Die Überlegenheit von Quantenalgorithmen: Lektion 6 –Das Unknackbare knacken: Der Shor-Algorithmus

In dieser Lektion betrachten wir, wie ein von Peter Shor im Jahr 1994 entwickelter Quantenalgorithmus verwendet werden kann, um die RSA-Verschlüsselung zu knacken. Obwohl Quantencomputer derzeit noch nicht leistungsfähig genug sind, hat Shors Algorithmus bereits eine Revolution in der Kryptographie ausgelöst, um sicherzustellen, dass die Internetverschlüsselung auch in Zukunft sicher bleibt.

Bestimmen einer Periode

Shors Algorithmus nutzt eine Schwachstelle des RSA-Verfahrens aus, die darin besteht, dass modulare Potenzierung periodisch ist. Das bedeutet: Wenn man eine Zahl wiederholt in der Modulo-Arithmetik potenziert, kehrt man irgendwann wieder zur ursprünglichen Zahl zurück und das Muster wiederholt sich.

Potenzieren wir zum Beispiel 2 sukzessive modulo 5 erhalten wir:

$2^1 \equiv 2 \ (mod\ 5)$

$2^2 \equiv 4 \ (mod\ 5)$

$2^3 = 8 \equiv 3 \ (mod\ 5)$

$2^4 = 16 \equiv 1 \ (mod\ 5)$

$2^5 = 32 \equiv 2 \ (mod\ 5)$

Die Folge lautet also: 2, 4, 3, 1, 2, 4, 3, 1, und das Muster setzt sich in diesem Zyklus fort. Die erste Potenz, bei der wir $1 \ (mod\ 5)$ erhalten, legt die **Periode** des Zyklus, in diesem Fall 4, fest.

Aufgabe 1: Bestimme die Periode einer RSA-Verschlüsselung

In dieser Aufgabe verwenden wir ein RSA-Verfahren mit pq=77 (p=7, q=11) und Verschlüsselungsschlüssel e = 7. Wir nutzen den Code für die modulare Potenz aus der Lektion 5:

```
def powerMod(a, b, c): # raises a to the power of b modulo c
    res = a % c
    for i in range(0,b-1):
        res = a * res % c
    return res

pq=77
e=7
a=13
print(powerMod(a,e,pq))
```

Wenn wir mit einem Klartext von 13 beginnen, ergibt dies den verschlüsselten Wert 62 (der Geheimtext). Das Ziel beim Brechen von RSA besteht darin, von einem verschlüsselten Wert auszugehen (den man in einem Kommunikationskanal abfangen kann) und seine Periode zu finden.

Bestimme mithilfe der modularen Potenz $(mod\ 77)$ die Periode von 62, d. h. bestimme die erste Potenz von 62, welche $1 \ (mod\ 77)$ ergibt.

ENTNOMMEN AUS QUANTEN-COMPUTING IM MINT-UNTERRICHT
Ein Projekt von Science on Stage Deutschland e.V.

Wie die Periode hilft, das RSA-Verfahren zu knacken

Wenn wir beispielsweise die Zahl 16 mit e = 7 verschlüsseln, ergibt sich der verschlüsselte Wert 58.
58 hat eine Periode von 15 [die Liste der Potenzen ist 58, 53, 71, 37, 67, 36, 9, 60, 15, 23, 25, 64, 16, 4, 1].
(Beachte: Die Periode endet bei 1, da sich der größte gemeinsame Teiler von 16 und 77 zu 1 ergibt.)

Zwei wichtige mathematische Aussagen über die Periode helfen uns, RSA zu knacken:

(1) 16 (die Originalnachricht) ist in der Liste der Potenzen von 58.

(2) 16 hat ebenfalls eine Periode von 15 und die Liste der Potenzen von 16 ergibt dieselben Werte wie die Potenzen von 58, nur in anderer Reihenfolge: [16, 25, 15, 9, 67, 71, 58, 4, 64, 23, 60, 36, 37, 53, 1].

Das bedeutet: Wenn wir die **Periode des Geheimtexts** kennen, kennen wir auch die des **Klartexts**.

Wir bezeichnen die Periode mit **r**. Wir bestimmen eine Zahl **d'** mit

$$\boldsymbol{ed' \equiv 1 \ (mod \ r)}$$

In unserem Beispiel mit e=7 and r=15 können wir **d'** = 13 verwenden, da 7 x 13 = 91 $\equiv 1 \ (mod \ 15)$.

Wenn wir das Chiffrat nun mit **d'** potenzieren, erhalten wir den ursprünglichen Klartext zurück – nur mit öffentlich zugänglichen Informationen (**e** und **pq**). Damit hätten wir den Code gebrochen! Dies lässt sich nachrechnen:

Geheimtext	$y \equiv x^e \ (mod \ pq)$	
damit	$y^{d'} \equiv x^{ed'} \ (mod \ pq)$,	
aber	$x^{ed'} \equiv x^{1+kr} \ (mod \ pq)$	(da $\boldsymbol{ed' \equiv 1 \ (mod \ r)}$)
und	$x^{kr} \equiv 1 (mod \ pq)$	(da jede Potenz mit dem Vielfachen der Periode 1 ergibt)
und damit:	$x^{ed'} \equiv x \ (mod \ pq)$	

Verwenden wir unseren Code mit dem Geheimtext 58 und d'=13, finden wir:

$$\boldsymbol{58^{13} \equiv 16 \ (mod \ 77)}$$

Wir haben also den Klartext 16 mit einer Methode gefunden, die keine Kenntnis über die Originalnachricht hat!

Das Auffinden der Periode ist schwer für einen klassischen Computer, da es neben der Periode kein erkennbares Muster in den Potenzen gibt. Ein klassischer Computer müsste daher entweder die Periode raten oder alle Potenzen durchrechnen, bis 1 erreicht wird. Für reale RSA-Zahlen (mit mehreren hundert Stellen) ist das unmöglich in einem praktischen Zeitrahmen auf einem klassischen Computer umsetzbar.

Quantencomputer lösen unser Problem!

Für einen klassischen Computer ist das Finden der Periode nicht effizient lösbar. Jedoch gibt es eine geniale Methode zur schnellen Bestimmung einer Periode durch Ausnutzen der Quanten-Fourier-Transformation. Diese wird im Shor-Algorithmus verwendet:

(1) Wir implementieren die modulare Potenz in einem Schaltkreis aus Quantengattern. Dabei muss die Anzahl der Qubits groß genug sein, um die Größe der RSA-Zahl (**pq**) zu kodieren;

(2) Wir wenden auf jedes Input-Qubit ein Hadamard Gatter an, um eine gleichgewichtete Superposition aller Basiszustände zu erhalten (dies ist die gleiche Strategie wie bei den Quanten-Algorithmen aus den anderen Lektionen) und wenden die Funktion auf diesen Zustand an;

(3) Wir führen Messungen auf den Outputs der Funktion an, so dass der Zustand zu einer Superposition von Zuständen kollabiert, mit denen die Periode der Funktion gefunden werden kann;

(4) Wir wenden die Quanten-Fourier-Transformation auf diesen Zustand an und führen wieder eine Messung durch, um die Periode zu erhalten.

Im 4. Schritt erhalten wir zwar nur ein (sehr) wahrscheinliches Ergebnis für die Periode und nicht zwingend den exakten Wert. Diese Information reicht aber bereits aus, da die Periode auf einem klassischen Computer schnell überprüft werden kann.

Praktisches Beispiel

Im folgenden Beispiel bilden wir eine modulare Potenz für den Schlüssel **pq**=15 mit Basis 7. In der linken Tabelle sind die Potenzen von 7 aufgeführt. In der rechten Tabelle sind diese Zahlen im Binärcode dargestellt, hier ist also die Funktion dargestellt, wie sie durch Gatter implementiert wird:

x	$7^x \bmod 15$
1	7
2	4
3	13
4	1
5	7
6	4
7	13
8	1
9	7
10	4
11	13
12	1
13	7
14	4
15	13
16	1

Input	Output
$x_3x_2x_1x_0$	$x_7x_6x_5x_4$
0001	0111
0010	0100
0011	1101
0100	0001
0101	0111
0110	0100
0111	1101
1000	0001
1001	0111
1010	0100
1011	1101
1100	0001
1101	0111
1110	0100
1111	1101
0000	0001

Wir verwenden die vier Qubits $|x_0\rangle, |x_1\rangle, |x_2\rangle, |x_3\rangle$, um die Input-Binärzahl $x_3x_2x_1x_0$ darzustellen, und die vier Qubits $|x_4\rangle, |x_5\rangle, |x_6\rangle, |x_7\rangle$ für die Output-Binärzahl $x_7x_6x_5x_4$.

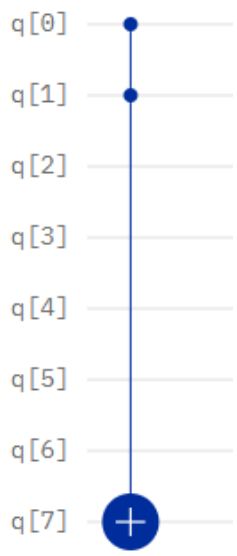
Wir konstruieren einen Schaltkreis durch Kombination von NOT-, CNOT- und Toffoli(CCNOT)-Gattern, um die Funktion aus der Tabelle zu implementieren.

Man sieht in der Tabelle, dass der Wert des Outputs nur von den beiden letzten Input-Qubits abhängt. Daher brauchen die beiden Qubits $|x_2\rangle$ und $|x_3\rangle$ nicht verbunden werden und können in diesem Fall ignoriert werden.

Wir konstruieren den Schaltkreis, indem wir betrachten, wie $|x_0\rangle$ und $|x_1\rangle$ jedes Output-Qubit beeinflussen. Wir beginnen bei $|x_7\rangle$ (das letzte Output-Qubit) und extrahieren die relevanten Informationen aus der Tabelle für die Abhängigkeit von $|x_0\rangle$ und $|x_1\rangle$:

x_1x_0	x_7
01	0
10	0
11	1
00	0

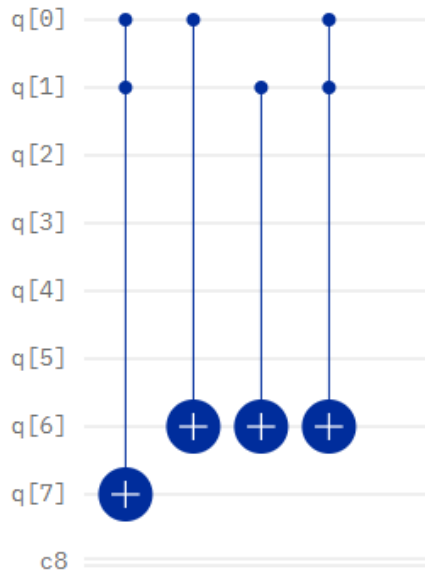
Das Qubit $|x_7\rangle$ ist also nur dann $|1\rangle$, wenn sowohl $|x_0\rangle$ als auch $|x_1\rangle$ im Zustand $|1\rangle$ sind, und $|0\rangle$ in allen anderen Fällen (im Klassischen ist dies eine UND-Operation). Implementiert wird dies durch ein Toffoli-Gatter, ein doppeltes CNOT, welches das Ziel-Qubit ändert, wenn beide Kontroll-Qubits in $|1\rangle$ sind. Im Quanten Composer wird dies so dargestellt:



Für $|x_6\rangle$:

x_1x_0	x_6
01	1
10	1
11	1
00	0

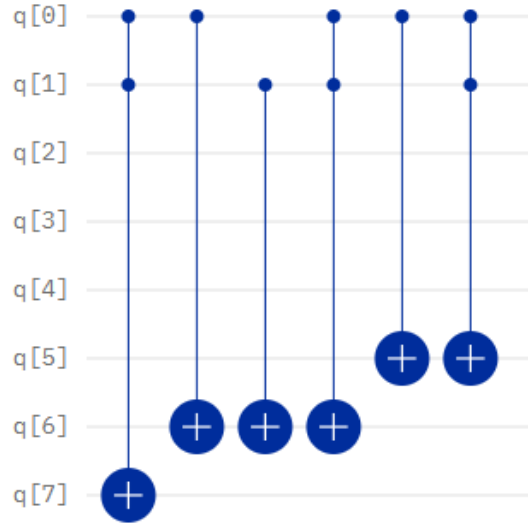
Der Output ist $|1\rangle$, wenn mindestens eines der Inputs $|x_0\rangle$ oder $|x_1\rangle$ im Zustand $|1\rangle$ ist (eine klassische ODER Operation). Wir können je ein CNOT verwenden, für den Fall, dass jeweils nur eines der Input-Qubits in Zustand $|1\rangle$ ist. Wenn jedoch beide Inputs in Zustand $|1\rangle$ sind, werden die beiden CNOTs den Zustand von $|x_6\rangle$ wieder zu $|0\rangle$ ändern. Dieser Fall wird durch ein Toffoli-Gatter aufgefangen:



Für $|x_5\rangle$:

x_1x_0	x_5
01	1
10	0
11	0
00	0

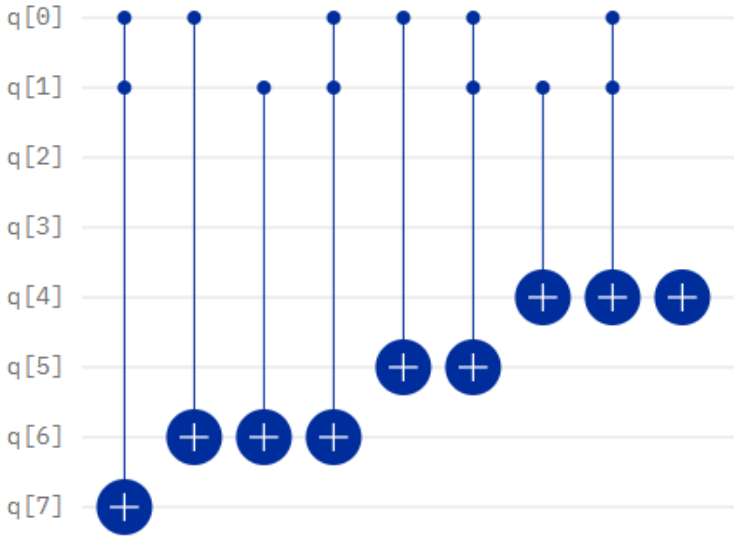
Der Output ist nur dann $|1\rangle$, wenn $|x_0\rangle = |1\rangle$ und $|x_1\rangle = |0\rangle$. Dies erreichen wir durch ein CNOT-Gatter, welches $|x_0\rangle$ und $|x_5\rangle$ verbindet und einem anschließenden Toffoli-Gatter, welches den Fall korrigiert, wenn beide Inputs in Zustand $|1\rangle$ sind:



Für $|x_4\rangle$:

x_1x_0	x_4
01	1
10	0
11	1
00	1

Der Output $|x_4\rangle$ ist also $|1\rangle$, wenn nur $|x_1\rangle$ in Zustand $|1\rangle$ ist. Dazu wenden wir ein CNOT-Gatter zwischen $|x_1\rangle$ und $|x_4\rangle$ sowie ein Toffoli-Gatter von $|x_0\rangle$ und $|x_1\rangle$ zu $|x_4\rangle$, gefolgt von einem NOT-Gatter auf $|x_4\rangle$:



Dieser Schaltkreis implementiert insgesamt die Funktion $7^x \pmod{15}$ für alle Inputs von x. Dies entspricht dem Schritt (1) des Shor-Algorithmus.

Aufgabe 2: Baue einen Schaltkreis für eine andere modulare Potenz

Entwickle den Schaltkreis für die Potenz $8^x \pmod{15}$. Beginne bei den beiden Tabellen und erzeuge, wie im Beispiel zu $7^x \pmod{15}$, schrittweise den Schaltkreis. Beachte, dass der Output wieder nur von $|x_0\rangle$ und $|x_1\rangle$ abhängt.

x	$8^x \bmod 15$
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	

Input	Output
$x_3x_2x_1x_0$	$x_7x_6x_5x_4$
0001	
0010	
0011	
0100	
0101	
0110	
0111	
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	
0000	

Knacken des Codes

Mit dieser Funktion der modularen Potenz können wir den Algorithmus implementieren:

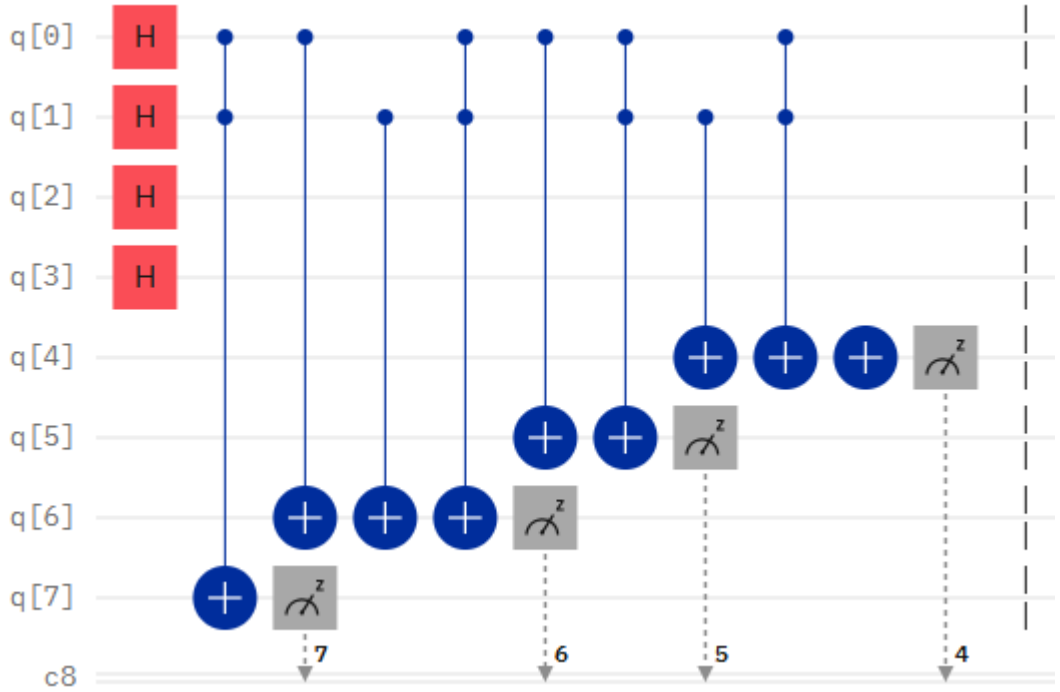
(1) Wir wenden Hadamard-Gatter auf alle Input-Qubits an und anschließend den Schaltkreis der modularen Potenz. Damit erhalten wir eine Superposition aller möglichen Output-Zustände der modularen Funktion.

(2) Führe eine Messung auf allen Output-Qubits durch. Dadurch wird die Superposition der Input-Qubits auf die Zustände reduziert, die die Periode erzeugen.

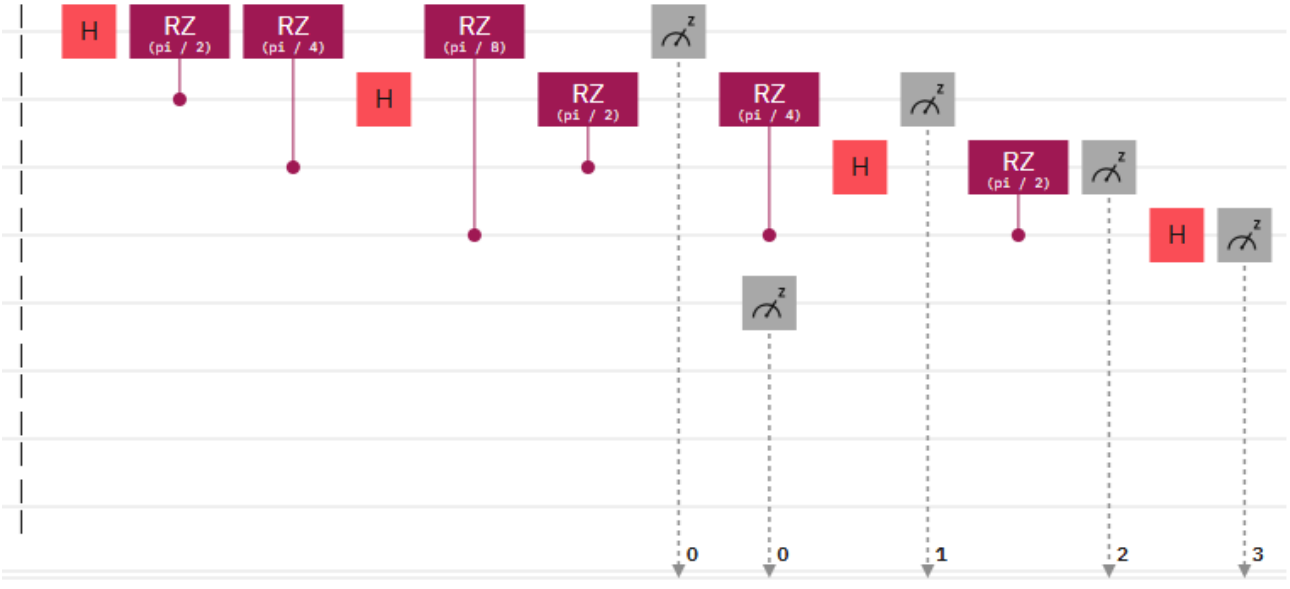
(3) Wende die Quanten-Fourier-Transformation durch, um einen Zustand zu erzeugen, dessen Wahrscheinlichkeitsdichtefunktion durch die Periode der modularen Potenz bestimmt ist.

(4) Führe eine Messung auf den Input-Qubits durch. Führe den Algorithmus mehrmals aus und verwende die Ergebnisse mit der höchsten Häufigkeit, um die Periode der Wahrscheinlichkeitsdichtefunktion zu erhalten.

Der komplette Algorithmus, in Gattern implementiert, ist in den folgenden Abbildungen für den Fall $7^x \pmod{15}$ gezeigt. Für die Übersichtlichkeit ist dies in zwei Abbildungen dargestellt. In der ersten Abbildung sind die ersten beiden Schritte (Hadamard, modulare Potenz und Messung der Output-Qubits) dargestellt:

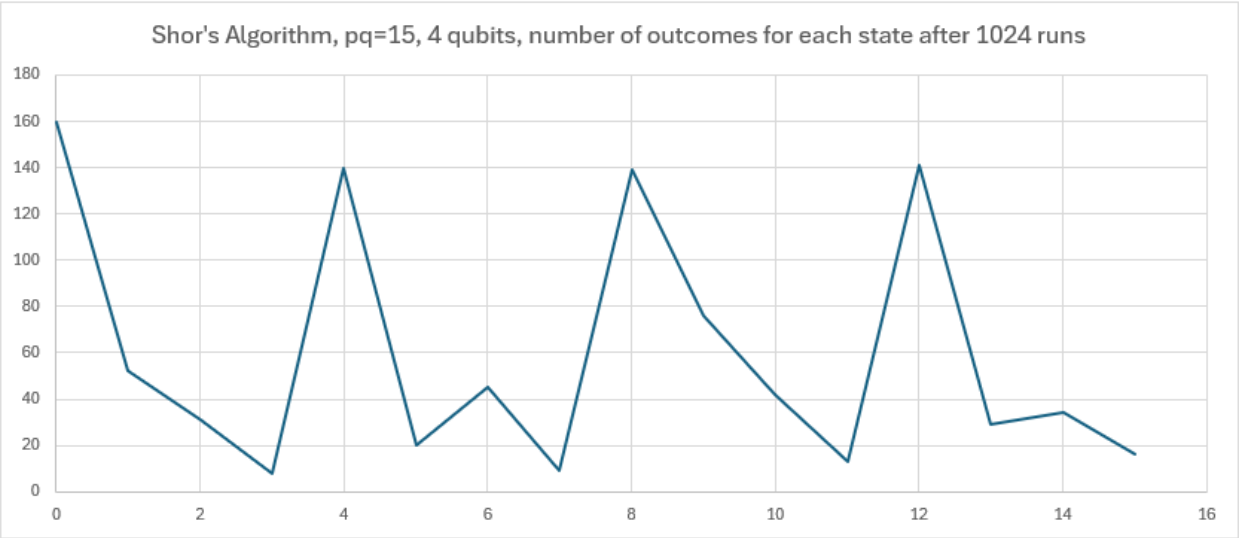


Anschließend folgt die Quanten-Fourier-Transformation und die Messung auf den input-Qubits. Die Quanten-Fourier-Transformation selbst ist eine Kombination aus Hadamard-Gattern und kontrollierten Rotationsgattern (dies wird hier nicht weiter ausgeführt):



Die Wahrscheinlichkeitsdichtefunktion des Endzustands der Input-Qubits hat Peaks bei Vielfachen von 2^n / r, wobei n die Anzahl der Qubits ist und r die Periode der modularen Potenz. Wenn man die Lage der Peaks betrachtet, kann man demnach die Periode r abschätzen.

Der Graph zeigt ein Ergebnis, wenn man den Algorithmus (s. oben) für $7^x \pmod{15}$ 1024 mal laufen lässt:



Man kann gut erkenne, dass Peaks bei Vielfachen von 4 auftreten. Da in diesem Fall n = 4 und damit $2^n = 2^4 = 16$ ist, ist die Periode $r = {}^{16}/_{4} = 4$. In diesem einfachen Beispiel konnte man dies bereits in der Tabelle erkennen. Bei größeren Schlüsseln wären wir allerdings nicht mehr in der Lage, die Periode vorher zu erkennen. In diesen Fällen liefert der Shor-Algorithmus eine Lösung des Problems in umsetzbarer Laufzeit.

Aufgabe 3: Überprüfe, dass das RSA-Verfahren geknackt wurde

Wir verwenden als Beispiel pq=15, Klartext =7 und Verschlüsselungsschlüssel e = 3:

(a) Berechne den Geheimtext.

(b) Nutze die Periode, die wir mit dem Shor-Algorithmus gefunden haben (r=4), um d' zu berechnen. (Beachte, dass d' so gewählt wird, dass $e.d' \equiv 1\ (mod\ r)$)

(c) Überprüfe, dass der Geheimtext wieder den Klartext ergibt, wenn er mit d' potenziert (mod 15) wird.

Herzlichen Glückwunsch! Du hast die RSA-Verschlüsselung geknackt!

Schlusswort

Shors Algorithmus wurde bisher nicht auf einem Quantencomputer implementiert, so dass reale RSA-Systeme bedroht sind. Unser Beispiel war stark vereinfacht, da die zugrunde liegende Funktion eine kurze Periode hatte. In echten Anwendungen gibt es keine einfache Abbildung von Input- auf Output-Qubits, und bisher wurde kein allgemeiner Weg für die Konstruktion des Schaltkreises für beliebige modulare Potenzen entwickelt.

Doch allein die Aussicht, dass Shors Algorithmus eines Tages praktikabel wird, hat zur Entwicklung eines neuen Forschungsfeldes geführt: Post-Quantum-Kryptographie – entworfen, um unsere Daten auch in einer Quantenwelt sicher zu halten.