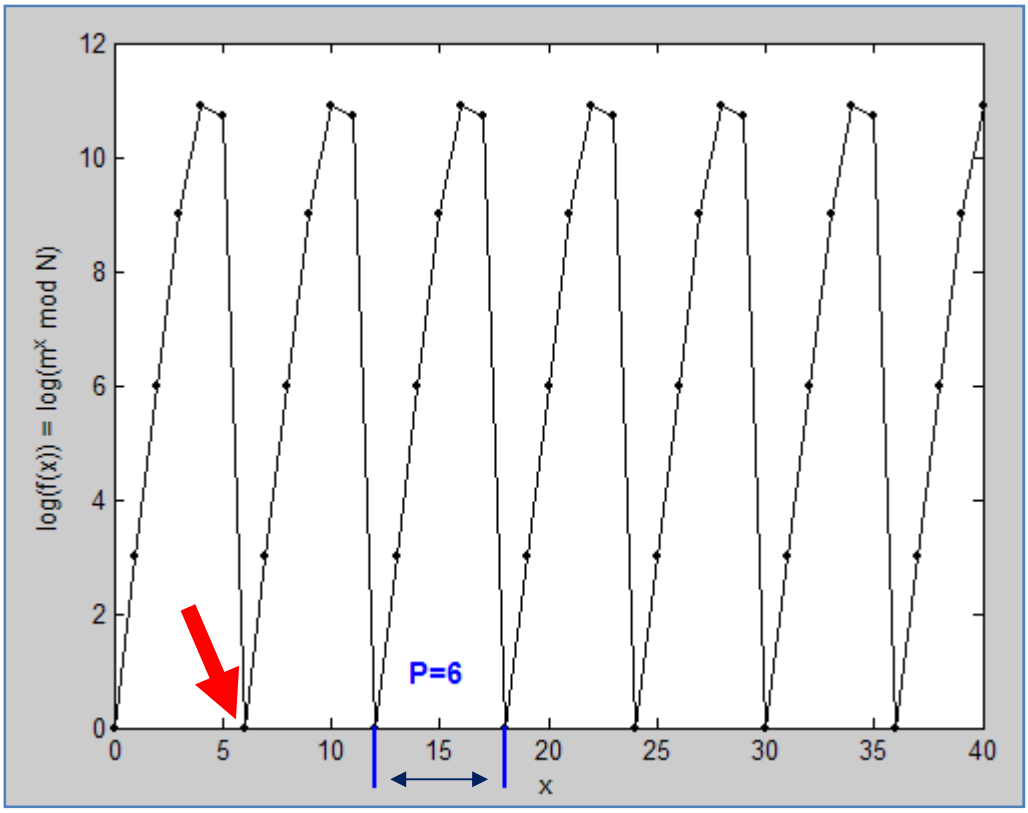

Abbildung 15 – logarithmisches Verhalten der periodischen Funktion *f(x)*. Mit *m* = 20, *N* = 53467, *P* = 6.

Hier erkennt man das kleinste *x* = 6 = *P* für das gilt $m^P \equiv 1 \; mod \; N$.
[Der Wert *log(f(x))* ergibt 0 in diesem Diagramm, d.h.: log(1) = 0. Daraus folgt *f(x)* = 1.]

Abschließend sei nochmal wiederholt:

Es wurde an einem Beispiel die Überlegenheit des Quantencomputers demonstriert, nämlich am Beispiel des RSA-Verschlüsselungsverfahrens. Dieses wurde mit dem Shor-Algorithmus geknackt, indem eine große Zahl faktorisiert wurde. Dies wiederum konnte, wie gezeigt nur mit einem Quantencomputer effizient durchgeführt werden, weil das Finden einer Periode einer periodischen Funktion mit einem klassischen System nicht effizient möglich ist. Ergänzend wurden auszugsweise Grundlagen der Themen diskrete Mathematik, Kryptographie, Quantencomputer, Quantenparallelismus, Quantenfouriertransformation erarbeitet. Das Verhalten eines Quantencomputers wurde simuliert und abschließend noch zwei weitere Beispiel angefügt. Die meisten Grafiken wurden mit Matlab erstellt, die Quellcodes befinden sich im Anhang.

10. Anhang- Verzeichnisse, Quellcodes

Quellenverzeichnis

Die Ausführungen folgen größtenteils diesen Quellen:

1. Quantum Computing-From Linear Algebra to Physical Realization
   von Mikio Nakahara und Tetsuo Ohmi

2. Diskrete Mathematik für Einsteiger
   von Albrecht Beutelspacher und Marc-Alexander Zschiegner

3. Skriptum, TUGraz, ITP, WS 2006/2007:
   Quantencomputer-Eine Einführung
   von Enrico Arrigoni, Roland Peckl, Stefan Rossegger

4. http://xxx.uni-augsburg.de/abs/quant-ph/0005003
   Introduction to quantum algorithms
   von Peter W. Shor

5. Vorlesungsskript über Quantum Computation, John Preskill:
   http://www.theory.caltech.edu/people/preskill/ph229.

6. A. Barenco, Contemp. Phys. 37, 375 (1996).

7. Quantum Computing-Explained von David Mc Mahon

8. Weisstein, Eric W. "Continued Fraction."
   From *MathWorld*--A Wolfram Web Resource.
   http://mathworld.wolfram.com/ContinuedFraction.html

9. http://xxx.lanl.gov/abs/quant-ph/9708016

Abbildungsverzeichnis

Abbildung 1-3 aus Quelle 1
Abbildung 4, 5, 6, 12 aus Quelle 3
Restliche Abbildungen wurden mit Matlab erstellt.

Quellcodes-Matlab

Nützliche Befehle in Matlab:
isprime(), gcd(), mod(),floor(), ... siehe help/doc

-Für Wahrscheinlichkeitsverteilung -
Simulation des Verhaltens eines Quantencomputers

```
function [P,max_W] = probability(P,z_min,z_max,r,q,fall)
%Zur Simulation eines Quantencomputers
%P...Periode
%Q = P*q + r;
%Fenster z_min bis z_max
%wenn fall == 1 wird im plot line verwendet

z = 0 :z_max;
W= z;
Q = P*q + r;
Q_0= P*q;

aa=(pi*P*z/Q);
bb=((Q_0/P)+1);
cc= sin(aa*bb);
dd= (pi*P*z/Q);
ee = (Q_0/P);
ff = sin(dd*ee);
gg = (pi*P*z/Q);
W =[r*cc.^2+ (P-r)*ff.^2]./[(Q^2)*sin(gg).^2];

for i = 0 : z_max
    if i == 0 & (~mod(i*P,Q))
    W(1) = [r*(Q_0+P)^2 + (P-r)*Q_0^2]/[(Q^2)*(P^2)];
    end
    if ~mod(i*P,Q)
    W(i+1) = [r*(Q_0+P)^2 + (P-r)*Q_0^2]/[(Q^2)*(P^2)];
    end
end

P=max(W)^(-1);
max_W = max(W);
W=W([z_min + 1 :z_max + 1]);
if r == 0
    if fall == 1
       plot([z_min:z_max],W,'k-','LineWidth',2)
    else
       plot([z_min:z_max],W,'k.')
    end
else
    if fall == 1
       plot([z_min:z_max],W,'k-','LineWidth',1)
    else
    plot([z_min:z_max],W,'k.')
    end
end
axis([z_min z_max 0 max(W)+.01])
xlabel('Messung an "Register 1": z')
ylabel('Wahrscheinlichkeit ... W(z)')
```

-klassisches Periodenfinden -

```
function out=periodfinding_classical(m,N)
out = zeros(1,N);
%Q soll zwischen N² und 2N² liegen

x = [0:N];
fun = mod(m.^x,N);
for P= 1 : N
    if fun(P+1) == fun(1)
        out=P;
        break
    end
end
```

-Euklidscher Algorithmus- , vgl. Quelle 5, Seite 311

```
function out_ggT_privateKey_phiStrich = privatekey(f,p,q)
%clear all
%f = 5;
%phi = 24;
%p = 5;
%q = 7;
phi=(p-1)*(q-1);
if phi > f
    a = phi;
    b = f;
    c= 2;
    d = 3;

else
    a = f;
    b = phi;
    c= 3;
    d = 2;

end

M=[a,1,0;b,0,1];


while ~(M(2,1)==0)

    g=floor(M(1,1)/M(2,1));
    save = M(2,:);
    M(2,:)= M(1,:)-M(2,:)*g;
    M(1,:)= save;
end
ggT = M(1,1);%sollte 1 sein weil ja f und phi laut Voraussetzung teilerfremd
d=M(1,d);% der private Schlüssel
phi_strich = M(1,c); % eigentlich uninteressant...

out_ggT_privateKey_phiStrich = [ggT,d,phi_strich];
```

-example() -

```
function example(m_min, m_max,P_min,P_max,N_min,N_max)

for N = N_min:N_max
  for P = P_min:2: P_max
    for m = m_min:m_max
      if (m^(P/2)-1 < N)
        if high(m,P,N)==1
          if ~(mod(m^(P/2)+1,N)==0)
            a=gcd(m^(P/2)-1,N);
              if isprime(N/a)
                if isprime(a)
                  if~(N/a==1)
                    if~(N==N/a)

                      out=[m,P,N]
                    end
                  end
                end
              end
          end
        end
      end
    end
  end
end
```

-CFE -

```
clear all;
z = 65536;
Q = 1048576;
M = 7;%>3

a(1) = floor(z/Q);
r(1) = z/Q - a(1);
p(1) = a(1);
q(1) = 1;
r(2) = 1/(r(1) - a(1));
a(2) = floor(r(2));
p(2) = a(2)*p(1) +1;
q(2) = a(2) * q(1);


for sum = 3: M
    r(sum)= 1/(r(sum-1) - a(sum-1));
    a(sum)= floor(r(sum));
    p(sum)= a(sum) * p(sum-1) + p(sum-2);
    q(sum) = a(sum)*q(sum-1) + q(sum-2);
end


bruch = p./q;
diff = abs(bruch-z/Q);
kleiner_als = ones(1,length(diff))*1/2/Q;
logivec= diff <= kleiner_als
a
q
```

-high () -

```
function out = high(m,k,N)
%Vereinfachung der Zahlen durch Restklassenmultiplikation.
%Geht bis zur Ordnung 10^9, dann muss anders umgeformt werden.

m2 = m*mod(m,N);
m_index = m2;
for index = 3 : k
    m_index = mod(mod(m,N)* mod(m_index,N),N);
end

out=m_index;
```

-verhalten () - ist für das Veranschaulichen von einer periodischen *f(x)*

```
function verhalten(m,N,x_max)
%Achtung benötigt high()
if x_max > N
    x_max = N;
end
funz_x(1) = 1;%mod(m^0,N)
funz_x(2) = m;%mod(m^1,N)
funz_x(3) = mod(m^2,N);
funz_x(4) = mod(mod(m,N)*funz_x(3),N);

for k = 4:x_max
funz_x(k+1) = high(m,k,N);
end

figure(1)
plot([0:x_max],funz_x(1:x_max+1),'k.-')
xlabel('x')
ylabel('f(x) = (m^x) mod N')

figure(2)
plot([0:x_max],log(funz_x(1:x_max+1)),'k.-')
ylabel('log(f(x)) = log((m^x) mod N)')
```
