Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelorarbeit

Einsatz von Quantencomputern zur Abschwächung klassischer Verfahren der Kryptographie

vorgelegt von

Ole Knief
Matrikelnummer 7306842
Studiengang Informatik

eingereicht am 24. August 2022

Betreuer: Dr.-Ing. Daniel Demmler

Erstgutachter: Dr.-Ing. Daniel Demmler

Zweitgutachter: Prof. Dr.-Ing. Hannes Federrath

Aufgabenstellung

Im Laufe der letzten Jahre hat die Forschung im Bereich Quantencomputing rapide Fortschritte gemacht. Seit 2018 gibt es erneut einen exponentiellen Zuwachs in der Anzahl verfügbarer Qubits in Quantencomputern. Beispielsweise prognostizieren Hochrechnungen von IBM bis zum Jahr 2023 eine Qubitanzahl von 1121 [Bol22]. Unter Fortsetzung dieser Entwicklung steigt die zukünftig erreichbare Rechenleistung, unter Nutzung der Prinzipien der Quantenphysik, so weit, dass sie neue Herangehensweisen an die Entschlüsselung von vielen aktuell genutzten kryptographischen Verfahren bietet [BL17].

Diese Ausarbeitung soll die Arbeitsweise eines Quantencomputers bei Angriffen auf Algorithmen der klassischen Kryptographie darlegen. Ziel ist es, im Detail zu analysieren, welche quantenmechanischen Eigenschaften auf welche Weise genutzt werden können, um die Sicherheit von kryptographischen Verfahren effektiv abzuschwächen. Zur Einordnung soll der aktuelle Forschungsstand des Bereichs Quantencomputing skizziert werden und eine Abschätzung der Voraussetzungen erfolgen, die für die praktische Anwendung von Abschwächungen der Kryptographie erfüllt sein müssen. Es soll auch ein Einblick in die Komplexitätstheorie gegeben werden. Um in ihrem Kontext Laufzeiten von Quantenalgorithmen gegen klassische analytische Algorithmen abzugrenzen und deren Skalierung zu betrachten, soll die Funktionsweise von Quantenalgorithmen anhand beispielhafter Implementationen erläutert werden. Dies soll mithilfe einer Auswahl von bereits implementierten Quantencomputing-Simulatoren durchgeführt werden. Auf diese Weise wird aufgezeigt, wie Quantenalgorithmen bestehende klassische Verfahren der Kryptographie brechen.

Zusammenfassung

Klassische Verfahren der Kryptographie basieren oft auf der Komplexität mathematischer Probleme, welche durch einen klassischen Algorithmus mit exponentiellem Zeitbedarf gelöst werden können. Diese Berechnungen haben für ausreichend große Parameter einen solch hohen Ressourcenbedarf, dass sie in der Praxis nicht lösbar sind. Quantenalgorithmen ermöglichen eine neue Herangehensweise an diese Probleme.

Quantencomputer manipulieren Qubits, welche nicht nur den Zustand $|0\rangle$ oder $|1\rangle$ annehmen, wie es bei klassischen Computern der Fall ist, sondern auch in Superpositionen vorzufinden sind, in denen sich ein Qubit in beiden Zuständen gleichzeitig befindet. Qubits werden daher als Matrizen repräsentiert. Quantengatter übernehmen die Funktion der Manipulation, indem sie die Matrizen nach Belieben verändern. Aus komplexeren Strukturen von Quantengattern lassen sich Quantenalgorithmen entwerfen, die auf Quantencomputern ausgeführt werden können.

Quantenalgorithmen können mithilfe von Superpositionen mehrerer Qubits komplizierte Berechnungen für alle möglichen Kombinationen dieser Qubits ausführen. Mithilfe jener Quantenparallelisierung kann Shors Algorithmus Lösungen mancher symmetrischen Verfahren der Kryptographie effizient berechnen und sie somit brechen. Mit Grovers Algorithmus können quantenmechanische Eigenschaften genutzt werden, um die effektive Schlüssellänge mancher symmetrischen Kryptographieverfahren zu halbieren.

Die Forschung ist allerdings noch Jahre davon entfernt, einen Quantencomputer zu bauen, der diese Quantenalgorithmen ausführen kann. Der Zeitpunkt, an dem Quantencomputer manche Probleme schneller lösen als klassische Computer, liegt demnach noch weit in der Zukunft. Zusätzlich existieren bereits Algorithmen der Post-Quanten-Kryptographie, die nach dem aktuellen Kenntnisstand nicht durch Quantencomputer abgeschwächt werden können und daher in Zukunft Verwendung finden werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

„I think I can safely say that nobody
really understands quantum mechanics.“

- Richard P. Feynman [Car19]

Kryptographische Verfahren bilden das Herzstück der digitalen Sicherheit. Das grundlegende Prinzip hat sich bereits über 2000 Jahre durchgesetzt [Buc16, S. 75] und findet seit den 1970er Jahren auch Anwendung im digitalen Raum [Buc16, S. 76], welches durch den hohen Grad an Kommunikation mithilfe digitaler Technologien bedingt ist. Darunter fallen auch höchst sensible Daten, wie die digitale Verwaltung von Finanzdaten und Zugänge zu Kryptowährungen. Viele Online-Systeme sind heutzutage im Alltag eines jeden Menschen. Es wurden daher verschiedene mathematische Probleme entwickelt, auf denen die unterschiedlichen Verfahren der Kryptographie basieren. Langfristig musste nur darauf geachtet werden, dass die Schwierigkeit der mathematischen Probleme mit der Entwicklung modernerer und schnellerer Prozessoren mitzieht [Buc16, S. 79].

Vor fast 100 Jahren, 1925, formulierten die Wissenschaftler M. Born, P. Jordan und W. Heisenberg die Grundlagen der Quantenmechanik [FP09]. Sie beschrieben damit eine Betrachtung der Welt auf einer Ebene, die vorher nicht erfassbar war. Das Bild der klassischen Realitätsbeschreibung wurde umgeworfen. So meinte der theoretische Physiker Neil Turok: „Quantum physics is one of the hardest things to understand intuitively, because essentially the whole point is that our classical picture is wrong.“ [McI12]. Auf diesem Konzept aufbauend, beschrieb Paul Benioff 1980 erstmalig die mögliche Konstruktion eines Computers, welcher den Grundgesetzen der Quantenmechanik unterliegt [Ben79]. Er legte damit das Fundament einer Konstruktion, die über die Möglichkeiten eines klassischen Computers hinausschießt.

Die Verbindung von komplexeren Konzepten aus der Informatik und der unintuitiven Weltbetrachtung, die von der Quantenmechanik ausgeht, erschwert die Erfassung der Möglichkeiten, welche Quantencomputer bieten. Die logische Folge daraus sind starke Abstrahierungen zum Thema im Rahmen von „Pop-Science“, welche das Bild von Quantencomputing verzerren. Aussagen zur angeblichen Entfaltung von Rechenleistung, die das Spektrum des menschlichen Wissens drastisch beeinflussen könne [Dil12], stellen Quantencomputer teilweise als eine allmächtige Rechenmaschine dar. Dies wird speziell dann interessant, wenn Quantencomputern die Fähigkeit zugeschrieben wird, wichtige Kryptographieverfahren wesentlich schneller zu brechen, als es ein klassischer Computer könne [Cho22]. Droht durch die Entwicklung von Quantencomputern ein vollständiger Bruch der digitalen Sicherheit?

Diese Frage soll im Fokus dieser Ausarbeitung stehen. Um sich der Antwort dieser Frage anzunähern, wird nach einer Betrachtung der nötigen Grundlagen von Quantenphysik und klassischen Verfahren der Kryptographie ein Einblick in das Fundament von Quantencomputern gegeben. Das Verständnis der Funktionsweise wird anschließend genutzt, um zwischen klassischen Algorithmen und Quantenalgorithmen zu unterscheiden und zu betrachten, worin die

neuen Möglichkeiten bestehen. Daraufhin werden Quantenalgorithmen untersucht, die aktuelle Kryptographieverfahren eventuell gefährden können, indem sie zusätzlich in Simulatoren implementiert werden. Abschließend wird mit diesem Wissen der aktuelle Forschungsstand beleuchtet, um abzuschätzen, wie weit das Forschungsfeld Quantencomputing fortgeschritten ist, sowie was noch passieren muss, welche Auswirkungen die Entwicklungen zukünftig haben und wann Quantencomputer unser Leben im hohen Maße beeinflussen werden.

Alle Codeausschnitte, die in dieser Ausarbeitung vorkommen, stehen zusätzlich als ausführbare Python-Skripte zur Verfügung und sind im Github-Repository [Kni22] zu finden.

2 Grundlagen

Der Umgang mit Quantencomputern erfordert nicht nur erweiterte Kenntnisse in der Informatik, sondern auch Kenntnis der mathematischen Notationen, welche in der Quantenphysik verwendet werden. In diesem Kapitel werden alle nötigen Definitionen aufgegriffen und die genutzten Schreibweisen definiert. Fundamentale Grundlagen werden nicht erklärt.

2.1 Mathematische Notation

In diesem Kapitel werden die mathematischen Grundlagen definiert, welche für das Verständnis von Quantenphysik und Kryptographie nötig sind. Da die Beweise etwaiger mathematischer Verfahren für das Verständnis nicht notwendig sind, wird an der Stelle auf die Fachliteratur verwiesen.

2.1.1 Division in Restklassen

Mehrere Verfahren der Kryptographie, welche im Rahmen dieser Ausarbeitung beleuchtet werden, basieren auf einer Division mit Rest, welcher durch eine Restklasse bestimmt wird.

Definition 2.1.1 (*Division in Restklassen*)

Sei eine Zahl $a \in \mathbb{Z}$ und ein $n \in \mathbb{N}$ gegeben. Dann beschreibt

$$a \mod n$$

den Rest von a nach der ganzzahligen Division durch n [Sch16, S. XI].

2.1.2 Komplexe Zahlen

Für die Arbeit mit Wellenfunktionen reichen die gängigen Zahlenräume $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ und $\mathbb{R}$ nicht aus, da sie aus komplexen Zahlen bestehen. Es wird der Zahlenraum der komplexen Zahlen $\mathbb{C}$ definiert.

Definition 2.1.2 (*Zahlenraum der komplexen Zahlen*)

Eine Zahl z ist eine komplexe Zahl, wenn

$$z = a + ib$$

gilt, mit $a, b \in \mathbb{R}$. Für die imaginäre Zahl i gilt

$$i = \sqrt{-1}.$$

Die komplexen Zahlen bilden den Zahlenraum $\mathbb{C}$ [DL10, S. 9].

Oftmals wird a als Realteil von z beschrieben, während ib als Imaginärteil von z verstanden wird [DL10, S. 10]. Komplexe Zahlen können auch in der Gaußschen Zahlenebene, die in Abbildung 2.1 zu sehen ist, betrachtet werden. Der Realteil einer komplexen Zahl liegt auf der horizontalen Achse und der Imaginärteil auf der vertikalen Achse. [DL10, S. 30]

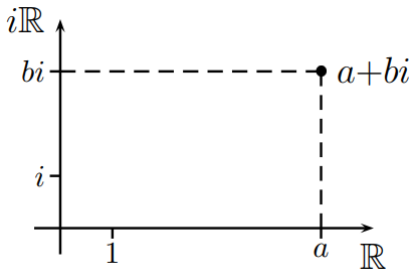

Abbildung 2.1: Die imaginären Zahlen als Darstellung in der Gaußschen Zahlenebene [DL10, S. 30].

2.1.3 Tensorprodukt

Zwei oder mehrere Tensoren können mithilfe des Tensorprodukts aufeinander abgebildet werden. Im Verlauf dieser Ausarbeitung wird das Tensorprodukt von Vektoren und Matrizen gebildet.

Definition 2.1.3 (*Tensorprodukt*)

Es seien zwei Tensoren A, B gegeben, sowie zwei endlichdimensionale Hilberträume $\mathbb{H}^a, \mathbb{H}^b$, wobei $A \in \mathbb{H}^a$ und $B \in \mathbb{H}^b$ gilt. Dann kann das Tensorprodukt

$$H = H^a \otimes H^b$$

gebildet werden [Sch16, S. 58]. Das Tensorprodukt H wird aus $a \cdot b$ Koeffizienten gebildet und in einer $a \times b$ Matrix H zusammengefasst. Bei zwei 2×2 Matrizen

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \; B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

ergibt sich

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{pmatrix}$$

[McM08, S. 83].

2.2 Klassische Verfahren der Kryptographie

Im digitalen Alltag werden verschiedene kryptographische Verfahren genutzt. In diesem Kapitel wird die Aufteilung in symmetrische und asymmetrische Verfahren vorgenommen. Zusätzlich

wird eine Auswahl der bekanntesten Verschlüsselungsverfahren innerhalb der beiden Kategorien erklärt, sodass die Arbeitsweise dieser Verfahren verstanden wird.

2.2.1 Symmetrische Kryptographie

Verschlüsselungen, die auf symmetrischen Verfahren basieren, nutzen denselben Schlüssel, um eine Nachricht sowohl zu verschlüsseln als auch zu entschlüsseln.

Definition 2.2.1 (*Symmetrische Verfahren*)

Symmetrische Verschlüsselungsverfahren werden durch ein Tupel $(K, P, C, KeyGen, Enc, Dec)$ beschrieben. K beschreibt die Menge des Schlüsselraums mit Elementen, die „Schlüssel" genannt werden. P ist die Menge des Klartextraums mit Elementen namens „Klartext". C ist der Chiffretextraum. Schlüssel dieser Menge werden „Chiffretexte" oder „Schlüsseltexte" genannt.

KeyGen bezeichnet einen probabilistischen Algorithmus, welche einen Schlüssel k definiert, mit $k \in K$. Der probabilistische Verschlüsselungsalgorithmus Enc nutzt k und einen Klartext $p \in P$, um einen Chiffretext $c \in C$ zu erzeugen. Dieser kann mithilfe von k und dem deterministischen Entschlüsselungsalgorithmus Dec in p zurückverwandelt werden [Buc16, S. 73-74].

Im Folgenden werden drei symmetrische Kryptographieverfahren grundlegend erklärt, welche dem aktuellen Sicherheitsstandard entsprechen und weit verbreitet sind, um diese später im Kontext des Quantencomputings zu bewerten.

Advanced Encryption Standard – AES

Dieses Verfahren ist ein Spezialfall der Rijndael-Chiffre, welche 1997 beim US National Institute of Standards and Technology (NIST) vorgeführt wurde. Die Chiffre besteht aus einem Algorithmusabschnitt, durch welchen mehrfach iteriert wird. Die Anzahl der Iterationen kann je nach der Schlüssellänge, welche zwischen 128, 192 und 256 Bits variiert, 10, 12 bzw. 14 betragen [Buc16, S. 145]. In dieser vereinfachten Betrachtung soll die Implementation von AES-128 dargestellt werden.

Zu Beginn des Prozesses müssen aus dem ursprünglichen 128-Bit Schlüssel 11 unterschiedliche Schlüssel erzeugt werden, damit für jede neue Iteration innerhalb der AES-128 Chiffre, siehe Def. 2.2.2, ein anderer Schlüssel verwendet werden kann. Der dafür genutzte Algorithmus heißt KeyExpansion [Buc16, S. 150]. Die erzeugten Schlüssel werden als Roundkeys bezeichnet. Der Ablauf von AES ist in Abbildung 2.2 grafisch als Flussdiagramm aufbereitet.

Definition 2.2.2 (*AES-128*)

Der 128-Bit Klartextblock wird in eine 4×4 Matrix überführt, gefüllt mit jeweils einem „SubByte“ $S_{b,w}$ mit $b,w \in [0;3]$, wobei w die vier 32 Bit langen Wörter beschreibt und b die vier Bytes des 32 Bit langen Wortes.

$$\begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix}$$

SubBytes bilden ihren Inhalt durch ihre nicht lineare Funktion auf einen anderen Byte ab, ist hierbei allerdings nicht weiter von Relevanz. Der Ablauf des Verfahrens ist in Abbildung 2.2.2 visualisiert. Nach der anfänglichen Verschlüsselung des Klartextes erfolgen zehn Iterationen der folgenden drei Operationen:

1. Matrixzeilen verschieben: Jedes Subbyte $S_{b,w}$ der Matrix wird in einem zyklischen Linksshift um b Stellen nach links verschoben. Die Transformation hat die folgende Auswirkung:

$$\begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,1} & S_{1,2} & S_{1,3} & S_{1,0} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,3} & S_{3,0} & S_{3,1} & S_{3,2} \end{pmatrix}$$

2. Matrixspalten vermischen: Jeder Eintrag der neuen Matrix wird mit einer festgelegten binären Matrix multipliziert. Für jeden Matrixeintrag m erfolgt folgende lineare Transformation:

$$m \leftarrow \begin{pmatrix} 10 & 11 & 01 & 01 \\ 01 & 10 & 11 & 01 \\ 01 & 01 & 10 & 11 \\ 11 & 01 & 01 & 10 \end{pmatrix} m$$

3. Roundkey anwenden: Schlussendlich wird der Roundkey der aktuellen Iteration mithilfe des bitweisen $\oplus$ Operators angewandt. Für einen Roundkey r und Matrix mit 128-Bit Inhalt i ergibt sich folgende Transformation:

$$i \leftarrow i \oplus r$$

Wenn das AES-128 Verfahren erfolgreich abgeschlossen wurde, kann die verschlüsselte Nachricht mithilfe desselben 128-Bit Schlüssel und der inversen Chiffre entschlüsselt werden [Buc16, S. 146-149].

Kryptographische Hashfunktion

Hashfunktionen gibt es in vielen Variationen, die für unterschiedliche Zwecke bestimmt sind. Die kryptographischen Hashfunktionen, die ein breites Spektrum von Anwendungsgebieten abdecken, übernehmen die Funktion von Authentizitätsprüfung, digitale Signaturen oder auch die Erzeugung von qualifizierten Zeitstempeln. Einen Einblick in das weite Feld von Hashfunktionen gibt [SG12]. Gängige Implementationen für Hashfunktionen, die einen digitalen Fingerabdruck

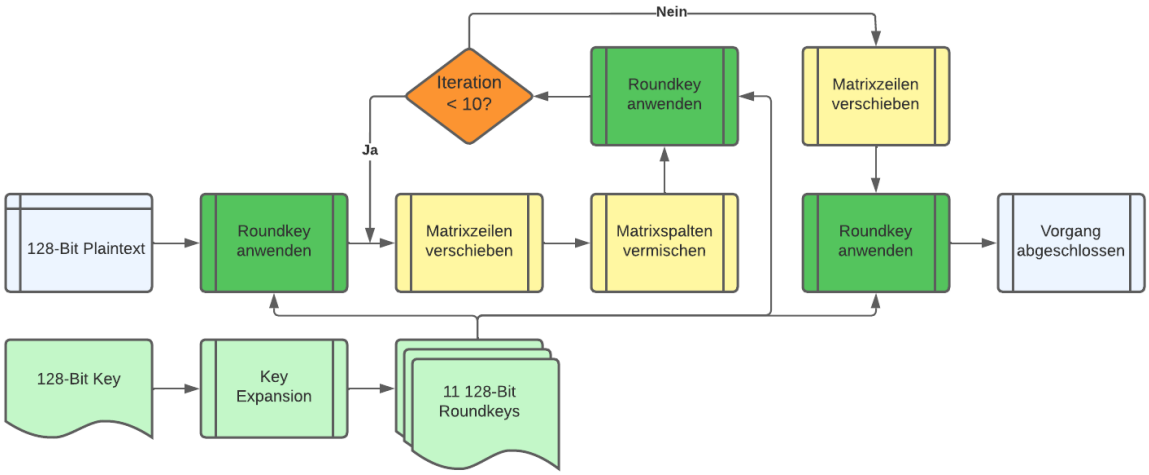

Abbildung 2.2: Der Ablauf des AES Verfahrens als Flussdiagramm respräsentiert

von Eingaben erzeugen und heutzutage genutzt werden und als sicher gelten, sind SHA-256 oder auch SHA3-256 [BL17]. In diesem Kontext soll aber nur die Grundidee von Hashfunktionen erläutert werden.

Definition 2.2.3 (*Einweg-Hashfunktion*)

Eine Hashfunktion ist eine Funktion $h : D \to R$, mit einer Nachricht $D = \{0,1\}^*$ und einem Resultat $R = \{0,1\}^n$ für $n > 0$.

An eine Einweg-Hashfunktion sind mindestens die folgenden fünf Bedingungen geknüpft:

1. Die Funktion h muss eine beliebige Größe der Nachricht D zulassen, die von praktischer Relevanz ist.

2. Das ausgegebene Resultat R hat eine festgelegte Länge n.

3. Bei einer gegebenen Eingabe x erfordert die Funktion h keinen hohen Rechenaufwand, um den Hash $h(x)$ zu berechnen.

4. Bei einem gegebenen Hash $h(x)$ muss der Rechenaufwand, um die Eingabe x zu berechnen, so hoch sein, dass die Berechnung praktisch undurchführbar ist.

5. Für einen Hash $h(x)$ muss der Rechenaufwand, um eine weitere Eingabe x' zu berechnen, wobei $h(x) = h(x')$ gilt, so hoch sein, dass die Berechnung praktisch undurchführbar ist [SG12].

Message-Authentication Codes (MACs)

Teilweise ist es nötig zu überprüfen, ob eine Nachricht oder ein Dokument echt ist und nicht durch eine veränderte Version ausgetauscht wurde, ohne dass es jemand mitbekommen hat. Für diesen Zweck wurden Message-Authentication Codes eingeführt. Sie ermöglichen die Überprüfung der Authentizität eines beliebigen Datenblocks.

Definition 2.2.4 (*Message-Authentication Codes*)

Sei k ein Schlüssel und m eine Nachricht, ein Dokument oder beliebiger Datenblock. Dann existiert ein Algorithmus MAC, der die Transformation

$$MAC(k,m) = s$$

durchführt, die den passenden MAC s erzeugt. Mithilfe eines Verifikation-Algorithmus VER kann überprüft werden, ob ein MAC s' das Resultat der vorherigen Transformation ist. Gilt $s = s'$, so gibt $VER(k,m,s')$ 1 zurück. Andernfalls erfolgt die Ausgabe von 0.

Eine sichere Implementation soll eine schnelle Berechnung von VER ermöglichen, während eine Berechnung einer Fälschung praktisch undurchführbar ist. Eine Fälschung ist dann gefunden, wenn eine andere Nachricht m' existiert, mit welcher $MAC(k,m')$ ein s' erzeugt, mit $s' = s$ [Buc16, S. 241-243].

2.2.2 Asymmetrische Verfahren

Um symmetrische Verfahren durchführbar zu machen, muss der Sender der Nachricht den symmetrischen Schlüssel an den Empfänger senden, damit dieser die Nachricht entschlüsseln kann. Dieser Austausch des Schlüssels muss zusätzlich gesichert werden, wofür in der Regel asymmetrische Verfahren genutzt werden [Buc16, S. 76]. Darüber hinaus werden die asymmetrischen Verfahren auch für selbständige Verschlüsselungen, Authentifizierungs-Verfahren und Integritätssicherung genutzt [Hau06].

Definition 2.2.5 (*Asymmetrische Verfahren*)

Asymmetrische Verschlüsselungen, auch Public-Key-Kryptosysteme genannt, nutzen zwei verschiedene Schlüssel, um eine Nachricht zu ver- und entschlüsseln. Ein Sender A kann den öffentlich zugängigen Schlüssel des Empfängers B nutzen, um einen Klartext p in einen Chiffretext c zu verwandeln. Mithilfe des privaten Entschlüsselungsschlüssel des Empfängers B kann aus dem Chiffretext c der Klartext p gewonnen werden [Buc16, S. 76-77].

RSA-Verfahren

Das RSA-Verfahren von Ron Rivest, Adi Shamir und Len Adleman ist ein wichtiges asymmetrisches Verschlüsselungsverfahren, das aktuell im Einsatz ist. Es basiert auf dem Problem, dass große Zahlen nicht effizient in ihre Primfaktoren zerlegt werden können und bietet daher sichere Verschlüsselungen [Buc16, S. 168].

Definition 2.2.6 (*RSA*)

Zu Beginn wird ein RSA-Modul n mit einer Bitlänge von k für $k \in \mathbb{N}$ gebildet. Für n gilt:

$$n = pq,$$

wobei p und q Primzahlen sind. Zusätzlich wird eine natürliche Zahl e bestimmt, für die gilt:

$$1 < e < (p-1)(q-1)$$

sowie

$$ggT(e,(p-1)(q-1)) = 1.$$

Der Algorithmus „ggT“ bestimmt hierbei den größten gemeinsamen Teiler. Außerdem wird eine weitere natürliche Zahl d durch den erweiterten euklidischen Algorithmus berechnet mit:

$$1 < d < (p-1)(q-1)$$

und

$$de \mod (p-1)(q-1) = 1.$$

Schlussendlich hat der Algorithmus den öffentlichen Schlüssel (e,n) und den privaten Schlüssel (d,n) erzeugt. Um eine Nachricht m zu verschlüsseln, wird der Chiffretext folgenderweise berechnet:

$$c = m^e \mod n.$$

Für die Entschlüsselung muss die folgende inverse Berechnung durchgeführt werden:

$$m = c^d \mod n$$

[Buc16, S. 168-171].

Diffie-Hellman-Schlüsselaustausch (DH)

Dieses Verfahren wurde von Diffie und Hellman entwickelt, um Schlüssel über unsichere Verbindungen gesichert auszutauschen. Meist werden die ausgetauschten Schlüssel für symmetrische Verfahren genutzt [Buc16, S. 188]. Die Sicherheit in diesem Verfahren basiert auf der Schwierigkeit des diskreten Logarithmus [Buc16, S. 190].

Definition 2.2.7 (*Diffie-Hellman-Schlüsselaustausch*)

Damit sich zwei Personen Alice und Bob auf einen geheimen Schlüssel K einigen können, müssen sich die beiden zunächst auf eine öffentlich bekannte Zahl g, sowie eine öffentliche Primzahl p einigen.
Alice wählt dann eine geheime Zahl a mit $a \in [0; p-2]$ und bestimmt

$$A = g^a \mod p.$$

Bob wählt analog eine geheime Zahl b mit $b \in [0; p-2]$ und berechnet

$$B = g^b \mod p.$$

Bob und Alice tauschen daraufhin A und B über die abhörbare Verbindung aus, ohne ihre geheimen Zahlen a und b bekannt zu geben. Alice kann anschließend

$$B^a \mod p = g^{ab} \mod p$$

berechnen, wobei Bob

$$A^b \mod p = g^{ab} \mod p$$

berechnet. Beide erhalten damit den geheimen Schlüssel K mit

$$K = g^{ab} \mod p$$

ohne, dass dieser öffentlich bekannt ist [Buc16, S. 189].

2.3 Komplexitätstheorie

Um eine Aussage über die Komplexität von Problemen und die Algorithmen, die sie lösen sollen, zu treffen, ist ein geeignetes Mittel vonnöten. Dieses Mittel wird von der Komplexitätstheorie gestellt, welche in diesem Kapitel zusammenfassend dargelegt wird.

2.3.1 Zeitkomplexität

Die Zeitkomplexität ist eins von zwei Komplexitätsmaßen, welche bei der Abschätzung von Berechnungen genutzt werden. Bei der Zeitkomplexität wird die Anzahl von elementaren Operationen gezählt, die ein Algorithmus für die Lösung eines Problems benötigt. Da jede Berechnung Zeit und Energie kostet, korreliert die Anzahl der Operationen direkt mit diesen beiden Kostenfaktoren [Hro14, S. 168]. Bei der Betrachtung von Laufzeiten muss zunächst eine Funktion f gegeben sein, welche die Anzahl der Operationen beschreibt. Da die Laufzeit eines Algorithmus meist von der Eingabegröße des Problems n abhängt, ist demnach die Betrachtung von $f(n)$ nötig. Um diese Funktion asymptotisch zu analysieren, gibt es die vier verschiedenen Funktionen Ω, O, Θ, o [Hro14, S. 171]. Da in der Kryptographie primär die Maximallaufzeit eines Algorithmus betrachtet wird, soll der Fokus bei der Funktion O liegen.

Definition 2.3.1 (*O-Notation*)

Für jede Funktion $f : \mathbb{N} \to \mathbb{R}^+$ *existiert*

$$O(f(n)) = \{r : \mathbb{N} \to \mathbb{R}^+\}$$

mit einem $n_0 \in \mathbb{N}$ *und einem* $c \in \mathbb{N}$*, sodass für alle*

$$n \geq n_0 : r(n) \leq c \cdot f(n)$$

gilt. Für jede Funktion $r \in O(f(n))$ *gilt, dass sie asymptotisch nicht schneller wächst als* f *[Hro14, S. 171].*

Mithilfe dieser Schreibweise kann die Laufzeit von Algorithmen für die beliebige Problemgröße n verglichen werden. Beispiele für die Anwendung der Schreibweise folgen im Unterkapitel 2.3.3.

2.3.2 Speicherkomplexität

Das andere Komplexitätsmaß bezieht sich auf die maximal genutzte Speichermenge, die ein Algorithmus benötigt, um ein Problem zu lösen, und wird daher als Speicherkomplexität bezeichnet. Dieses Maß agiert analog zur Zeitkomplexität. Für die Formulierung des maximal genutzten Speichers wird dieselbe asymptotische Beschreibung mithilfe der O-Notation durchgeführt. Formal heißt das, dass es eine Funktion f gibt und einen Algorithmus *SPACE*, der die maximale Speichermenge $O(f(n))$ bestimmt, wobei n die Problemgröße bezeichnet [Sip13, S. 331-332].

2.3.3 Komplexitätsklassen

Innerhalb der algorithmischen Probleme gibt es verschiedene Unterteilungen bezüglich der Komplexität des Problems. Die beiden wichtigsten Zeitkomplexitätsklassen sind P und NP, die hier vorgestellt werden. Der ausschlaggebende Faktor für die Zuteilung ist die Zeitkomplexität, die ein Algorithmus besitzt, um ein Problem zu entscheiden. Ein Problem zu entscheiden bedeutet, dass der Algorithmus das Problem entweder löst oder eine definitive Aussage trifft, dass das Problem nicht lösbar ist [Sip13, S. 209]. Da die Komplexitätsklassen normalerweise durch komplexe Turingmaschinen definiert werden, welche allerdings den Rahmen dieser Ausarbeitung überschreiten würden, sind die Definitionen 2.3.2 und 2.3.3 vereinfacht.

Definition 2.3.2 (*Die Komplexitätsklasse P*)

P ist die Komplexitätsklasse, die alle Probleme enthält, welche in polynomieller Zeit entschieden werden können. Die formale Beschreibung lautet:

$$P = \bigcup_k p \mid p \in O(n^k)$$

[Sip13, S. 286].

Hierbei ist anzumerken, dass die Klasse P grob der Klasse von Problemen entspricht, die von einem klassischen Computer berechnet werden können. Zwar kann die Variable k die Berechnungszeit n^k in die Höhe treiben, allerdings hat sich erwiesen, dass die Variable k bei

vielen polynomiellen Algorithmen über die Zeit hinweg gesenkt werden kann. Daher sind diese Probleme in der Praxis effizient lösbar [Sip13, S. 286].

Definition 2.3.3 (*Die Komplexitätsklasse NP*)

NP ist die Komplexitätsklasse, die alle Probleme enthält, welche zeitlich polynomielle Verifizierer haben [Sip13, S. 286].
Ein Verifizierer bestimmt, ob ein Lösungsvorschlag für ein Problem das Problem löst [Sip13, S. 297].

In der Praxis ergeben sich für die Berechnung von NP-Problemen damit Schwierigkeiten. Zum aktuellen Zeitpunkt gibt es keinen anderen Weg, Probleme aus der NP-Klasse zu lösen, als per „Brute-Force“ alle möglichen Antwortmöglichkeiten auszuprobieren und diese zu verifizieren. Daher ist die Klasse auch als

$$NP = \bigcup_k p \mid p \in O(2^{n^k})$$

definierbar. Heutzutage wird allerdings immer noch debattiert, ob die Aussage $NP \neq P$ tatsächlich gilt. Es ist möglich, dass es einen Algorithmus gibt, der NP-Probleme in polynomieller Zeit lösen kann und damit NP auf P reduzierbar macht. Nach aktuellem Forschungsstand ist diese Aussage weder belegbar noch widerlegbar und bleibt damit eins der größten ungelösten Forschungsprobleme der theoretischen Informatik [Sip13, S. 298]. Die Abbildung 2.3 visualisiert die beiden potenziellen Verhältnisse von P und NP.

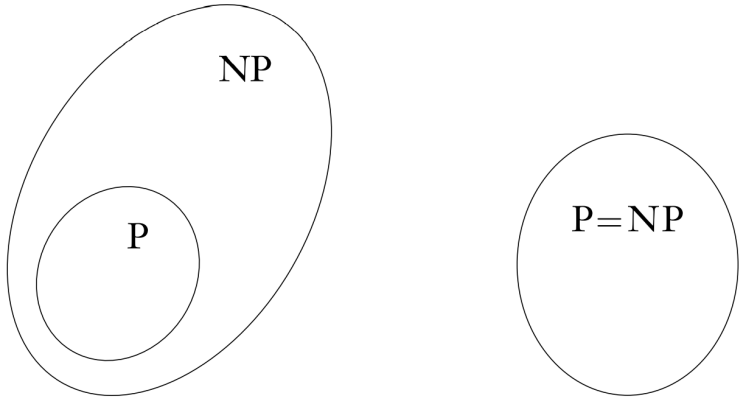

Abbildung 2.3: Die Komplexitätsklassen P und NP als Mengendiagramm dargestellt [Sip13, S. 298]

Die Einteilung von Komplexitätsklassen der Speicherkomplexität unterscheidet sich von den Zeitkomplexitätsklassen. Die Grundlage definiert die Klasse PSPACE, welche das Analogon zur Klasse *P* bildet.

Definition 2.3.4 (*Die Komplexitätsklasse PSPACE*)

PSPACE ist die Komplexitätsklasse, die alle Probleme enthält, welche mit polynomiellem Speicherplatz entschieden werden können. Formell wird die Klasse mithilfe des Algorithmus SPACE beschrieben:

$$PSPACE = \bigcup_k p \mid SPACE(p) \in O(n^k)$$

[Sip13, S. 336].

Im Gegensatz zur Zeit kann Speicher wiederverwendet werden. Daher können Probleme aus der Klasse *NP* mit polynomiellen Speicher entschieden werden. Daher ergibt sich die Relation

$$P \subseteq NP \subseteq PSPACE$$

[Sip13, S. 336].

3 Funktionsweise eines Quantencomputers

Um die Arbeitsweise von Quantenalgorithmen zu verstehen, werden in diesem Kapitel die genutzten Mittel innerhalb eines Quantencomputer erklärt. Dafür ist eine kurze Einführung in die Quantenphysik notwendig, die an dieser Stelle auf das notwendige Wissen für Quantencomputer reduziert und teilweise vereinfacht dargestellt ist. Eine umfassendere Erklärung von Konzepten der Quantenphysik ist in der verwiesenen Literatur zu finden.

3.1 Qubits

Im Gegensatz zu Quantencomputern speichern klassische Computer Informationen in Transistoren, die entweder ein- oder ausgeschaltet werden können und damit die digitalen Zustände 1 und 0 darstellen. Eine Einheit, die entweder 1 oder 0 darstellt, wird „Bit“ genannt. Quantencomputer nutzen allerdings keine klassischen Transistoren, sondern sogenannte „Quantenbits “, welche auch als „Qubits“ bezeichnet werden. Qubits haben grundlegend andere Eigenschaften als das klassische Bit, welche andere Ansätze an die Datenmanipulation erfordern [McM08, S. 11].

Definition 3.1.1 (*Qubit*)

Ein Qubit ist ein Quantenobjekt, welches den Zustand 0 oder 1 annehmen kann. Im Gegensatz zum klassischen Bit kann ein Qubit allerdings auch eine „Superposition“ annehmen, in der das Qubit eine Linearkombination aus allen möglichen Zuständen annimmt. Für die Notation dieses Phänomens wird die Bra-Ket-Notation verwendet.

Wenn ein Qubit einen Zustand $|0\rangle$ oder $|1\rangle$ annehmen kann, existiert das Qubit in einer Superposition $|\Psi\rangle$ mit

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

für $\alpha, \beta \in \mathbb{C}$.

Das Besondere an der Superposition ist, dass sich ein Qubit so lange in einer Superposition befindet, bis der Zustand des Qubits gemessen wird. In dem Moment der Messung verfällt das Qubit zufällig in einen der möglichen Zustände $|0\rangle$ oder $|1\rangle$. Das Quadrat der Koeffizienten α und β gibt dabei die Wahrscheinlichkeit an, dass der jeweilige Zustand eintritt. Das heißt bei einer Messung von $|\psi\rangle$ beträgt die Wahrscheinlichkeit $|\alpha|^2$ das Qubit im Zustand $|0\rangle$ aufzufinden. Analog beschreibt $|\beta|^2$ die Wahrscheinlichkeit für $|1\rangle$. Für Qubits gilt, dass die Summe der Wahrscheinlichkeiten aller Zustände 1 ergeben muss [McM08, S. 11-12].

Prinzipiell können alle quantenmechanischen Eigenschaften als Repräsentation von $|0\rangle$ und $|1\rangle$ dienen, solange sie zwei gegensätzliche Ausprägungen annehmen können. Eine trivialere Eigenschaft, die das Verständnis vereinfacht, ist die Position eines Quantenpartikels. Beispielhaft kann ein Elektron in einer Superposition $|\Psi\rangle$ von zwei verschiedenen Orten sein. $|0\rangle$ kann die Abwesenheit an einem Ort darstellen, sodass $|1\rangle$ die Anwesenheit des Elektrons repräsentiert.

Gleiches kann mit der An- oder Abwesenheit von Photonen in einem optischen Hohlraum erreicht werden, wobei die verschiedenen Quanten Vorteile, als auch Nachteile in der Handhabung mit sich bringen. David P. DiVincenzo hat fünf verschiedene Kriterien an die genutzten Qubits aufgestellt, die für Quantencomputing erfüllt werden müssen, welche in [DiV00] aufgeschlüsselt werden.

Die Bra-Ket-Notation, die bereits in Definition 3.1.1 verwendet wurde, ist die gängige Schreibweise in der Quantenphysik. Die Berechnungen erfolgen in bestimmten Vektorräumen, welche in späteren Unterkapiteln aufgegriffen werden. Die Bra-Ket-Notation gestaltet die Vektorschreibweise übersichtlicher, da ein Ket einen Spaltenvektor zusammenfasst. Für einen 3-dimensionalen Spaltenvektor gibt es einen Ket $|a\rangle$ für den

$$|a\rangle = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

gilt [McM08][S. 15]. Analog gibt es für einen 3-dimensionalen Zeilenvektor einen Bra $\langle b|$ für den

$$\langle b| = \begin{pmatrix} b_1 & b_2 & b_3 \end{pmatrix}$$

gilt [McM08][S. 21]. Beim Ausschreiben dieser Notation kann $|\Psi\rangle$ auch als

$$|\Psi\rangle = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

dargestellt werden. Mithilfe der Vektoraddition und skalaren Multiplikation ist $|\Psi\rangle$ zu

$$|\Psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

umstellbar, was die Lesbarkeit vereinfacht [McM08, S. 18].

3.2 Quantenschaltkreise

Die Lebensspanne von Qubits innerhalb eines Quantencomputers wird durch Quantenschaltkreise beschrieben. Sie dienen als grafische Veranschaulichung von Berechnungen und sind das Analogon zu klassischen Schaltnetzen. Ein Schaltnetz mit zwei Qubits ist in Abbildung 3.1 visualisiert. Auf der linken Seite befinden sich die beiden Qubits q[0] und q[1].

Abbildung 3.1: Ein leerer Quantenschaltkreis mit zwei Qubits

Die Linien rechts der Qubits symbolisieren den zeitlichen Verlauf der Qubits und werden von links nach rechts gelesen, sodass Elemente, die weiter links stehen, früher durchgeführt werden,

als Elemente, die weiter rechts stehen. Zusätzlich zu den Qubits gibt es in der Regel gleich viele klassische Register, welche in der Abbildung als c2 zusammengefasst wurden und über c[0] und c[1] ansteuerbar sind. Diese ermöglichen eine digitale Abbildung von Messergebnissen der Qubits, sodass diese als Bits von klassischen Computern in Form von 0 oder 1 ausgelesen werden können. In der Abbildung 3.2 ist ein Quantenschaltkreis zu sehen, indem beide Qubits zunächst auf $|0\rangle$ gesetzt werden. Daraufhin erfolgt jeweils eine Messung pro Qubit, sodass q[0] auf c[0] und q[1] auf c[1] abgebildet wird [IBMa].

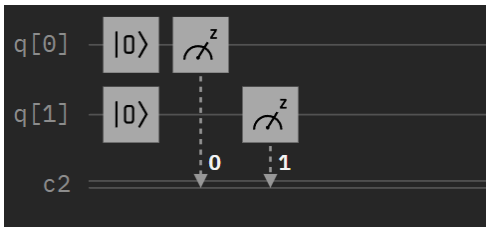

Abbildung 3.2: Ein simpler Quantenschaltkreis mit zwei Qubits, die auf $|0\rangle$ initialisiert und anschließend gemessen werden.

3.3 Quantengatter

Wie in einem klassischen Computer, muss es möglich sein, die Informationen der Qubits zu manipulieren. In Quantencomputern wird das mithilfe von Quantengattern durchgeführt. In der mathematischen Schreibweise sind die Wechselwirkungen durch Quantengatter als Operatoren beschrieben, welche analog zu logischen Operatoren arbeiten.

Definition 3.3.1 (*Operatoren*)

Operatoren im komplexen Vektorraum (Hilbertraum) sind Rechenregeln im Zusammenspiel mit Vektoren und haben die Eigenschaft, dass sie bei einer Anwendung auf ein Ket $|a\rangle$ ein neues Ket $|a'\rangle$ erzeugen. Für einen Operator A gilt

$$A|a\rangle = |a'\rangle$$

[McM08, S. 39]. Die Operatoren, welche beim Quantencomputing genutzt werden, sind hermitesch und einheitlich. Das heißt, dass ein Operator U einen inversen Operator $U^\dagger$ hat, wobei

$$U = U^\dagger$$

gilt. Bei Anwendung auf ein Ket $|a\rangle$ gilt zusätzlich

$$UU^\dagger|a\rangle = U^\dagger U|a\rangle = |a\rangle\,.$$

Daher können Auswirkungen der genutzten Operatoren durch die erneute Anwendung aufgehoben werden [McM08, S. 47-48]. Dieses Phänomen wird „destruktive Interferenz“ genannt [DC20, S. 22].

Innerhalb des Quantencomputings gibt es eine breite Anzahl an Operatoren, die zur Informationsmanipulation genutzt werden. Im Folgenden wird eine Auswahl der wichtigsten Operatoren dargestellt.

3.3.1 Pauli-Matrizen

Die „Pauli-Matrizen“ sind die vier grundlegendsten Operatoren, welche auf Qubits anwendbar sind. Der trivialste ist der Identitätsoperator, welcher als σ_0 oder I notiert wird. In der Matrixschreibweise gilt

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

wobei die Anwendung dieser Matrix das Ket nicht verändert [McM08, S. 41].

Die zweite Pauli-Matrix wird als σ_1, σ_x oder X notiert und beschreibt einen „Bit-Flip“. Dieser Operator ist das quantenmechanische Äquivalent zum logischen NOT-Gate. Bei der Anwendung auf ein Ket ergibt sich

$$X|0\rangle = |1\rangle \;\; bzw. \;\; X|1\rangle = |0\rangle.$$

Die Matrix des X-Operators wird als

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

aufgeschrieben [McM08, S. 41].

Zusätzlich gibt es den σ_2, σ_y bzw. Y-Operator, der ein Ket auf folgende Weise transformiert:

$$Y|0\rangle = -i|1\rangle \;\; bzw. \;\; Y|1\rangle = i|0\rangle,$$

mit einer Matrixschreibweise von

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

[McM08, S. 41].

Der vierte und letzte Operator, der durch die Pauli-Matrizen beschrieben wird, ist der σ_3, σ_z oder Z-Operator. Durch seine Matrix

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

verändert er ein Ket, sodass

$$Z|0\rangle = |0\rangle \;\; bzw. \;\; Z|1\rangle = -|1\rangle,$$

gilt [McM08, S. 41]. Alle Operationen, die ein einzelnes Qubit betreffen, werden wie in Abbildung 3.3 als Schaltkreis notiert. Auf der linken Seite des Gatters wird das eingehende Qubit vermerkt und auf der rechten Seite dasselbe manipulierte Qubit.

Abbildung 3.3: Das X-Gatter als Schaltkreis dargestellt [DC20, S. 34]

3.3.2 Hadamard-Gatter

Die Pauli-Matrizen reichen allerdings nicht aus, um Quantencomputing vorteilhaft zu machen. Die Besonderheiten und Vorteile von Quantencomputern ergeben sich hauptsächlich durch die möglichen Superpositionen. Daher ist ein Gatter notwendig, welche Qubits in Superpositionen versetzen kann oder diese kontrolliert auflöst. Dies wird durch das Hadamard-Gatter bewerkstelligt. Wird das Hadamard-Gatter, welches auch als H abgekürzt wird, auf einen Qubitzustand $|0\rangle$ bzw. $|1\rangle$ angewandt, ergibt sich

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad bzw. \quad H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}},$$

woraus sich erneute Superpositionen mit Koeffizienten $\frac{1}{\sqrt{2}}$ ergeben [McM08, S.198]. Die Matrix des Hadamard-Gatters entspricht

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

[McM08, S.158]. Letzteres kann auch auf ein beliebiges Qubit angewandt werden, unabhängig davon, ob es in einer Superposition ist oder nicht. Für ein Qubit, welches durch $|\Psi\rangle$ beschrieben wird, ergibt die Anwendung des Hadamard-Gatters

$$H|\Psi\rangle = \alpha H|0\rangle + \beta H|1\rangle = \alpha \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) + \beta \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) = \left(\frac{\alpha + \beta}{\sqrt{2}} \right) |0\rangle + \left(\frac{\alpha - \beta}{\sqrt{2}} \right) |1\rangle$$

[McM08, S.198]. Da, wie in Definition 3.3.1 beschrieben wurde, alle Operatoren sich selbst umkehren, gilt dies auch für das Hadamard-Gatter. Wenn für ein $|\Psi\rangle$ $a, b = \frac{1}{\sqrt{2}}$ gilt, ergibt die erneute Anwendung von H

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

3.3.3 Kontrollierte Gatter

Unter ausschließlicher Nutzung von Gattern, welche nur ein Qubit als Eingang nutzen können, sind die Möglichkeiten stark limitiert. Daher gibt es kontrollierte Gatter, welche ein Qubit nur verändern, wenn ein weiteres Kontrollqubit einen bestimmten Zustand hat. Das grundlegendste kontrollierte Gatter ist das CNOT-Gatter.

Unter Angabe von zwei Qubitzuständen $|a\rangle$ und $|b\rangle$, welche als Tensorprodukt $|a\rangle \otimes |b\rangle$ zusammengeführt wurden und somit als $|ab\rangle$ notiert werden, entscheidet der Zustand $|a\rangle$, ob der X-Operator auf $|b\rangle$ angewendet werden soll. Dieses Gatter führt eine kontrollierte Umkehrung durch (Controlled NOT), welche der logischen XOR-Schaltung ähnelt. Das CNOT-Gatter führt die logische Transformation

$$|a, b\rangle \rightarrow |a, b \otimes a\rangle$$

durch. Die Matrix, welche sich für den CNOT-Operator ergibt, ist durch

$$CN = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

gegeben und erzeugt bei Anwendung auf alle möglichen Eingaben die Transformationen

$$CN|00\rangle = |00\rangle$$
$$CN|01\rangle = |01\rangle$$
$$CN|10\rangle = |11\rangle$$
$$CN|11\rangle = |10\rangle$$

Das Schema der kontrollierten Gatter kann mit verschiedenen Operatoren durchgeführt werden. Somit kann der X-Operator aus dem CNOT-Operator, der wie in Abbildung 3.4 sichtbar auf $|b\rangle$ wirkt, z. B. durch einen Z-Operator ausgetauscht werden. Diese Art von Gattern bieten Kontrollstrukturen, welche dem $if/else$ der klassischen Programmierung gleicht [McM08, S.186-187].

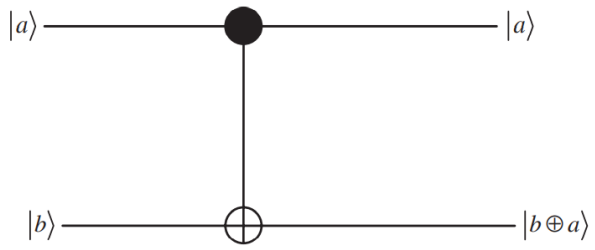

Abbildung 3.4: Das kontrollierte NOT-Gatter (CNOT), als Schaltkreis dargestellt. Der schwarze Kreis repräsentiert den Kontrollpunkt für $|a\rangle$, während $\oplus$ den NOT-Operator darstellt [McM08, S. 187].

3.4 Quantenverschränkung

Konstellationen von mehreren Qubits wurden bisher als Tensorprodukt $|\Psi\rangle = |\Psi_a\rangle \otimes |\Psi_b\rangle$ beschrieben. Das beliebig große Gesamtsystem von Qubits, welche unter dem Tensorprodukt vereint wurden, ist im Inneren allerdings nur eine Zusammensetzung von Bausteinen, welche keinen weiteren Zusammenhang haben [Kas21, S. 212]. Nichts anderes gilt für Berechnungen mit klassischen Bits, welche nur in einem größeren Gefüge zusammengesetzt werden. Die Quantenmechanik erlaubt durch Quantenverschränkungen andere Beziehungen zwischen Qubits.

Werden zwei Qubits $|a\rangle, |b\rangle$ miteinander verschränkt, korrelieren diese stark miteinander. Das heißt, dass die möglichen Zustände von $|a\rangle$ von dem Zustand des Qubits $|b\rangle$ abhängt, was andersherum genauso gilt [Kas21, S. 213]. Wird eines der beiden Qubits gemessen, werden nicht nur Informationen vom Zustand des gemessenen Qubits gewonnen, sondern auch über das verschränkte Qubit. Folglich zerfällt die Superposition beider Qubits gleichzeitig [Kas21, S. 219].

Ein Beispiel für Zustände, die auf Quantenverschränkung basieren, sind die vier „Bell-Zustände“. Sie basieren auf zwei Qubits $|a\rangle$ und $|b\rangle$, welche in den Grundzuständen $|0\rangle$ oder $|1\rangle$ vorzufinden sind. Der erste Bell-Zustand wird auf Grundlage der Qubits $|00\rangle$ erzeugt. Es gilt

$$|00\rangle \rightarrow \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = |\phi^+\rangle$$

[Kas21, S. 213]. Die Qubits liegen in einer Superposition vor, welche bedingt, dass beide Qubits denselben Wert haben. Wird das erste Qubit gemessen und zerfällt zu $|0\rangle$, dann wird eine Messung vom zweiten Qubit immer $|0\rangle$ ergeben, unabhängig davon, wie weit diese beiden Qubits räumlich voneinander entfernt sind [McM08, S. 151]. Die drei anderen Bell-Zustände haben dieselben Eigenschaften, werden allerdings aus den Grundzuständen $|01\rangle, |10\rangle$ und $|11\rangle$ gebildet. Sie werden als

$$|01\rangle \to \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) = \left|\phi^-\right\rangle$$
$$|10\rangle \to \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) = \left|\Psi^+\right\rangle$$
$$|11\rangle \to \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) = \left|\Psi^-\right\rangle$$

definiert [Kas21, S. 213].

Ein Quantencomputer kann zwei oder mehr Qubits mithilfe von Quantengattern miteinander verschränken. Dafür wird das Hadamard-Gatter, sowie das CNOT-Gatter benötigt. Der Ablauf des folgenden Vorgangs ist in Abbildung 3.5 grafisch veranschaulicht. Zunächst müssen die Qubits auf die entsprechenden Werte initialisiert werden. Um den ersten Bell-Zustand zu erreichen, werden beide Qubits auf $|0\rangle$ gesetzt. Daher ergibt sich das Tensorprodukt $|00\rangle$. Daraufhin wird das Hadamard-Gatter auf das erste Qubit angewendet, welches die Transformation zu

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$$

durchführt. Wird daraufhin ein CNOT-Gatter angewendet, sodass das erste Qubit, welches sich in der Superposition befindet, als Kontroll-Qubit genutzt wird, ergibt sich

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = \left|\phi^+\right\rangle,$$

was dem ersten Bell-Zustand entspricht [Kas21, S. 213].

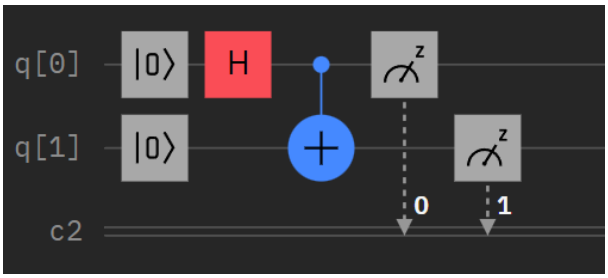

Abbildung 3.5: Die algorithmische Erzeugung des ersten Bell-Zustands als Quantenschaltkreis mit einer anschließenden Messung beider Qubits. Der rote H-Operator repräsentiert das Hadamard-Gatter, während der blaue Operator das CNOT-Gatter darstellt.

3.5 Fehlerkorrektur

Die Verwaltung von Qubits ist bedeutend komplizierter als die von Bits in einem klassischen Computer. In den Unterkapiteln 3.3 und 3.4 wurde bereits aufgezeigt, wie kontrollierte Interaktionen mit Qubits vorgenommen werden können, auch wenn diese in der Praxis nur zu einem bestimmten Grad präzise sein können [DC20, S. 149]. Jedes Qubits hat zusätzlich eine gewisse Wahrscheinlichkeit, mit der Umgebung unkontrolliert zu interagieren. Diese Interaktionen können den Zustand des Qubits verändern und damit auch die Quanteninformationen verfälschen. Dies äußert sich nicht nur in Form von leichten Abweichungen der Koeffizienten α und β in $|\Psi\rangle$, sondern kann auch zum Verfall der erzeugten Superposition des Qubits führen. Weiterhin können Qubits Kohärenz verlieren, welche die Eigenschaft beschreibt, mit anderen Komponenten, wie Qubits oder Quantengattern, zu interagieren [McM08, S. 99]. Dieser Vorgang von Verlust der Kohärenz wird als Dekohärenz bezeichnet. Die dadurch erzeugten Störungen innerhalb des Quantencomputers sind Teil des Quanten-Rauschens [McM08, S. 252, 272]. Für komplexere Berechnungen mit Quantencomputern muss mit dem Quanten-Rauschen umgegangen, sowie Mechanismen entwickelt werden, die Fehler innerhalb dieser Systeme korrigieren können.

Ein naheliegender Ansatz für solch eine Fehlerkorrektur ist die Erzeugung von redundanten Datenblöcken, die groß genug sind, um Fehler zu entdecken und folglich zu korrigieren [DC20, S. 142]. Was in klassischen Computern kein Problem darstellt, da beliebig viele Daten beliebig oft kopiert werden können, ist in dieser Form für Quantencomputer unmöglich. Der Grund dafür ist das No-Cloning-Theorem [McM08, S. 279].

Definition 3.5.1 (*No-Cloning-Theorem*)

Das No-Cloning-Theorem besagt, dass es unmöglich ist, ein beliebiges Qubits $|\Psi\rangle$ perfekt zu kopieren. Sei ein weiteres Qubit $|x\rangle$ gegeben, auf das $|\Psi\rangle$ kopiert werden soll, kann ein Operator U mit

$$U\left(|\Psi\rangle \otimes |x\rangle\right) = |\Psi\rangle \otimes |\Psi\rangle$$

nicht existieren [McM08, S. 280].

Um Fehlerkorrektur durchzuführen, gibt es daher andere Ansätze. Der „Quantenfehlerkorrektur-Code“ (QECC) bietet den umgekehrten Ansatz, indem physikalische Qubits zusammengefasst werden und als Gruppe ein logisches Qubit darstellen. Um n logische Qubits zu nutzen, werden k physikalische Qubits benötigt, wobei $k > n$ gelten muss. Oft wird $k = 3$ genutzt, um einen einfachen Fehler zu entdecken. Ein logisches Qubits $|\Psi_L\rangle$ kann dann als Tensor von drei Qubits in Form von

$$|0_L\rangle = |000\rangle$$
$$|1_L\rangle = |111\rangle$$

dargestellt werden. Aus einem Mehrheitsentscheid der Qubits kann geschlussfolgert werden, welches Qubits korrigiert werden muss. Hierbei muss allerdings berücksichtigt werden, dass nicht nur Fehler erfasst und korrigiert werden dürfen, die durch einen Bit-Flip ausgelöst wurden [DC20, S. 142]. Zusätzlich zum Bit-Flip Fehler, der durch eine unabsichtliche Anwendung des X-Gatters beschrieben werden kann, können in Quantencomputern auch Phasen-Flips geschehen, die einer unabsichtlichen Anwendung des Z-Gatters gleichen [McM08, S. 253]. Da das Prinzip der Lokalisierung und Korrektur von Phasen-Flips, dem der Bit-Flips gleicht, werden nur weitere physikalische Qubits benötigt, um beide Fehlertypen festzustellen [DC20, S. 143].

Einer der bekanntesten Quantenfehlerkorrektur-Codes ist der Neun-Qubit Shor Code. Ein logisches Qubit wird dabei durch neun physikalische Qubits repräsentiert, die in einem 3×3 Gatter angeordnet werden. Bei der Aufteilung in Form von

$$Q_1 Q_2 Q_3$$
$$Q_4 Q_5 Q_6$$
$$Q_7 Q_8 Q_9$$

kann jede Reihe zunächst auf Bit-Flips überprüft werden. Dafür bietet sich das Z-Gatter an. Bei drei Qubits $Q_1 Q_2 Q_3$ kann mit einer Gatterkombination von *ZZI* aus dem Ergebnis abgelesen werden, ob der Fehler in den ersten beiden Qubits vorhanden ist. In diesem Fall muss das Ergebnis der Gatterkombination negativ sein. Wird daraufhin ein weiteres Gatter *IZZ* angewendet, kann aus einem negativen Ergebnis geschlussfolgert werden, dass der Fehler in den hinteren beiden Qubits liegt. Bei Kombination dieser beiden Operationen ist das fehlerhafte Qubit lokalisiert und wird anschließend korrigiert. Ein ähnliches Verfahren wird für die möglichen Phasen-Flips durchgeführt. Der Unterschied ist hierbei, dass die Qubits in Reihen verglichen werden. Das heißt, die erste Reihe, die $Q_1 Q_2 Q_3$ enthält, wird mithilfe von drei X-Gattern mit der zweiten Reihe verglichen. Anschließend passiert selbiges für die zweite und dritte Reihe. Der Vorgang von Lokalisierung und Korrektur gleicht daraufhin prinzipiell dem der Bit-Flips [DC20, S. 143-144]. Die Implementation des Neun-Qubit Shor Codes ist in Abbildung 3.6 als Quantenalgorithmus dargestellt. Es existieren weitere Quantenfehlerkorrektur-Codes, die auf dem Prinzip von Shor aufbauen und es erweitern [DC20, S. 144-150].

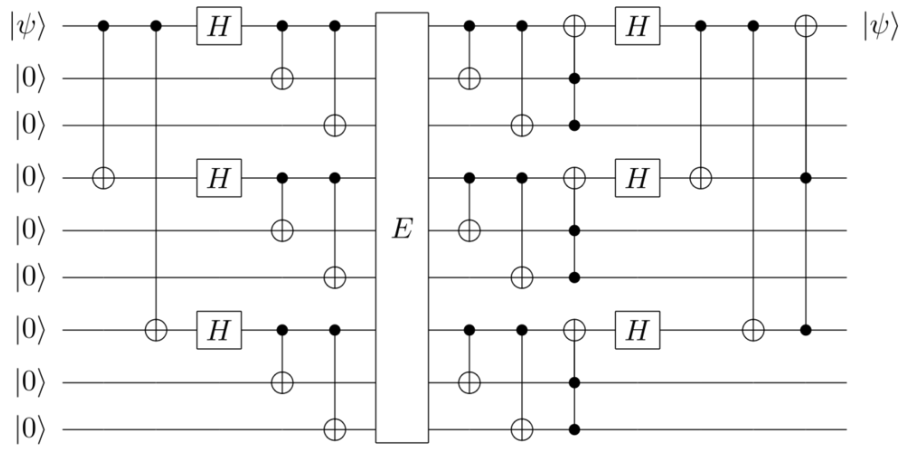

Abbildung 3.6: Die Implementation des Quantenalgorithmus vom Neun-Qubit Shor Code. Zusätzlich zum Hauptqubit sind acht weitere Qubits vonnöten. Im Abschnitt E wird der vermeintliche Error hinzugefügt. Der Operator mit zwei Kontrollblöcken stellt dabei das Toffoli-Gatter dar, welches einem CNOT-Gatter mit einem weiteren Kontrollpunkt gleicht [UK20].

4 Quantenalgorithmen

Nachdem die grundlegende Arbeitsweise von Quantencomputern, sowie die Unterschiede zu klassischen Computern erläutert wurden, kann dieses Wissen genutzt werden, um komplexere Algorithmen auf Quantencomputern auszuführen. In diesem Kapitel wird gezeigt, mit welchen Hilfsmitteln Quantenalgorithmen entwickelt werden und wie es möglich ist, Quantenalgorithmen zu konstruieren, die eine höhere Performanz haben als klassische Algorithmen. Diese werden schlussendlich auf klassische Kryptographieverfahren angewandt.

4.1 Simulationen

Um bedeutende algorithmische Probleme zu lösen und aus ihnen Statistiken zu gewinnen, wird ein Mittel benötigt, um die entwickelten Quantenalgorithmen auszuführen. In der Praxis ist die Forschung noch nicht so weit, als dass Quantencomputer existieren würden, die komplexere Algorithmen ausführen können (siehe Kapitel 5). Daher müssen die Quantenalgorithmen auf klassischen Computern simuliert werden. Im Folgenden wird aufgezeigt, worin sich verschiedene Arten von Simulationen unterscheiden und welche Simulatoren zur Verfügung stehen.

4.1.1 Starke und schwache Simulationen

Simulationen können in viele verschiedene Untergruppen eingeteilt werden. Die Unterscheidung von starken und schwachen Simulationen ist dabei grundlegend. Wird ein Quantenalgorithmus durch einen klassischen Computer simuliert, müssen die Ergebnisse mit hoher Präzision berechnet werden. Unter Angabe eines Quantenschaltkreises S, mit n Eingangqubits und N Quantengattern kann ein Ergebnis berechnet werden, wobei für N in der Regel $O(poly(n))$ gilt. Das Ergebnis der Berechnung ist eine simulierte Messung der Qubits, die einen einzigen Wert aus allen möglichen Ergebnissen liefert. Bei einer binären Repräsentation ist das Ergebnis α eine Bitfolge mit $\alpha \in \{0,1\}^n$. Simulationen, die auf diese Weise arbeiten, werden als schwache Simulationen beschrieben. Sie berechnen nah an der tatsächlichen Umsetzung in einem Quantencomputer, da nur das Ergebnis α mit der Wahrscheinlichkeit $p(\alpha)$ aus einer Wahrscheinlichkeitsverteilung P gewonnen wird [DC20, S. 151-152]. In den folgenden Unterkapiteln wird allerdings deutlich, dass es zahlreiche Quantenalgorithmen gibt, bei denen aus einer einzigen Messung kein bedeutendes Ergebnis geschlussfolgert werden kann.

Im Kontrast dazu stehen die starken Simulationen. Das Ergebnis dieser ist die Wahrscheinlichkeitsverteilung $P(\alpha)$ (oder eine Verteilung, die nah an $P(\alpha)$ liegt), die für alle möglichen Ergebnisse α die Wahrscheinlichkeit $p(\alpha)$ enthält. Starke Simulationen können auch als Sammlung von vielen Iterationen schwacher Simulationen bezeichnet werden. Zusätzlich ist in der Unterscheidung zu beachten, dass nicht alle Quantenalgorithmen mit starken Simulationen effizient simuliert werden können. Wenn eine schwache Simulation eines Quantenschaltkreises einen polynomiellen Zeitbedarf hat und eine exponentielle Anzahl von möglichen Ergebnissen existiert, gilt für dieses algorithmische Problem $p \in O(2^{poly(\alpha)})$ [DC20, S. 152]. Folglich gilt

auch $p \in NP$. Daher kann dieser Quantenalgorithmus nicht effizient mit einer starken Simulation simuliert werden.

Für die Verdeutlichung der Komplexität von Quantensimulationen ist die Matrixschreibweise hilfreich. Ein Schaltkreis S kann als eine Matrix zusammengefasst werden, die auf n Qubits wirkt. Da die Matrix folglich eine $2^n \times 2^n$ Matrix ist und alle N Quantengatter mit der Matrix multipliziert werden müssen, ergeben sich $O(2^{2n})$ Multiplikationen pro Gatter, was zu einer zeitlichen Gesamtkomplexität von $O(N2^{2n})$ führt. Zusätzlich müssen die 2^{2n} komplexen Zahlen gespeichert werden, wozu noch zwei 2^n große Vektoren hinzukommen, um die Ein- und Ausgangszustände darzustellen. Je nach Implementation kann die Speicherkomplexität ebenfalls $O(N2^{2n})$ erreichen. Durch diese Skalierung haben selbst die besten modernen Supercomputer Schwierigkeiten, Schaltkreise mit mehr als 65 Qubits zu simulieren [DC20, S. 155]. An dieser Stelle ist anzumerken, dass die Komplexität der Berechnung drastisch sinkt, wenn sie nur aus einfachen Tensoren besteht und somit keine Quantenverschränkungen genutzt werden. Da in diesem Fall nur die beiden Koeffizienten α und β für jedes Qubit gespeichert werden müssen und die Quantengatter nur die beiden Koeffizienten verändern, ergibt sich ein Speicherbedarf für $2n$ komplexe Zahlen und dementsprechend ein polynomieller Berechnungsaufwand. Dieser Sachverhalt verdeutlicht die Rechenleistung, die Quantencomputer mithilfe von Quantenverschränkungen erreichen können [DC20, S. 156].

4.1.2 Nutzbare Simulatoren

Es existiert bereits eine Bandbreite verschiedenere Quantensimulatoren, die für Interessenten zur Verfügung stehen. Einen vereinfachten Einstieg in die Erzeugung von simplen Quantenschaltkreisen, bietet der „IBM Quantum Composer“ [IBMb]. Ein Ausschnitt der grafischen Oberfläche dieses Simulators ist in Abbildung 4.1 zu sehen.

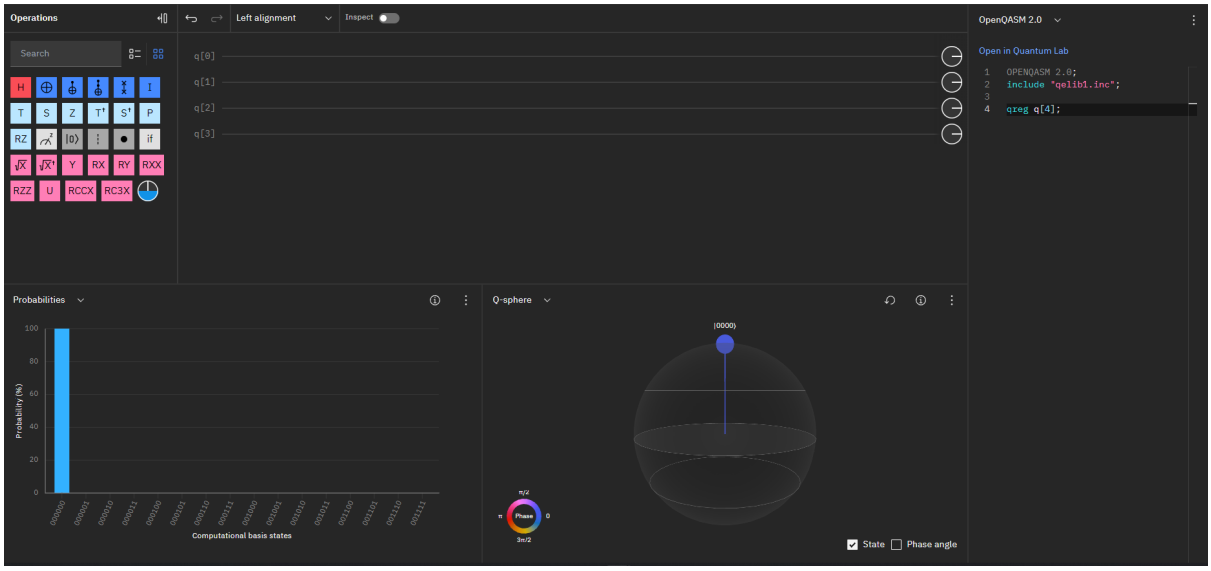

Abbildung 4.1: Die grafische Oberfläche des IBM Quantum Composers

Der Composer ermöglicht die Erzeugung von Quantenschaltkreisen mit zahlreichen Operationen und theoretisch unbegrenzt vielen Qubits. Wenn der Schaltkreis simuliert werden soll, ist die Qubitanzahl allerdings auf 15 begrenzt. Unter Nutzung von maximal sechs Qubits kann eine Live-Simulation der Wahrscheinlichkeitsverteilung $P(\alpha)$ eingesehen werden. Da dies nur eine

Approximation mithilfe pseudozufälliger Zahlen ist, um Quantenzufall zu imitieren, ist die Live-Simulation primär eine Hilfestellung für die Entwicklung des Quantenschaltkreises. Zusätzlich ermöglicht IBM die Ausführung des Schaltkreises auf ihren verfügbaren Quantencomputern, wobei für komplexere Schaltkreise außerdem Supercomputer zur Verfügung stehen, die zu einem gewissen Grad tausende Qubits simulieren können. Die grafische Oberfläche bietet darüber hinaus eine Ein- und Ausgabe des Quantenschaltkreises in Qiskit, ein Software Development Kit für Python [Qis], sowie OpenQASM, eine Open-Source Programmiersprache zur Entwicklung von Quantenalgorithmen, die der klassischen Assembly-Sprache ähnelt [Cro+17].

Für Algorithmen, die im Verlauf dieses Kapitels implementiert werden, wird Qiskit verwendet, da die einfache Syntax von Python die Implementation auf das Wesentliche beschränkt. Die folgenden Implementationsbeispiele sind aus [Qis22] abgeleitet, um einen Einblick in die Möglichkeiten von Qiskit zu geben. Das Ziel dieses Beispielalgorithmus ist die Erstellung eines Quantenschaltkreises, der drei Qubits miteinander verschränkt, um sie in den Greenberger-Horne-Zeilinger-Zustand (GHZ) zu versetzen, der dem ersten Bell-Zustand für drei Qubits gleicht. Bei einer Messung sollen folglich die Qubits im Zustand $|000\rangle$ oder $|111\rangle$ vorzufinden sein, wobei die Wahrscheinlichkeit für beide Zustände bei ungefähr 50% liegen sollen.

```
1 from qiskit import *
2 circ = QuantumCircuit(3)
3 circ.h(0)
4 circ.cx(0, 1)
5 circ.cx(0, 2)
6 print(circ)
```

Codeausschnitt 4.1: Erstellung eines Schaltkreises, der den ersten Bell-Zustand mit drei Qubits erzeugt

Nachdem Qiskit importiert wurde, wird in Zeile 2 im Codeausschnitt 4.1 ein Quantenschaltkreis in *circ* hinterlegt, der drei Qubits enthält. Daraufhin wird ein Hadamard-Gatter auf das erste Qubit angewandt, welches als Kontrollqubit für die folgenden beiden CNOT-Gatter dient, die das zweite und dritte Qubit manipulieren. Wird *circ* über den *print*() Befehl ausgegeben, erfolgt eine visuelle Ausgabe des Quantenschaltkreises auf die Konsole. Diese Ausgabe ist in Abbildung 4.2 zu sehen.

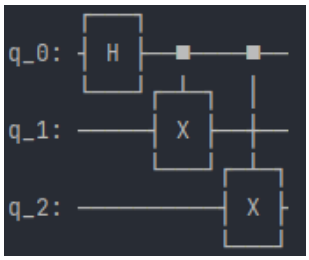

Abbildung 4.2: Konsolenausgabe des Algorithmus aus dem Codeausschnitt 4.1

Mit Qiskit ist es möglich, mehrere Quantenschaltkreise gleichzeitig zu modellieren und diese zusammenzufügen. Um dieses Verhalten zu demonstrieren, wird im Codeausschnitt 4.2 ein Quantenschaltkreis erstellt, der eine Messung an drei Qubits durchführt.

```
1 meas = QuantumCircuit(3, 3)
2 meas.barrier(range(3))
3 meas.measure(range(3), range(3))
4 circ.add_register(meas.cregs[0])
5 qc = circ.compose(meas)
6 print(qc)
```

Codeausschnitt 4.2: Erzeugung eines Messungsschaltkreises, der dem ersten Schaltkreis anhängt

Dieser Schaltkreis wird in *meas* hinterlegt, allerdings enthält er zusätzlich zu den drei Qubits drei klassische Register. In Zeile 2 wird eine Barriere erzeugt, die sich über alle drei Qubits erstreckt. Die Barriere isoliert den hinteren Abschnitt des Schaltkreises, sodass Optimierungen, die Qiskit bei der Laufzeit ausführt, nur lokal innerhalb eines Abschnittes durchgeführt werden. Um die Messgatter einzufügen, wird der Befehl *measure*() genutzt, der die Ausgaben der Messung auf die jeweiligen klassischen Register abbildet. Nachdem der Messschaltkreis fertiggestellt ist, soll er an den ersten Quantenschaltkreis *circ* angehängt werden. Dafür müssen beiden Schaltkreise gleich viele Qubits und klassische Register haben. Daher werden *circ* die nötigen Register in Zeile 4 hinzugefügt. Über den Befehl *compose* wird schließlich *meas* an *circ* angehängt und in *qc* hinterlegt. Die Konsolenausgabe, die *qc* visualisiert, ist Abbildung 4.3 zu entnehmen.

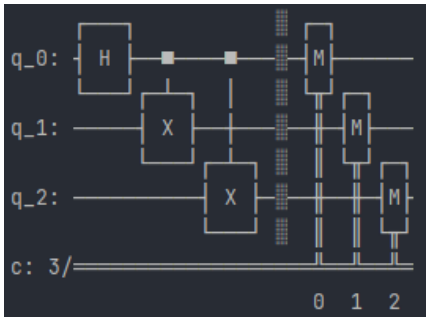

Abbildung 4.3: Konsolenausgabe des Algorithmus aus dem Codeausschnitt 4.2

Um den Quantenschaltkreis zu simulieren, bietet Qiskit verschiedene Implementationen von Simulatoren an. Manche dieser Simulatoren unterstützen die Einbindung von Quanten-Rauschen, welches entweder direkt im Simulator enthalten ist oder manuell durch ein Quanten-Rauschen-Modell eingebunden werden kann. Falls daran Bedarf besteht, z. B. um Simulationen praxisnäher zu gestalten, kann auf diese zurückgegriffen werden.Um die Wahrscheinlichkeitsverteilung zu ermitteln, wird an dieser Stelle der *qasm_simulator* genutzt, der Teil der Aer-Simulatoren ist. Das Spektrum von Simulatoren, das die Klasse *AerSimulator* anbietet, ist in [Tea22a] aufgeschlüsselt.

```
1 backend_sim = Aer.get_backend('qasm_simulator')
2 job_sim = backend_sim.run(transpile(qc, backend_sim), shots=10000)
3 result_sim = job_sim.result()
4 counts = result_sim.get_counts(qc)
5 print(counts)
```

Codeausschnitt 4.3: Erstellung der Simulation zur Messung des Quantenschaltkreises

Nach einer Instanziierung der Simulationsumgebung, wird die Simulation in Zeile 2 des Codeausschnitts 4.3 ausgeführt. Innerhalb des Befehls wird *qc* mithilfe von *transpile*() auf die

Simulationsumgebung optimiert. Der Parameter *shots* gibt an, wie oft Quantenschaltkreise ausgeführt werden sollen. In diesem Beispiel werden 10000 Iterationen durchgeführt, sodass die Wahrscheinlichkeitsverteilung auf 10000 Messergebnissen beruht. Die Simulation gibt ein Objekt *job_sim* zurück, aus welchem unter anderem die Messergebnisse ausgelesen werden können. Der Befehl *get_counts* ordnet die Ergebnisse und erzeugt eine Wahrscheinlichkeitsverteilung. Für eine beispielhafte Ausführung des gesamten Algorithmus ergab sich das Ergebnis {'000': 5006, '111': 4994}.

4.2 Möglichkeiten durch Quantenalgorithmen

In dem Kapitel 3 wurde bereits dargestellt, dass Quantencomputer ähnlich agieren wie klassische Computer. Der zentrale Unterschied ergibt sich aus Superpositionen und den damit möglichen Quantenverschränkungen. Im Unterkapitel 4.1.1 wurde verdeutlicht, dass diese der Grund sind, dass Quantencomputer Berechnungen höherer Komplexität durchführen können. Um die Überlegenheit von Quantencomputing anhand eines Problems zu visualisieren, wird im Folgenden ein Quantenalgorithmus aufgegriffen, der ein Problem schneller lösen kann, als ein analoger klassischer Algorithmus.

Das Deutsch-Jozsa Problem wurde erstmalig in [DJ92], eine Ausarbeitung von David Deutsch und Richard Jozsa aus 1992, erwähnt. Damit wurde das erste Problem vorgestellt, welches schneller durch einen Quantenalgorithmus gelöst werden kann, als durch einen klassischen Computer. Das Problem ist folgendermaßen aufgebaut:

Definition 4.2.1 (*Das Deutsch-Jozsa Problem*)

Es sei eine boolesche Funktion gegeben, welche in einer Blackbox versteckt ist. Die Funktion hat die Form

$$f(\{x_0, x_1, x_2, ...x_n\}) \rightarrow 0 \; oder \; 1, \; mit \; x_n \in \{0,1\},$$

für eine n-Bit lange Eingabe. Das Ziel ist es herauszufinden, ob die versteckte Funktion balanciert ist, also für die Hälfte der Eingaben 0 und für die andere Hälfte 1 zurückgibt, oder konstant ist, also entweder 0 oder 1 für alle Eingaben zurückgibt [Com21a].

Für einen klassischen Computer gibt es keinen besseren Ansatz, als den Algorithmus mehrfach aufzurufen. Wenn beispielsweise $n = 2$ gilt und der Algorithmus auf die Eingabe 00 den Output 0 erzeugt, können zwei Szenarien eingetreten sein. Im ersten Szenario kann die Funktion konstant gewesen sein, sodass die Ausgabe 0 immer erzeugt wird. Das andere Szenario erfordert eine ausgeglichene Funktion, welche auf die Eingabe 00 die Ausgabe 0 erzeugt hat. Tatsächlich erfordert der Worst Case, dass mehr als die Hälfte aller möglichen Eingaben durch die Blackbox geführt werden müssen, um den Typ der booleschen Funktion mit einer Sicherheit von 100 % zu bestimmen. Bei 2^n möglichen Eingaben sind folglich bis zu $2^{n-1} + 1$ Iterationen notwendig [Com21a].

Im Gegensatz dazu kann die Implementation des folgenden Quantenalgorithmus das Deutsch-Jozsa Problem mit einer einzigen Eingabe lösen. Der Algorithmus, der lediglich n+1 Qubits benötigt, ist in Abbildung 4.4 als Quantenschaltkreis visualisiert. Die roten Linien unterteilen den Algorithmus in die folgenden vier Phasen.

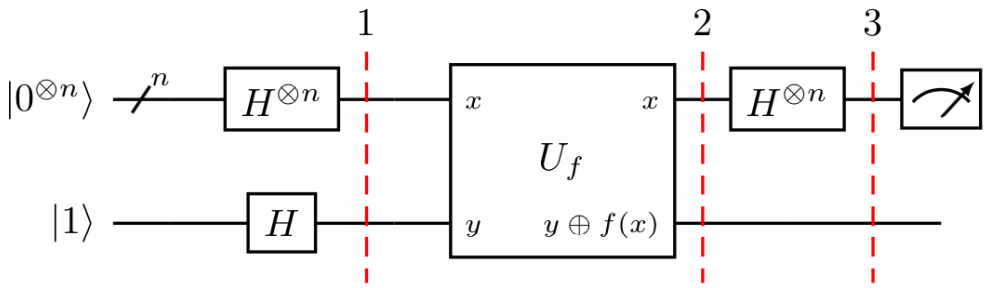

Abbildung 4.4: Der Quantenschaltkreis, der das Deutsch-Jozsa Problem mit einer Eingabe löst. Der U_f Operator stellt das Quanten-Orakel dar, welches alle n+1 Qubits einbindet [Com21a].

Zu Beginn werden die n-Qubits, welche die Eingabe darstellen, auf $|0\rangle$ initialisiert und das Extraqubit auf $|1\rangle$. Daraufhin wird auf jedes Qubit ein Hadamard-Gatter angewendet, sodass sich alle Qubits in einer Superposition befinden [Com21a].

In der darauf folgenden Phase werden die Qubits im Quanten-Orakel zusammengeführt. Das Quanten-Orakel beinhaltet einen Quantenschaltkreis, der äquivalent zur booleschen Funktion ist, die infrage steht. Auch hier kann das Orakel entweder konstant oder ausgeglichen sein. Wenn es konstant ist, ändert sich nur das Extraqubit, sodass die Eingabequbits unberührt sind. In diesem Fall sorgen die *n* Hadamard-Gatter dafür, dass die Eingabequbits in ihren anfänglichen Zustand zurückfallen. Ergibt die Messung in Phase vier n-Bits mit dem Wert 0, muss die Funktion im Orakel konstant sein [Com21a].

Sollte das Orakel ausgeglichen sein, erfolgt jedoch ein anderer Ablauf. Da die Hadamard-Gatter dafür gesorgt haben, dass die Wahrscheinlichkeit der Eingabequbits für alle möglichen 2^n Zustände $\frac{1}{\sqrt{2^n}}$ beträgt, würde ein ausgeglichenes Orakel dafür sorgen, dass die Hälfte aller möglichen Zustände eine negative Phase erhalten, sodass für diese Zustände die Wahrscheinlichkeit $\frac{-1}{\sqrt{2^n}}$ entsteht. Werden daraufhin die Phasen drei und vier durchgeführt, summiert sich die Wahrscheinlichkeit für eine Ausgabe in Form 000...0 auf 0. Das bedeutet, dass das Orakel zwangsläufig ausgeglichen sein muss, wenn eine Ausgabe produziert wird, die nicht nur aus Nullen besteht [Com21a].

4.3 Shor-Algorithmus

Der Algorithmus, der von Peter Shor 1994 in einer vorläufigen Ausarbeitung vorgestellt und 1996 in [Sho96] finalisiert wurde, hat im Gegensatz zum Algorithmus von Deutsch und Jozsa nicht nur das Ziel zu beweisen, wie leistungsstark Quantenalgorithmen sein können, sondern hat eine bedeutende praktische Relevanz. Denn Shors Algorithmus nutzt die Vorteile von Quantencomputing, sodass Faktorisierungen von Zahlen in polynomieller Zeit durchgeführt werden können [Kas21, S. 328]. Außerdem ermöglicht der Algorithmus das effiziente Berechnen des diskreten Logarithmus [PZ04].

4.3.1 Theoretisches Konzept

Im Gegensatz zur gängigen Referenzierung als Faktorisierungsverfahren, ist der Shor-Algorithmus im Grunde nicht in der Lage, Zahlen zu faktorisieren. Der Algorithmus löst allerdings das Problem der Periodenfindung von exponentiellen Funktionen in einer Restklasse [Kas21, S. 328].

Definition 4.3.1 (*Periode in einer Restklasse*)

Sei eine Funktion $f(x) = a^x \mod N$ *gegeben, für die* $a, N \in \mathbb{N}$ *gilt, sowie* $a < N$ *während* a *und* N *keine gemeinsamen Faktoren teilen. Dann existiert eine Periode* r*, sodass für das kleinstmögliche* r *mit* $r \neq 0$

$$a^r \mod N = 1$$

gilt [Com21b].

In Abbildung 4.5 ist beispielhaft die Periode für die Funktion $3^x \mod 35$ dargestellt. In diesem Fall ist schnell sichtbar, dass $r = 12$ gilt. Für große Zahlen kann die Berechnung der Periode auf klassischen Computern sehr anspruchsvoll sein, da alle möglichen Potenzen in der Restklasse ausmultipliziert werden müssen, bis die Periode erreicht ist.

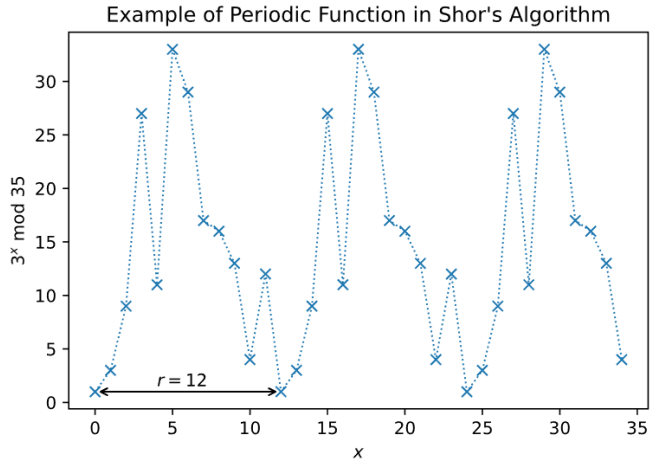

Abbildung 4.5: Die Periode r für die Funktion $3^x \mod 35$ in einem Graphen dargestellt [Com21b].

Der Algorithmus von Shor hingegen kann die Periode wesentlich effizienter bestimmen. Dafür sind zunächst ausreichend viele Qubits vonnöten. Wie Abbildung 4.6 zu entnehmen ist, werden die Qubits in zwei Gruppen unterteilt, welche als Quellregister (source register) und Zielregister (target register) bezeichnet werden. Beide Register enthalten s Qubits. Die Qubits pro Gruppe müssen die Informationen der Potenzen, sowie das Ergebnis der Anwendung dieser Potenz in der Restklasse speichern. Daher gilt bei einem Modulus N

$$N^2 \leq 2^s \leq 2N^2$$

für s Qubits. Um die 2^s Informationen in den Qubits des Quellregisters speichern zu können, werden diese in eine Superposition versetzt, indem s Hadamard-Gatter angewandt werden [Kas21, S. 328-329].

Der Vorteil, der sich durch die Eigenschaften des Quantencomputers ergibt, ist die Parallelität der folgenden Berechnungen. Das Quantenorakel bestimmt gleichzeitig für alle möglichen Zustände

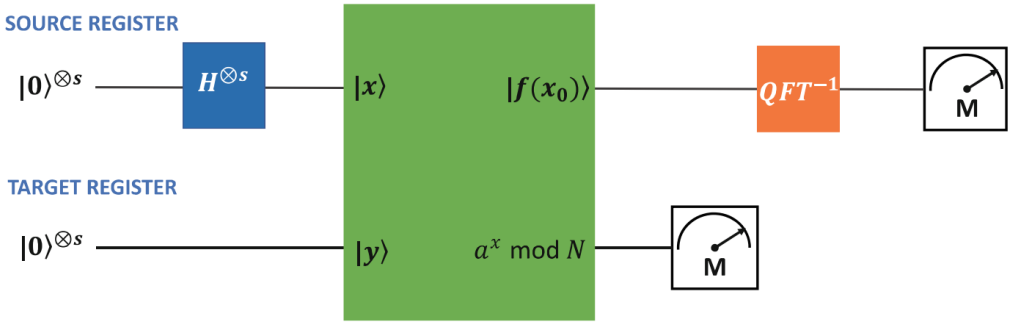

Abbildung 4.6: Der Shor-Algorithmus als Quantenschaltkreis dargestellt. Der orange QFT^{-1} Operator stellt die inverse Quanten-Fouriertransformation dar [Kas21, S. 329].

in $|x\rangle$ aus dem Quellregister, welche alle Zahlen von 0 bis $2^s - 1$ enthalten, die Multiplikation über die Restklasse N und speichert diese in $|y\rangle$ des Zielregisters. Wird $|y\rangle$ an dieser Stelle gemessen, ist ein beliebiges Ergebnis der Funktion $f(x_0)$ bekannt. Zu wissen, was $f(x_0)$ ist, ist zunächst nicht wichtig. Die Regeln der Quantenmechanik sorgen allerdings dafür, dass die Superposition von $|x\rangle$ zu einer kleineren Superposition zerfällt. Alle Zustände, die an dieser Stelle in $|x\rangle$ enthalten sind, erzeugen dasselbe $|y\rangle$, das gemessen wurde. Beim Zurückdenken zur Abbildung 4.5 fällt auf, dass Zustände, die dasselbe $|y\rangle$ ergeben, durch die Periode r oder ein Vielfaches dieser Periode kr getrennt sind. Die Periode ist daher in der Superposition in Form von $x_0 + kr$ kodiert, kann allerdings zu diesem Zeitpunkt nicht abgerufen werden, da nur $f(x_0)$ bekannt ist und nicht x_0. Der aktuelle Zustand von $|x\rangle$ ist in Abbildung 4.7 visualisiert.

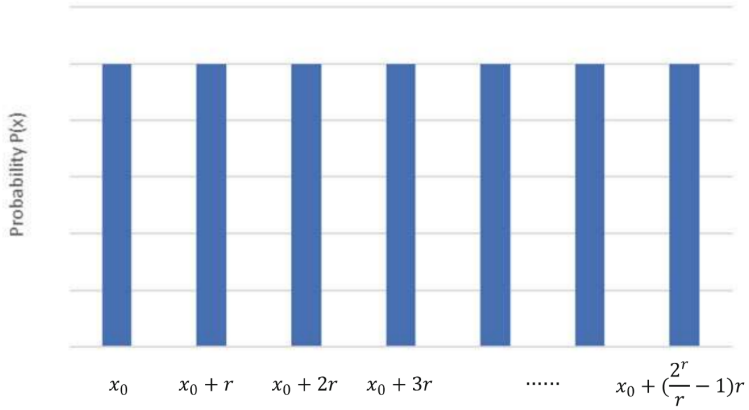

Abbildung 4.7: Der Inhalt von $|x\rangle$ nach Durchführung des Orakels von Shor. Alle möglichen Zustände haben dieselbe Wahrscheinlichkeit und sind durch die Periode r getrennt [Kas21, S. 330].

Die Quanten-Fouriertransformation kann ebenfalls die Darstellung von Qubits in eine Schreibweise abändern, sodass gesuchte Informationen in der Phase einzelner Qubits kodiert sind. Der Ablauf der Quanten-Fouriertransformation, wird in [Sch16, S. 180-188] zusätzlich zur detaillierten Mathematik dieses Abschnitts erklärt. Für den Shor-Algorithmus reicht es zu wissen, dass die Quanten-Fouriertransformation eingegebene Zustände zu einer Frequenz umschreibt. Wird eine Superposition eingegeben, wird eine Superposition von Frequenzen produziert. Da das Shor-Orakel eine Superposition erzeugt hat, die Einträge mit einem Abstand von der Periode r enthält, kann dies auch als Frequenz interpretiert werden. Wenn folglich die inverse Quanten-Fouriertransformation angewandt wird, erzeugt diese eine Superposition von Zuständen. Hierbei

ist die Anzahl der Elemente dieser Superposition relevant, welche durch mehrere Messungen dieses Algorithmus in einer Wahrscheinlichkeitsverteilung dargestellt werden kann. Enthält die Wahrscheinlichkeitsverteilung y Einträge, gilt

$$r = \frac{2^s}{y}$$

[Kas21, S. 336]. Auf diese Weise hat Shors Algorithmus die Periode r wesentlich effizienter berechnet, als es ein klassischer Computer kann [Kas21, S. 331-332].

4.3.2 Shor-Algorithmus im Einsatz gegen RSA

Shor theoretisierte in seiner Ausarbeitung [Sho96] bereits, dass sein Algorithmus genutzt werden kann, um effizient große Zahlen zu faktorisieren. Da die Sicherheit vom RSA Verfahren auf der Schwierigkeit vom Faktorisieren großer Zahlen beruht [Buc16, S. 168], stellt der Shor-Algorithmus eine direkte Bedrohung für das Verfahren dar.

Effiziente Faktorisierung

Große Zahlen zu faktorisieren, ist ein rechenaufwändiges Unterfangen. Die ursprüngliche Methode, welche die Primfaktoren einer Zahl N herausfinden soll, bedient sich am Brute-Force Ansatz, der für alle Primzahlen bis $\sqrt{N}$ überprüft, ob diese N teilen. Draus ergibt sich eine Laufzeit von $O\left(\sqrt{N}\right)$. Für eine Darstellung abhängig zur Bit-Länge d von N, kann die Laufzeit in $O\left(2^{d/2}\right)$ umgeformt werden. Die klassische Methode erfordert daher eine exponentielle Laufzeit [IBMc].

Einen effizienteren Ansatz bietet das Zahlkörpersieb, das für zwei Zahlen a, b $a^2 - b^2$ konstruiert, welches ein Vielfaches von N ergibt. Dann kann $a \pm b$ Zahlen ergeben, die Primfaktoren mit N teilen [IBMc]. Der schnellste Algorithmus aus dem Bereich der Zahlkörpersiebe läuft mit einer Laufzeit von $O\left(e^{1{,}9d^{\frac{1}{3}}(\log d)^{\frac{2}{3}}}\right)$ und ist damit das schnellste Faktorisierungsverfahren, das auf einem klassischen Computer läuft [Kas21, S. 332].

Seit den 1970er Jahren ist bekannt, dass das Faktorisieren einer großen Zahl N einfach ist, wenn die Periode einer exponentiellen Funktion innerhalb der Restklasse N gefunden werden kann. Wenn N genau zwei Primfaktoren hat, also durch

$$N = p_1 p_2$$

definiert ist, dann kann ein zufälliges a gewählt werden, dass zwischen 2 und $N-1$ liegt. Daraufhin wird der größte gemeinsame Teiler (ggT) von N und a bestimmt. Mit hoher Wahrscheinlichkeit wird der Fall $ggT(N,a) = 1$ eingetroffen sein, sodass a und N keine gemeinsamen Primfaktoren teilen. Wenn die Periode r von

$$a^r \mod N = 1$$

bekannt ist, können die gesuchten Primfaktoren von N bestimmt werden, wenn man

$$ggT\left(N, a^{r/2} \pm 1\right)$$

berechnet. Es kann passieren, dass diese Prozedur bei manchen Werten für a den Primfaktor 1 zurückgibt. In der Praxis ist eine richtige bzw. glückliche Auswahl von a nicht selten, da im Durchschnitt nur zwei Anläufe dieser Berechnung benötigt werden, um die richtigen Primfaktoren p_1 und p_2 zu berechnen [IBMc].

Ein klassischer Algorithmus müsste alle möglichen Potenzen von a bis zur Potenz r durchlaufen und die Restklassenmultiplikation dieser Werte ausführen, um die Periode zu ermitteln. Folglich würde sich eine Laufzeit von $O\left(2^d\right)$ ergeben. Dieser Ansatz ist für klassische Computer genauso ineffizient, wie der Brute-Force Ansatz. Da Shor die Periode von a in der Restklasse N in polynomieller Zeit berechnen kann und im Schnitt nur zwei verschiedene Perioden berechnet werden müssen, die jeweils eine Iteration des Algorithmus von Shor erfordern, ergibt sich insgesamt ebenfalls eine polynomielle Laufzeit von $O\left(d^2(\log d)(\log\log d)\right)$ [Kas21, S. 332]. Alle drei genannten Laufzeiten sind in der Tabelle aus Abbildung 4.8 noch einmal gegenübergestellt. Selbst bei mehreren „unglücklichen“ Selektionen für a, also Werte, die bei Anwendung der Formel die Faktoren 1 und N zurückgeben, und einer hohen Anzahl von Iterationen für die Wahrscheinlichkeitsverteilung innerhalb von Shors Algorithmus, bleibt die asymptotische Laufzeit gleich und ermöglicht eine zuverlässige Faktorisierung in polynomieller Zeit. Das Verhältnis der Anzahl von Operationen der Faktorisierung unter Nutzung des Algorithmus von Shor im Gegensatz zur Zahlkörpersieb-Methode ist in Abbildung 4.9 als Graphen visualisiert. Es wird die Effizienz von Shors Algorithmus deutlich, selbst wenn große Werte für d genutzt werden.

Laufzeiten verschiedener Algorithmen zur Faktorisierung			
Algorithmus	Brute-Force	Zahlkörpersieb	Shor-Algorithmus
Laufzeit	$O\left(2^{d/2}\right)$	$O\left(e^{1{,}9d^{\frac{1}{3}}(\log d)^{\frac{2}{3}}}\right)$	$O\left(d^2(\log d)(\log\log d)\right)$

Abbildung 4.8: Ein Laufzeitvergleich von drei Algorithmen, die eine d-Bit lange Zahl N in ihre Primfaktoren zerlegen.

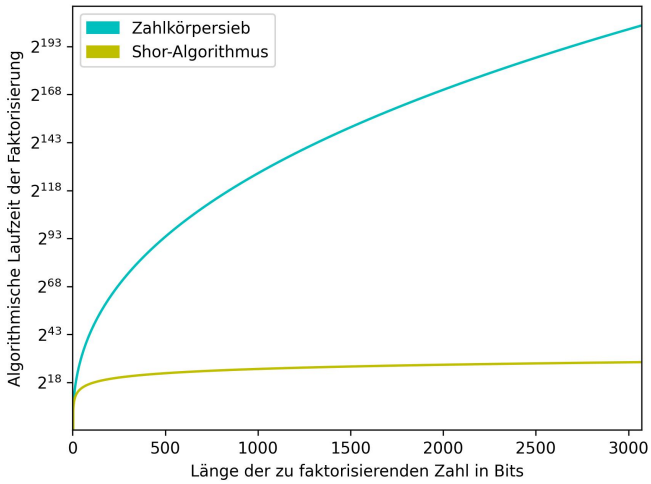

Abbildung 4.9: Die Anzahl von Operationen, die beim Faktorisieren einer Zahl benötigt werden, steigen mit der Länge der Zahl. In diesem Beispiel ist die Länge d in der Binärschreibweise angegeben. Die Graphen visualisieren, dass Shors Algorithmus asymptotisch bedeutend weniger Operationen braucht, als die klassische Methode der Zahlkörpersiebe. Der Brute-Force Ansatz ist aufgrund seiner Ineffizient nicht in der Grafik enthalten.

Beispielhafte Faktorisierung von 15

Der allgemeine Ablauf von Shors Algorithmus und der anschließenden Faktorisierung ist nicht sehr intuitiv. Daher wird im Folgenden die beispielhafte Faktorisierung von 15 vorgenommen. Angefangen wird mit den nötigen Definitionen. Es ergibt sich $N = 15$. Anschließend muss ein a gewählt werden, welches teilerfremd zu N ist. In diesem Beispiel wird dafür $a = 13$ gewählt. Es können auch andere Werte gewählt werden, die aber ggf. nicht immer zur gesuchten Lösung führen. Als letzten Schritt müssen die Register vorbereitet werden. Zur Überschaubarkeit reichen an dieser Stelle Register mit jeweils vier Qubits aus. Bei der Schreibweise

$$|\Psi\rangle = |0000\rangle\,|0000\rangle$$

beschreiben die vorderen vier Qubits das Quellregister und die hinteren das Zielregister, welches im Folgenden auch als $|\Psi_z\rangle$ referenziert wird [Kas21, S. 332-333].

Auf das Quellregister werden daraufhin die vier Hadamard-Gatter angewandt. Das Register enthält folglich alle 2^4 möglichen Qubit-Konstellationen, welche die Zahlen von 0 bis 15 repräsentieren. Da die Wahrscheinlichkeiten für alle Konstellationen gleich sind, ergibt sich jeweils der Koeffizient $\frac{1}{\sqrt{16}}$. Beim Ausschreiben des Quellregisters erhält man

$$|\Psi\rangle = \frac{1}{\sqrt{16}}\left(|0000\rangle + |0001\rangle \ldots + |1111\rangle\right)|0000\rangle\,,$$

was nicht sehr übersichtlich ist. Es wird daher vorläufig als

$$|\Psi\rangle = \frac{1}{\sqrt{16}}\sum_{k=0}^{15}|k\rangle\,|0000\rangle$$

zusammengefasst [Kas21, S. 333].

Im nächsten Abschnitt werden die Potenzen k der Funktion $f(k) = 13^k \mod 15$ berechnet. Dies wird vom Quantenorakel übernommen. Das Ergebnis wird in das Zielregister geschrieben. Die Wertepaare von k und $f(k)$ sind der Tabelle aus Abbildung 4.10 zu entnehmen.

Wertetabelle für $f(k) = 13^k \mod 15$																
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f(k)$	1	13	4	7	1	13	4	7	1	13	4	7	1	13	4	7

Abbildung 4.10: Eine Wertetabelle der Funktion $f(k) = 13^k \mod 15$ mit Werten für k von 0 bis 15.

Wenn diese Multiplikation innerhalb der Restklasse 15 durchgeführt und die Ergebnisse $f(k)$ in das Zielregister geschrieben wurden, haben sich die Registerzustände geändert. Der Einfachheit halber werden die folgenden Qubitzustände als Dezimalzahlen repräsentiert. Der aktuelle Zwischenstand sieht folgendermaßen aus:

$$
\begin{aligned}
|\Psi\rangle &= \frac{1}{\sqrt{16}} \sum_{k=0}^{15} |k\rangle |f(k)\rangle \\
&= \frac{1}{\sqrt{16}} (|0\rangle |f(0)\rangle + |1\rangle |f(1)\rangle + ... + |15\rangle |f(15)\rangle) \\
&= \frac{1}{\sqrt{16}} \begin{pmatrix} |0\rangle |1\rangle + |1\rangle |13\rangle + |2\rangle |4\rangle + |3\rangle |7\rangle + \\ |4\rangle |1\rangle + |5\rangle |13\rangle + |6\rangle |4\rangle + |7\rangle |7\rangle + \\ |8\rangle |1\rangle + |9\rangle |13\rangle + |10\rangle |4\rangle + |11\rangle |7\rangle + \\ |12\rangle |1\rangle + |13\rangle |13\rangle + |14\rangle |4\rangle + |15\rangle |7\rangle \end{pmatrix}
\end{aligned}
$$

Beide Register liegen jetzt in einer Superposition vor, die allerdings in direkter Abhängigkeit zueinander stehen [Kas21, S. 333].

Das Zielregister $|\Psi_z\rangle$ kann auch separat betrachtet werden. In diesem Fall gilt für das Zielregister eine Gleichverteilung aller möglichen Reste, die sich aus der Multiplikation über die Restklasse 15 ergeben haben. Die Verteilung kann als

$$|\Psi_z\rangle = \sqrt{\frac{4}{16}} (|1\rangle + |13\rangle + |4\rangle + |7\rangle)$$

veranschaulicht werden. Erfolgt im Anschluss eine Messung des Zielregisters, fällt die Superposition dieses Registers zusammen. Nur der gemessene Wert ist danach im Zielregister vorzufinden. In diesem Beispiel hat die Messung $|7\rangle$ ergeben. Diese Messung hat allerdings auch das Quellregister beeinflusst. Da 7 als fester Wert für $f(k)$ bestimmt wurde, kann das Quellregister nur noch Werte für k enthalten, mit $f(k) = 7$. Die Register wurden folglich auf

$$|\Psi\rangle = \sqrt{\frac{4}{16}} (|3\rangle + |7\rangle + |11\rangle + |15\rangle) |7\rangle$$

reduziert. Das Ziel ist aus der Superposition von $|3\rangle$, $|7\rangle$, $|11\rangle$, $|15\rangle$ die Abstände zwischen den Einträgen zu gewinnen, da diese die gesuchte Periode r widerspiegeln. Dafür wird die inverse Quanten-Fouriertransformation angewandt. Nach der Anwendung gewinnt man eine Wahrscheinlichkeitsverteilung von den vier Zuständen $|0\rangle$, $|4\rangle$, $|8\rangle$, $|12\rangle$ [Kas21, S. 333-334]. Für die Abschätzung

$$r = \frac{2^s}{y}$$

ist nun $y = 4$ bestimmt, da dies der Anzahl von Einträgen innerhalb der Wahrscheinlichkeitsverteilung entspricht. Da der Wert von s von Beginn an bekannt ist, können beide Werte in die Formel eingesetzt werden. Es ergibt sich $r = 16/4 = 4$. Die Periode ist damit bestimmt.

Für die Ermittlung der Primfaktoren von 15 muss lediglich $ggT\left(N, a^{r/2} \pm 1\right)$ errechnet werden. Durch Einsetzen ergibt sich

$$ggT\left(15, 13^{4/2} \pm 1\right),$$

was trivialerweise berechnet werden kann. Es ergeben sich die Primfaktoren 3 und 5, welche den korrekten Primfaktoren von 15 entsprechen.

Umsetzung in Qiskit

Um das RSA-Verfahren zu brechen, muss aus dem öffentlichen Schlüssel (e,n) und der verschlüsselten Nachricht c der private Schlüssel (d,n) berechnet werden, sodass die ursprüngliche Nachricht m wiederhergestellt werden kann. Da durch das Verfahren bekannt ist, dass n aus zwei Primzahlen p,q besteht und zusätzlich bekannt ist, dass für d die Eigenschaft

$$de \mod (p-1)(q-1) = 1$$

besteht. Wenn die Primfaktoren von n bekannt sind, ist auch der Modulus $(p-1)(q-1)$ bekannt. d kann anschließend durch das multiplikative Inverse in der vorherigen Eigenschaft bestimmt werden. Da durch Shor ein polynomieller Algorithmus zur Faktorisierung bekannt ist, können effizient RSA-Schlüssel berechnet werden.

Für die Umsetzung eines Algorithmus in Qiskit, der die RSA-Schlüsselfindung durchführt, ist eine eigene Programmierung von Shor nicht notwendig. Aus dem Modul *qiskit.algorithms* kann Shor direkt importiert werden. Diese Implementation nutzt $4n+2$ Qubits, wobei n die Anzahl an binären Stellen der eingegebenen Zahl repräsentiert [Tea22c].

```
1  from qiskit import Aer
2  from qiskit.utils import QuantumInstance
3  from qiskit.algorithms import Shor
4
5  def factor_finding_shor(num):
6      backend = Aer.get_backend('aer_simulator')
7      quantum_instance = QuantumInstance(backend, shots=1024)
8      shor = Shor(quantum_instance=quantum_instance)
9      result = shor.factor(num)
10
11     print(f"The list of factors of {num} as computed by the Shor's
           algorithm is {result.factors[0]}.")
12     return result.factors[0]
```

Codeausschnitt 4.4: Eine Funktion, mit der Instanziierung von Shors Algorithmus und einer anschließenden Faktorisierung für eine beliebige Zahl *num* [Tea22c].

Die Primfaktorzerlegung für das n in dem öffentlichen RSA-Schlüssel übernimmt der Code im Codeausschnitt 4.4. Das Schema von Erzeugung eines Backends bis zur Instanziierung eines Quantenschaltkreises (in diesem Fall Shors Algorithmus) gleicht dem aus Unterkapitel 4.1.2. Die Anzahl der Iterationen, mit welchen der Algorithmus ausgeführt wird, bleibt bei dem Standardwert 1024. Diese Genauigkeit reicht für Demonstrationszwecke aus. Würde dieser Algorithmus auf einem Quantencomputer ausgeführt, der groß genug ist, könnten selbst große Zahlen schnell berechnet werden. Die Simulation für die Primfaktorzerlegung von 15 benötigt allerdings ca. 6 Sekunden. Für die Faktorisierung von 55 sind schon 508 Sekunden nötig. Diese Messungen ergeben sich aus einer Ausführung der Skripte auf einem AMD Ryzen 5 5600X. In beiden Fällen ist der klassische Brute-Force Ansatz mit < 0.001 Sekunden bedeutend schneller. Ein Zeitvergleich des klassischen Algorithmus und der Simulation ist daher belanglos und sollte stattdessen auf der asymptotischen Laufzeit der Algorithmen beruhen.

```
1 def rsa_key_break(e, n):
2     factors = factor_finding_shor(n)
3     p = factors[0]
4     q = factors[1]
5     d = pow(e, -1, (p-1)*(q-1))
6     print(f"The public key ({e},{n}) requires the private key to be
          ({d},{n})")
```

Codeausschnitt 4.5: Mithilfe der Faktorisierungsfunktion kann aus dem öffentlichen Schlüssel der private Schlüssel berechnet werden.

Nach Bestimmung der Faktoren p und q wird das multiplikative Inverse von e im Modulus $(p-1)(q-1)$ berechnet. In Zeile 5 des Codeausschnitts 4.5 wird dafür der Befehl $pow()$ mit dem Exponenten -1 genutzt. Damit ist die Berechnung des privaten Schlüssels (d,n) abgeschlossen. Da d genutzt werden kann, um die verschlüsselte Nachricht c zu entschlüsseln, wie es in Unterkapitel 2.2.2 beschrieben ist, ist das RSA Verfahren mithilfe von Shors Algorithmus gebrochen.

4.3.3 Lösung des Diskreten-Logarithmus-Problems

Die Einsatzmöglichkeiten der effizienten Periodenfindung mithilfe Shors Algorithmus enden nicht bei Faktorisierungsverfahren. Manche asymmetrischen Verfahren, wie der Diffie-Hellman-Schlüsselaustausch, basieren auf der Komplexität des Diskreten-Logarithmus-Problems. Hierbei gilt es den diskreten Logarithmus einer Funktion in einer Restklasse zu finden. Dieses Problem kann erneut durch die Kenntnis der Periode vereinfacht werden [PZ04].

Wenn der Kommunikationskanal, der für den Diffie-Hellman-Schlüsselaustausch genutzt wurde, abgehört wird, ist die Basis g, der Modulus p, sowie A und B bekannt. Um das Verfahren zu brechen, müssen die Werte a und b gefunden werden, die $A = g^a \mod p$ bzw. $B = g^b \mod p$ erzeugen. Dafür wird das Orakel von Shors Algorithmus so modifiziert, dass es die Periode von A und B in p findet. Dies repräsentiert die Funktion f für die Perioden x, y mit

$$f(x,y) = A^x B^y$$

in der Restklasse p. Die praktische Umsetzung des Problems muss Shors Algorithmus stark anpassen, da zum einen zwei Sourceregister benötigt werden, um A und B zu halten, zum anderen folglich die Fouriertransformation angepasst werden muss. Eine detailliertere Betrachtung dieser Abänderungen sind [PZ04] zu entnehmen.

Ähnlich zur Faktorisierungs-Alternative bleibt einem klassischen Algorithmus keine Alternative zum Brute-Force Ansatz der gesuchten Zahlen a und b. Mit einer effizienten Einbindung von klassischer Ergebnisaufarbeitung reicht eine einzige Iteration von Shors angepasstem Algorithmus aus, um das Diskrete-Logarithmus-Problem zu lösen, a und b zu berechnen und damit den Diffie-Hellman-Schlüsselaustausch ebenfalls zu brechen [PZ04].

4.4 Grover-Algorithmus

Ein weiterer Algorithmus, der sich quantenmechanischer Eigenschaften bedient und gewisse Berechnungen in Zukunft deutlich vereinfachen könnte, ist der Grover-Algorithmus. In der

Ausarbeitung [Gro96] von Lov K. Grover stellte er einen Quantenalgorithmus vor, der die Suche von Elementen in einer Datenbank beschleunigt.

4.4.1 Theoretisches Konzept

Bei der Suche eines spezifischen Wertes w in einer Menge von N Werten, wie bei einer Datenbank, welche keiner bestimmten Sortierung unterliegt, muss ein klassischer Algorithmus durch alle N Element iterieren, um sie mit w zu vergleichen. Die Laufzeit für solch einen Algorithmus liegt in $O(N)$, wird sich im Durchschnitt allerdings an $\frac{N}{2}$ annähern. Der Quantenalgorithmus von Grover kann stattdessen das Element mit einer Laufzeit in $O\left(\sqrt{N}\right)$ ausfindig machen, wodurch er eine quadratische Beschleunigung verspricht [Gro96].

Für die erfolgreiche Umsetzung von Grovers Suche ist eine grundlegende Voraussetzung zu erfüllen. Die Datenbank, die durchsucht werden soll, muss eine Größe von N besitzen, sodass $N = 2^n$ entspricht. Somit ist es mögliche, alle Zustände in n Qubits zu repräsentieren. Es ist nicht zwangsläufig notwendig, dass das gesuchte Element w einzigartig ist [Kas21, S. 320]. Inwiefern sich Dopplungen von w auf die Laufzeit auswirkt, wird am Ende dieses Abschnitts ersichtlich.

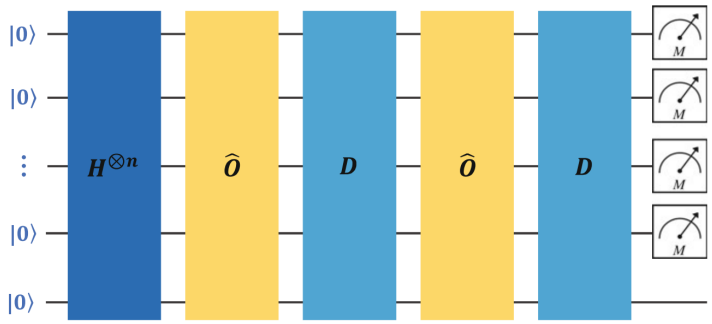

Abbildung 4.11: Der Quantenschaltkreis von Grovers Algorithmus als Blockdiagramm dargestellt. Es sind zwei Iterationen des Orakels $\hat{O}$ und des Diffusionsoperators D enthalten [Kas21, S. 321].

Die Abbildung 4.11 zeigt den Ablauf des folgenden Algorithmus auf. Da Grovers Suche im Gegensatz zu Shors Algorithmus einen Abschnitt enthält, der abhängig von N mehrere Iterationen durchläuft, muss selbst der vereinfachte Quantenschaltkreis auf die Problemgröße angepasst werden. Das Blockdiagramm enthält zwei Iterationen des Abschnitts $\hat{O}$ und D. Der Algorithmus erfordert anfänglich eine Präparierung der Qubits, die eine Initialisierung auf $|0\rangle$ und Anwendung von n Hadamard-Gattern einschließt. Dieser Schritt ist bereits aus Shors Algorithmus bekannt. Anschließend wird das Quantenorakel $\hat{O}$ erstellt [Kas21, S. 320].

Das Ziel des Quantenorakels ist es, alle eingegebenen $|x\rangle$ auf Gleichheit zum gesuchten Element w zu überprüfen. Wird das passende Element gefunden, negiert das Orakel die Amplitude für w. Folglich haben alle anderen $2^n - 1$ Zustände eine Amplitude von $\frac{1}{\sqrt{N}}$, während der Zustand der w repräsentiert eine Amplitude von $-\frac{1}{\sqrt{N}}$ hat. Die Abbildung 4.12 visualisiert die Amplitude der möglichen Zustände nach Anwendung des Orakels für ein $w = |011\rangle$. Zusätzlich ist der Mittelwert der Amplituden als μ eingezeichnet. Bei großen Werten für N weicht μ zunächst kaum von $\frac{1}{\sqrt{N}}$ ab [Kas21, S. 320].

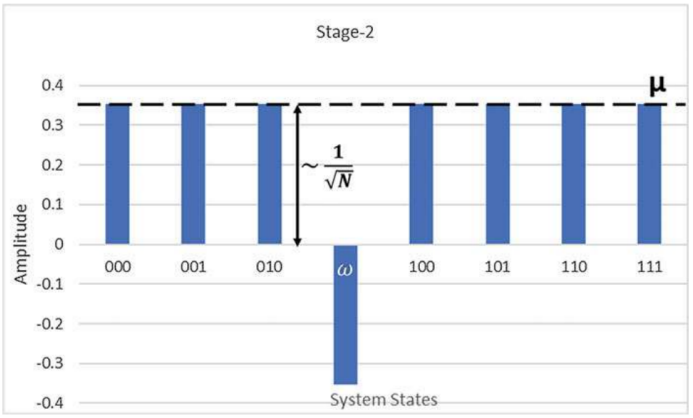

Abbildung 4.12: Die Amplituden aller möglichen Zustände nach Anwendung des Orakels von Grover. Die Amplitude von $w = |011\rangle$ wurde negiert [Kas21, S. 322].

Anschließend wird der Grover-Diffusionsoperator angewandt. Hierbei handelt es sich um eine Veränderung der Amplituden a_x, die der Formel

$$2\mu - a_x$$

folgt. Für alle Zustände ungleich w ergibt sich eine minimale Abschwächung der Amplitude. Dabei erfolgt eine Amplitudenverstärkung für den Zustand w, welche die Amplitude annähernd verdreifacht [Kas21, S. 322]. Das neue Verhältnis der N Amplituden wird in Abbildung 4.13 verdeutlicht.

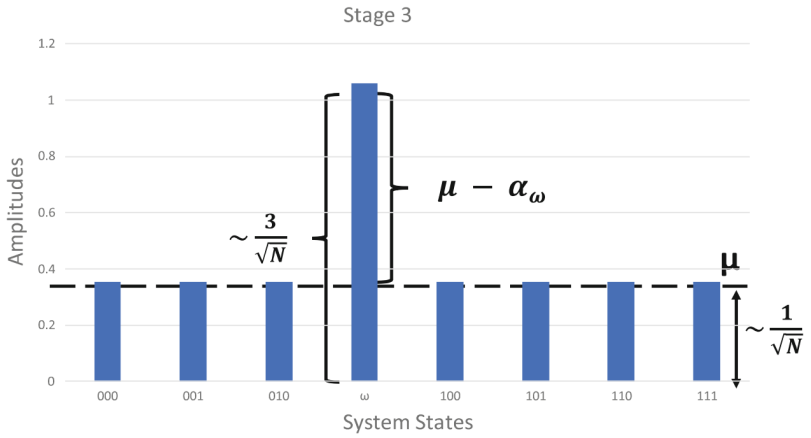

Abbildung 4.13: Nach Anwendung des Grover-Diffusionsoperators hat sich die Amplitude von w nahezu verdreifacht [Kas21, S. 323].

Der Zusammenschluss vom Quantenorakel und Grover-Diffusionsoperator wird auch als Grover-Iteration bezeichnet. Wird die Grover-Iteration oft genug durchgeführt, steigt die Amplitude von w weiterhin, während sich alle anderen Amplituden 0 annähern. Bei einer Messung der Qubits liegt die Wahrscheinlichkeit für w folglich nahe bei 1. Die nötige Anzahl von Grover-Iterationen liegt in $O\left(\sqrt{N}\right)$, woraus sich die entsprechende Laufzeit ergibt. Gibt es k Datenbankeinträge, die w entsprechen, also Dopplungen von w in der Datenbank, reichen $O\left(\sqrt{N/k}\right)$ Iterationen aus [Kas21, S. 322-323].

4.4.2 Implementation im Simulator

Auch für Grovers Suche bietet Qiskit eine schnell umzusetzende Implementation an. Der Quantenschaltkreis von Grover kann wie bei Shor aus der Klasse *qiskit.algorithms* importiert werden. Qiskits Version von Grover unterscheidet sich zu Shors Algorithmus in dem Aspekt, dass Shor nur die Zahl N als Eingabe benötigt, während Grover einen leicht höheren Aufwand der Präparation erfordert. Der folgende Codeausschnitt basiert auf dem Beispiel der Dokumentation von Qiskit [Tea22b].

```
1  from qiskit import QuantumCircuit
2  from qiskit import Aer
3  from qiskit.algorithms import AmplificationProblem
4  from qiskit.algorithms import Grover
5
6  def find_11_with_grover():
7      w = ['11']
8      oracle = QuantumCircuit(2)
9      oracle.cz(0, 1)
10     problem = AmplificationProblem(oracle, is_good_state=w)
11
12     aer_simulator = Aer.get_backend('aer_simulator')
13     grover = Grover(quantum_instance=aer_simulator)
14     result = grover.amplify(problem)
15
16     print('Success!' if result.oracle_evaluation else 'Failure!')
17     print('Top measurement:', result.top_measurement)
```

Codeausschnitt 4.6: Eine mögliche Implementation von Grovers Algorithmus. Im Gegensatz zu Shors Algorithmus, muss das Orakel, welches das Problem spezifiziert, manuell erzeugt werden [Tea22b].

Nach den nötigen Importierungen muss der gesuchte Zustand aus der Datenbank definiert werden. Im Beispiel aus dem Codeausschnitt 4.6 wird ein Element aus einer Datenbank gesucht, welches an $w = 11$ gekoppelt wurde. Dabei kann w z. B. die ID des Elements repräsentieren. Die Datenbank besteht hierbei aus den Einträgen $00, 01, 10$ und 11. Bevor w in den Groverschaltkreis übergeben werden kann, muss das Amplifikations-Problem beschrieben werden, damit Grovers Algorithmus die Amplitude des richtigen Zustands verstärkt. Dies wird in Zeile 10 mithilfe des Aufrufs von $AmplificationProblem()$ durchgeführt. Als Parameter wird das Element w, sowie das vorher erzeugte Orakel, dass den Zustand w als einzig richtigen Zustand markiert, übergeben [Tea22b].

Wenn diese Vorarbeit geleistet ist, kann über das bereits bekannte Verfahren die passende Simulation erzeugt werden. Nach der Instanziierung von Grovers Schaltkreis kann die Groversuche über den Aufruf $grover.amplify()$ durchgeführt werden, wenn als Parameter das entsprechend definierte Problem übergeben wird. Die anschließende Ausgabe besteht aus dem Ergebnis w, wenn w in der Datenbank enthalten ist. Grovers Algorithmus ist damit abgeschlossen [Tea22b].

4.4.3 Anwendung auf symmetrische kryptographische Verfahren

Zwar kann der Suchalgorithmus von Grover durch die effizientere Suche von Datenbanken bereits praktisch relevant werden, allerdings ist auch ein anderweitiger Einsatz denkbar. Durch

einen geschickten Einsatz dieses Algorithmus, können manche symmetrische Verfahren der Kryptographie zwar nicht gebrochen, aber stark abgeschwächt werden.

AES

Kryptographische Verfahren, deren Sicherheit besonders unter den neuen Möglichkeiten durch Grovers Algorithmus leiden, sind Blockchiffren, worunter auch AES fällt. Für die Bestimmung des Schlüssels K, der von AES genutzt wurde, um die gesuchte Nachricht m zur verschlüsselten Nachricht c zu transformieren, ist zunächst eine Voraussetzung zu erfüllen. Eine geringe Anzahl von beispielhaften Paaren werden benötigt, die eine andere Nachricht m_i, sowie den entsprechenden verschlüsselten Text c_i enthalten, welcher durch denselben Schlüssel erzeugt wurde [RS18]. Es werden mehrere Paare benötigt, da nicht garantiert werden kann, dass die verschlüsselten Nachrichten nur durch einen einzigen Schlüssel erzeugt werden können. Bei höheren Schlüssellängen, wie bei AES-192 oder AES-256, muss die Anzahl r an Paaren erhöht werden. Zur Absicherung sollte der Wert von r mindestens $\lceil k/n \rceil$ betragen, wenn k die Länge von K und n die Länge m beschreibt [Jaq+19].

Nachdem diese Voraussetzungen erfüllt sind, kann Grovers Algorithmus angewandt werden. Der gesuchte Zustand, der nach dem Durchlauf der Suche ausgegeben werden soll, ist der Schlüssel K, der für alle r Wortpaare, sowie für die neue verschlüsselte Nachricht m genutzt wurde. Da K nicht explizit bekannt ist, muss das Orakel auf andere Weise konstruiert werden. Die Funktion im Orakel bekommt eine Eingabe von k Qubits in einer Superposition, die alle möglichen 2^k Schlüssel präsentieren sollen. Bei AES-128 müssen bis zu 2^{128} mögliche Schlüssel überprüft werden. Formal wird die im Orakel enthaltende Funktion durch

$$f(K) = (AES_K(m_1) = c_1) \wedge ... \wedge (AES_K(m_r) = c_r)$$

beschrieben. Anders gesagt wird für alle möglichen Schlüssel gleichzeitig berechnet, ob AES unter Nutzung dieses Schlüssels aus allen m_i alle c_i erzeugt. Ist dies der Fall, so wird der Zustand der den Schlüssel repräsentiert als Zustand w markiert. Daraufhin folgen die Grover-Iterationen, welche als Resultat K Verstärkung und zu einer entsprechenden Messung führen [RS18].

Der Sicherheitsgrad von AES bestimmt sich aus der Länge des Schlüssels K und somit aus der Anzahl von möglichen Schlüsseln, die überprüft werden müssen. Mithilfe des eben beschriebenen Verfahrens reduziert sich der Sicherheitsgrad auf die Anzahl nötiger Grover-Iterationen, da diese bestimmen, wann der Schlüssel gefunden wird. Die ursprüngliche Sicherheit von 2^k schwächt Grovers Algorithmus auf $\sqrt{2^k} = 2^{k/2}$ ab. Da der Sicherheitsgrad 128 als ausreichend angesehen wird, gilt AES-128 unter Grovers Suche nicht mehr als sicher. Folglich müssen Implementationen von AES-128 auf AES-256 umgestellt werden, um die Sicherheit auch gegen leistungsstarke Quantencomputer zu gewährleisten [BL17].

Kryptographische Hashfunktionen

Ein ähnliches Verfahren kann genutzt werden, um kryptografische Hashfunktionen abzuschwächen. Der verfolgte Ansatz basiert auf Kollisionen, die in Hashfunktionen auftreten müssen. Das heißt, dass bei einem Hash $h(x)$, welcher aus der Ursprungsnachricht x berechnet wurde, weitere x' existieren müssen, mit $h(x') = h(x)$. Grund dafür ist die feste Länge des Hashes für beliebig

lange Eingaben [GMN05]. Unter Einbezug von Grovers Suche kann ein x' effizienter gefunden werden [RS18].

Um generell Hashfunktionen abzuschwächen, werden als Eingabe des Orakels n Qubits in Form einer Superposition vorbereitet, damit 2^n mögliche Werte für x' überprüft werden können. Innerhalb des Orakels wird für jeden Zustand der Eingabe überprüft, ob der Hash dieser Eingabe $h(x)$ gleicht. Ist dies der Fall, wird der Zustand als w markiert. Der weitere Verlauf gleicht dem bekannten Verfahren [RS18].

Bei Hashfunktionen, die einen Hash der Länge H erzeugen, müssen 2^H mögliche Werte gehasht werden, um eine Kollision zu finden [Pre22]. Eine Implementation mit Grover schwächt die Komplexität des Problems, wie bei AES, auf $2^{H/2}$ ab. Zwei prominente Hashfunktionen sind SHA-256 und SHA3-256, die beide einen Hash der Länge 256 erzeugen. Auch hier gilt, dass diese Verfahren noch sicher sind, da ein Sicherheitsgrad von 128 gegeben ist. Verfahren mit geringeren Hashlängen sind vor Grovers Algorithmus nicht sicher [BL17]. In [Pre22] werden mehrere Implementationen für MD5, SHA-1, SHA-2 und SHA-3 unter einer Ressourcenabschätzung beleuchtet.

MACs

Message-Authentication Codes sind ebenfalls Teil der symmetrischen Kryptographie. Daher ist es nicht verwunderlich, dass mehrere Verfahren genauso von Grovers Algorithmus betroffen sind. Ein besonders offensichtliches Beispiel hierfür ist HMAC, das auf kryptographischen Hashfunktionen beruht. Wie aus dem vorherigen Unterkapitel bekannt ist, kann Grover die effektive Schlüssellänge von Hashfunktionen halbieren. Diese Eigenschaft überträgt sich direkt auf HMAC [HI21].

Solch eine Aussage ist allerdings nicht für alle MACs gültig. Zum einen gibt es bereits MACs, die entwickelt wurden, um nicht anfällig für bekannte Angriffe durch Quantencomputer zu sein. Ein Einblick über die Voraussetzungen und Entwicklung dieser Verfahren bietet [BZ12]. Zum anderen existieren bereits alte Verfahren, wie das von Daniel J. Bernstein entwickelte Poly1305 [Ber05] oder GMAC, die nicht nur als quantensicher gelten, also nach einem Quantenangriff einen Sicherheitsgrad von über 128 behalten, sondern dessen Sicherheitsgrad zu keinem Teil reduziert wird [BL17].

5 Stand der Forschung

Nach Beleuchtung der Theorie von Quantencomputing wird in diesem Abschnitt aufgezeigt, wie weit die Forschung es gebracht hat, diese Prinzipien in der Praxis zu implementieren. Es soll deutlich werden, worin aktuell Probleme bestehen und wann Quantencomputer tatsächlich praktisch relevant sein werden. Schlussendlich wird angerissen, was die Folgen im Bereich Kryptographie sind und wie sich kryptographische Systeme wandeln müssen, um ihre Funktion im Quantenzeitalter zu erfüllen.

5.1 Aktuelle Entwicklungen

Die Forschung hat in den letzten Jahren im Bereich Quantencomputing einen großen Aufschwung erfahren. Auslöser dafür ist die hohe Aufmerksamkeit durch große Firmen wie Google, IBM, Microsoft oder Honeywell, die das große Potenzial in diesem Feld sehen. Daher gibt es bereits zahlreiche Forschungsprojekte, welche die praktische Umsetzung von Quantencomputern realisieren wollen. Darunter fallen allerdings nicht nur Technik-Riesen, sondern auch eine Breite an Forschungsgruppen von Universitäten, sowie zahlreiche Start-up-Unternehmen. Teilweise paaren sich Unternehmen und Universitätsforschungen, mit dem Ziel, Investoren für sich zu gewinnen. Mittlerweile sind 46 Länder in den Forschungsprozessen involviert [Wil21].

Die Ziele der verschiedenen Forschungsteilnehmer setzen große Forschungsfortschritte voraus. 2021 hatte IBM einen Prozessor namens „Eagle“ vorgestellt, welcher aus 127 Qubits besteht. IBM hat das Ziel, 2022 den „Osprey“ Prozessor zu präsentieren. Dieser Chip soll eine Qubit Anzahl von 433 haben, was seinen Vorgänger deutlich in den Schatten stellt. Allerdings plant IBM, bereits 2023 die Qubitanzahl auf 1121 zu erhöhen. Zwar ist die stetige Erhöhung der Anzahl von Qubits wichtig, um die mögliche Komplexität der Berechnung zu erhöhen, jedoch tragen weitere Eigenschaften zur Rechenleistung bei. Dazu zählt nicht nur die Art der Qubits oder wie lange Qubits gehalten werden können, sondern auch die Architektur und somit die Schaltung der Qubits. Um die Vergleiche der Rechenleistung zwischen verschiedenen Quantencomputern zu vereinfachen, hat IBM zwei weitere messbare Metriken eingeführt [Gen21].

Die Metrik der Qualität spiegelt die Größe genutzter Schaltkreise wider, die verlässlich arbeiten. Der Messwert wird als Quantenvolumen bezeichnet. Er ergibt sich aus der Anzahl genutzter Qubits, auch Breite des Quantencomputers genannt, sowie der Tiefe der Quantenschaltkreise, welche durch die Anzahl von Zeitschritten definiert ist, in welchen der Quantencomputer schalten kann, bevor Dekohärenz eintritt. Für die Messung wird ein Algorithmus mit bekanntem Ergebnis ausgeführt, wobei die Zuverlässigkeit des Ergebnisses über $\frac{2}{3}$ betragen muss. Wenn dies der Fall ist, wird die genutzte Breite und Tiefe schrittweise erhöht [Man20]. IBMs „Eagle“-Prozessor erreicht bereits ein Quantenvolumen von 256 [Gam+21].

Zusätzlich hat IBM eine Metrik eingeführt, welche die Geschwindigkeit eines Quantencomputers beschreibt. Die dafür verwendete Einheit heißt CLOPS (Circuit Layer Operations per Second). Dafür wird gezählt, wie viele Schaltkreise ein Quantencomputer pro Sekunde durchschalten kann.

Die gemessene Anzahl korreliert mit der Geschwindigkeit der einzelnen Komponenten, bspw. wie schnell die einzelnen Quantengatter schalten. Da IBM die Zukunft in Quanten-Klassischen Interaktionen sieht, bei welchen ein klassischer Computer Berechnungsaufgaben auf einen Quantencomputer auslagert, spielt auch diese Interaktionsgeschwindigkeit eine wichtige Rolle bei der Anzahl der CLOPS. Beim „Eagle“-Prozessor wurden 1400 CLOPS gemessen [Gam+21].

Bei der Diskussion des aktuellen Forschungsstands kommt des Öfteren der Begriff „Quantum Supremacy“, bzw. Quantenüberlegenheit vor. Der Begriff wird benutzt, wenn ein Quantencomputer fähig sein sollte, eine komplexe Aufgabe schnell auszuführen, für welche ein moderner Hochleistungscomputer unangemessen viel Zeit und Energie bräuchte. Google behauptet seit 2019 die Quantenüberlegenheit erreicht zu haben. In einem Experiment wurden mit 53 Qubits zufällige Zahlen generiert, die im Zahlenbereich von 0 bis 2^{53} liegen. Durch Interferenzen der Operationen kamen gewisse Zahlen öfter vor als andere. Der Quantencomputer konnte die Wahrscheinlichkeiten der Zahlen in 200 Sekunden bestimmen, wobei der Supercomputer „Summit“ von IBM 10000 Jahre brauchen würde [Cho19]. Näheres zu diesem Experiment ist [Aru+19] zu entnehmen.

Google wurde für diese Behauptung allerdings stark kritisiert. IBM meint, dass ihr Supercomputer nur 2,5 Tage brauchen würde, wenn eine andere Herangehensweise an das Problem gewählt werde. Zusätzlich kann der Quantencomputer von Google keine Fehlerkorrektur durchführen, was wahrscheinlich für größere Skalierungen essenziell sein wird. Der Forschungsdirektor von IBM Dario Gil meint, dass dieses Experiment nicht repräsentativ sei, da ein spezielles Problem angegangen wird, das keine praktische Relevanz habe [Cho19]. Er verspricht sich allerdings bereits in den nächsten Jahren einen praktischen Nutzen von Quantencomputern. Es soll dann möglich sein, den Quantenvorteil eines Quantencomputers zu demonstrieren [Gen21].

5.2 Voraussetzungen an Quantencomputer für die praktische Anwendung

Die Roadmap von IBM sieht in den kommenden Jahren einen hohen Anstieg an Qubits vor, sodass Prozessoren mit über einer Million Qubits vorgesehen sind [Gam20]. Wie viele Qubits letztendlich für praktische Anwendungen benötigt werden, ist allerdings noch umstritten. In der Regel wird bei der Diskussion über die benötigte Rechenleistung von Quantencomputern stets ein Vergleich zu aktuellen Supercomputern gezogen. Daher konzentriert sich die Frage nach der angestrebten Rechenleistung meist auf die Quantenüberlegenheit. Die Antwort auf diese Frage ist allerdings nicht trivial [Cho20].

Eine Gruppe von Forschern aus verschiedenen Bereichen der Physik, Mathematik und Informatik hat versucht, eine Antwort zu formulieren, welche über die Komplexitätstheorie abgeleitet wurde. Die detaillierte mathematische Ausarbeitung ist in [Dal+20] dargestellt. Die Approximierung orientiert sich an der Rechenleistung moderner Supercomputer, welche aktuell bis zu 10^{18} FLOPS (Floating-Point Operations per Second) beträgt. Um eine Qubitanzahl anzugeben, welche benötigt wird, um diese Leistung in einem Quantencomputer zu replizieren, ist die Art der Schaltkreise ausschlaggebend. Laut der Forschungsgruppe sind 420 Qubits unter Nutzung der QAOA (Quantum Approximate Optimization Algorithm) Schaltkreise ausreichend. Bei Nutzung von IQP (Instantaneous Quantum Polynomial-Time) Schaltnetzen kann die Qubitanzahl auf 208 reduziert werden, wobei die Nutzung von Photonen in Kombination mit dem „Boson sampling“-Modell die Qubitanzahl auf 98 heruntersetzen kann. Bei diesen Angaben ist zu

beachten, dass viele Worst Case Abschätzungen genutzt wurden. Daher ist es wahrscheinlich, dass vollwertige Implementationen mit weniger Qubits umsetzbar sind [Cho20].

Eine andere Herangehensweise an die benötigte Qubitanzahl ist die Betrachtung spezieller Probleme, welche Quantencomputer lösen sollen. Beispielsweise kann ausgerechnet werden, wie viele Qubits vonnöten sind, um eine Zahl mithilfe von Shors Algorithmus zu faktorisieren. Hierbei ist relevant, wie viele Bits die Zahl zur Darstellung benötigt, da davon die Skalierung des Problems in einem Quantencomputer abhängt. 2003 zeigte Stéphane Beauregard auf, dass für die Anwendung von Shors Algorithmus $2n+3$ Qubits benötigt werden, wobei n die Bitlänge der zu faktorisierenden Zahl darstellt [Bea03]. Eine Forschungsgruppe veröffentlichte allerdings 2016 ein Paper, aus dem hervorgeht, dass die Qubitanzahl auf $n+1$ reduziert werden kann [Mon+16]. Da das Bundesamt für Sicherheit in der Informationstechnik für zukünftige RSA-Implementationen eine Schlüssellänge von 3000 Bits empfehlt [Sic22], werden im Idealfall voraussichtlich mindestens 3001 Qubits nötig sein, um RSA-Verfahren zu entschlüsseln.

Bei diesen Hochrechnungen gibt es allerdings keine Angabe zu der benötigten Zeit, welche der zentrale Faktor der RSA-Verschlüsselung ist. Je mehr Qubits zur Verfügung stehen und je qualitativer diese sind, desto schneller kann Shors Algorithmus aktuelle RSA-Schlüssel brechen. Hochrechnungen vom Bundesamt für Sicherheit in der Informationstechnik ergeben, dass ein 2048-Bit RSA-Schlüssel mithilfe von etwa einer Million physikalischen Qubits innerhalb von 100 Tagen gebrochen werden kann, wenn eine Fehlerrate von 0,01 % gegeben ist. Bei einer Qubitanzahl von einer Milliarde wäre eine Faktorisierung in einer Stunde möglich [Sic20]. Es werden auch Verfahren entwickelt, welche mit störanfälligen Qubits arbeiten. Eines dieser Verfahren soll es möglich machen, einen 2048-Bit RSA-Schlüssel mit 20 Millionen Qubits in nur 8 Stunden zu faktorisieren [GE21].

Damit diese Angaben akkurat bleiben, müssen viele Fortschritte bei der Umsetzung der Hardware geschehen. Wenn keine Störungen der Qubits erfolgt und weitere optimale Bedingungen herrschen, kann echte Quantenüberlegenheit in naher Zukunft erfolgen [Cho20]. Solange dies nicht garantiert werden kann, sind noch viele Verbesserungen durchzuführen. Dazu zählt die Verbesserung von Fehlerkorrekturen, die mit minimalem Hardwareaufwand durchgeführt werden sollten, sowie verbesserte Algorithmen, um hochkomplexe Probleme zu lösen. Das zentrale Problem besteht allerdings darin, robuste Qubits zu entwickeln, die eine möglichst geringe Störanfälligkeit haben, verlässlicher sind und möglichst lange in Kohärenz bleiben können. Wenn diese Probleme gelöst werden, ist auch das Problem der Skalierung von Quantencomputern einfacher zu lösen [Bru19].

5.3 Post-Quanten-Kryptographie

Unabhängig davon, ob es in den nächsten Jahren tatsächlich möglich sein wird, RSA oder andere anfällige Verschlüsselungen mithilfe von Quantencomputern stark abzuschwächen, müssen die Vorbereitungen dafür getroffen werden. Der vorherige Ansatz für die Verstärkung von kryptographischen Verfahren war die Verlängerung der genutzten Schlüssel. Wie aus der Abbildung 5.1 hervorgeht, würde dies für einen Anstieg der benötigten Zeit und Qubits sorgen. Durch die exponentielle Entwicklung der Qubitanzahl müsste allerdings auch die Schlüssellänge asymmetrischer Verfahren linear erhöht werden. Anhand der blauen Graphen in Abbildung 5.1 ist erkennbar, dass das RSA-Verfahren mit einer Schlüssellänge von 3072 in ca. 20 Stunden mithilfe von 40 Megaqubits (ein Megaqubit entspricht einer Million Qubits)

gebrochen werden kann. Eine Verdopplung der Schlüssellänge würde die Dauer auf ca. 85 Stunden erhöhen, wenn 60 Megaqubits zur Verfügung stehen. Wenn genug Qubits genutzt werden können, wäre selbst ein Schlüssel mit der Länge 65536 in nur 3000 Stunden gebrochen. Die Kosten der Verschlüsselungen wären unverhältnismäßig hoch, sodass stattdessen oft neue Verfahren entwickelt oder alte Verfahren abgeändert werden müssen. Das Forschungsgebiet der Post-Quanten-Kryptographie beschäftigt sich mit diesem Unterfangen [BL17].

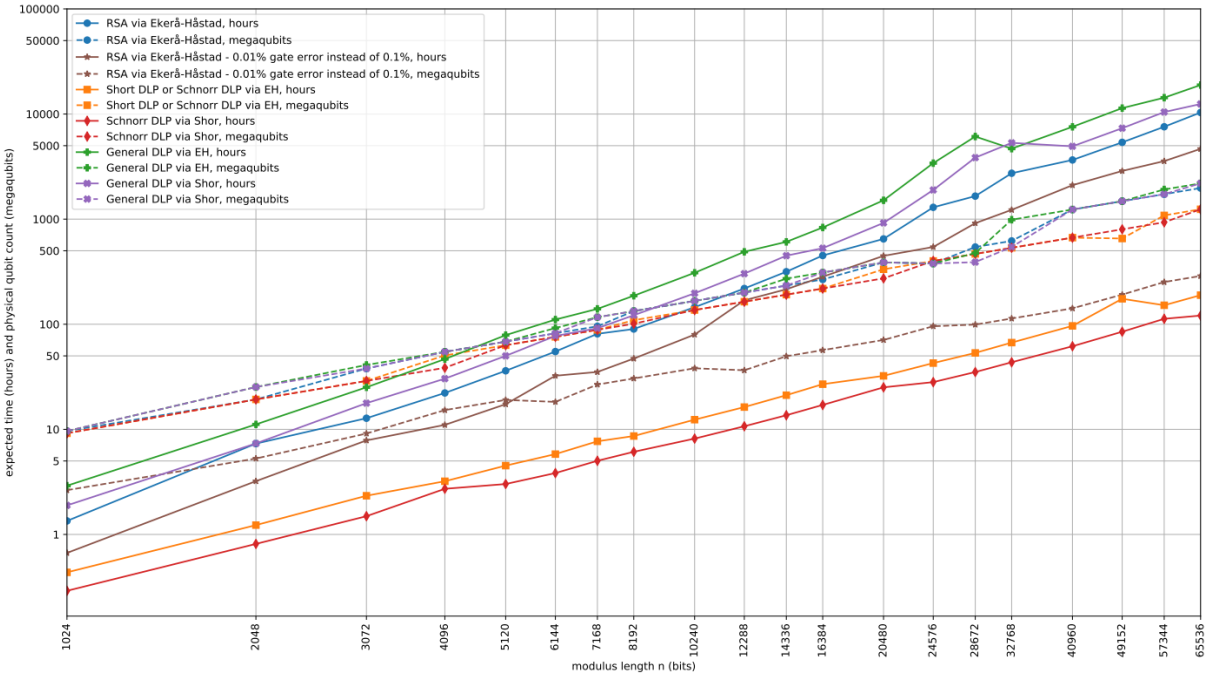

Abbildung 5.1: Die Veränderungen von Kosten der Faktorisierung für verschiedene Schlüssellängen asymmetrischer Verfahren [GE21]. Bei einem exponentiellen Wachstum von Qubits müsste die Schlüssellänge linear ansteigen, um denselben Grad von Sicherheit zu gewährleisten.

„Code-based encryption“ ist eine der Möglichkeiten für Post-Quanten-Kryptographie. Hierbei muss allerdings die Schlüsselgröße komprimiert werden, damit das Verfahren in der Praxis nutzbar bleibt. Dahingehend werden viele neue Varianten von „Code-based encryption“ vorgestellt, wobei McEliece/Niederreiter [McE78] weiterhin der am besten erforschte und daher empfohlene Kandidat bleibt. Als Alternative wurde zusätzlich „NTRU“ [HPS96] vorgeschlagen, welches Teil der „Lattice-based encryption“ ist, da dieses Verfahren mit wesentlich kleineren Schlüsseln arbeitet als McEliece. Zwar ist dieses Verfahren noch nicht gebrochen worden, steht allerdings durch aktuelle Entwicklungen in Gefahr. Durch eine Erweiterung von Shors Algorithmus sind bereits andere „Lattice-based encryption“ gebrochen worden. Zwar war „NTRU“ nicht direkt betroffen, allerdings sind diese Angriffsarten noch nicht gut genug erforscht, sodass dieses System noch nicht mit gutem Gewissen empfohlen werden kann [BL17].

Es gibt bereits viele weitere Systeme, welche sich für die Post-Quanten-Kryptographie eignen. Dazu kommen auch Signaturverfahren, wie „Lattice-based signatures“, „Multivariate-quadratic-equation signatures“ und „Hash-based signatures“. Die Bandbreite an nutzbaren Systemen aus der Post-Quanten-Kryptographie ist mittlerweile groß und wächst stetig. Initiativen von verschiedenen Institutionen fördern die Forschung in diesem Bereich [BL17]. Eins der bekanntesten ist das US National Institute for Standards and Technology (NIST). Dieses hat 2017 einen Wettbewerb gestartet, um aus 82 teilnehmenden Forschungsgruppen, mit verschiedenen

Implementationen von Systemen der Post-Quanten-Kryptographie, über einen rundenbasierten Ansatz die besten Systeme zu ermitteln. Die gewinnenden Verfahren sollen später als Standard in der IT-Sicherheit dienen [Ala+19]. Zum einen beschränkt sich das Ausschreiben auf Verfahren der Public-Key-Kryptosysteme und öffentlichen Schlüsselaustausch. In der aktuellsten Wettbewerbsrunde 3 kandidieren hierfür primär die Verfahren Classic McEliece, CRYSTALS-KYBER, NTRU und SABER. Zum anderen werden auch Verfahren für digitale Signaturen bewertet. Die eingereichten Implementationen von CRYSTALS-DILITHIUM, FALCON und Rainbow haben sich als am vielversprechensten erwiesen. Näheres zu diesen Algorithmen kann von den entsprechenden Websites entnommen werden, welche auf der offiziellen Website der dritten Runde [ST22b] verlinkt sind.

Am 5. Juli 2022 hat NIST die erste Gruppe von Gewinnern bekannt gegeben. Für generelle Verschlüsselungen wurde CRYSTALS-KYBER ausgewählt, während für digitale Signaturen CRYSTALS-DILITHIUM, FALCON und SPHINCS+ ausgezeichnet wurden [ST22a]. Das heißt jedoch noch nicht, dass diese Algorithmen bereits finalisiert sind. Sie werden stetig weiterentwickelt und durch verschiedene Angriffsmethoden auf Robustheit geprüft. Ein Beispiel hierfür ist das vorherige genannte Verfahren Rainbow, welches erst in diesem Jahr durch einen neuen Angriff gebrochen wurde [Beu19], obwohl es in die dritte Runde des Wettkampfs gekommen ist.

5.4 Weitere Einsatzfelder

Die hohe Rechenleistung von zukünftigen Quantencomputern könnte auch in anderen wissenschaftlichen Bereichen revolutionäre Auswirkung haben. Ein zentraler Auslöser ist die neue Herangehensweise an Optimierungsprozesse, die Quantencomputer versprechen. Da Optimierungsprobleme mindestens in der Komplexitätsklasse NP liegen oder sogar schwerer sind und damit in der Klasse NP-Schwer liegen, gab es bisher keine andere Möglichkeit, als über Approximationsalgorithmen die optimale Lösung zu approximieren [Sip13, S. 393-394]. Da Quantencomputer diese Probleme effizienter lösen und daher mehr Faktoren in die Simulation einbeziehen können, werden bessere Vorhersagen getroffen und damit Prozesse mit höherer Genauigkeit optimiert. Diese Entwicklung verspricht speziell für Unternehmen eine gesteigerte Effizienz [Bru19].

Die neuen Möglichkeiten mit komplexeren Simulationen sind allerdings nicht auf Optimierungsprozesse begrenzt. Wie bereits aus der Debatte von Googles angeblicher Quantenüberlegenheit hervorgeht, sind Quantencomputer bereits fähig, Quantensysteme besser zu simulieren, als klassische Computer (siehe Unterkapitel 5.1). Auf einer erweiterten Skala könnten daher komplexe biologische Moleküle und deren Verhalten simuliert werden. Dies kann mit universalen Quantensimulatoren durchgeführt werden, dessen theoretische Konzepte in [Tac+19] aufgegriffen werden. Folglich würde die Genauigkeit bei der Erzeugung und Entwicklung von speziellen Baustoffen oder Chemikalien drastisch ansteigen. Selbiges gilt auch für Medikamente, was neue Ansätze in der Pharmaindustrie ermöglicht, wie z. B. in der Krebsforschung oder für die Forschung von Gentechnik [Bru19]. Da der Prozess von Medikamentenentwicklungen mithilfe von Quantencomputern nicht nur schneller durchführbar sein wird, sondern auch günstiger, können Krankheiten behandelt werden, welche zurzeit kaum Aufmerksamkeit bekommen [Wil21]. Die Einbindung von Quantencomputern in den pharmazeutischen Prozess wird in [EHO21] detaillierter aufgeschlüsselt.

Ein weiterer Bereich, in dem Quantencomputer klassische Supercomputer ablösen werden, ist die Klimaforschung. Auch dieses Gebiet beschäftigt sich mit Simulationen, welche verbessert werden können. Aus den genaueren Berechnungen lassen sich Folgen auf die Umwelt genauer abschätzen und Wettervorhersagen präzisieren. Quantenbasierte Klimamodellierungen könnten uns ein besseres Verständnis des Klimawandels ermöglichen und darüber, wie er am besten zu bekämpfen ist [Wil21].

Im Kontrast zu Simulationen gibt es auch im Bereich der neuronalen Netze und künstlicher Intelligenz Konzepte, um Quantencomputer einzubinden. Die sogenannten Quantenneuronalen Netze (QNN) sind das Produkt aus den Grundlagen der Berechnung mit Quantencomputern und künstlichen neuronalen Netzen (ANN). Für die Implementation von QNNs gibt es bereits verschiedene Varianten, welche die grundlegende Eigenschaft haben, klassische ANNs in mehreren Kriterien zu überbieten. Dazu zählt zunächst, dass QNNs nicht nur effizienter sind, sondern auch höhere Rechenleistung als ANNs für eine große Anzahl von Problemen versprechen. Zusätzlich können QNNs schneller lernen, durch Quantenmechanik parallelisieren und verlässlichere Ergebnisse erzielen, unter erhöhter Stabilität. Dabei kann all dies erreicht werden, während die Anzahl von Neuronen innerhalb der inneren Schichten reduziert werden können [JC18].

6 Ausblick

Aus der Diskussion kristallisiert sich heraus, dass es zahlreiche Konzepte gibt, um Quantencomputer effizient in den Prozess zur Abschwächung von Kryptographieverfahren einzubinden. Die theoretischen Ausarbeitungen haben durch mathematische Beweise starken Rückhalt und können, wie auch in dieser Ausarbeitung gezeigt wurde, mithilfe von Simulationen in einem praktischen Kontext auf Korrektheit überprüft werden. Die Algorithmen von Shor und Grover stellen dabei die bisherigen Kronjuwelen des Feldes dar und überzeugen durch ihre Möglichkeiten gegen symmetrische, sowie asymmetrische Kryptographieverfahren. Die Suche nach weiteren Algorithmen, die einen ähnlich großen Einfluss wie die eben genannten haben, wird stetig fortgesetzt. Ziel ist es, das Potenzial von Algorithmen des Bereichs Quantencomputing auszuschöpfen und Konzepte zu perfektionieren.

Es zeigt sich allerdings auch, dass die Konzeptionierung eines Quantencomputers eine unglaublich schwierige Aufgabe ist. Der praktische Anteil von Quantencomputing hinkt dem theoretischen Jahrzehnte nach. Die Instabilität von Qubits stellt eine hohe Herausforderung dar, die bisher nur im kleinen Rahmen gelöst werden konnte. So haben Forschungsteams kleine Quantencomputer gebaut, die simple Berechnungen ausführen können. Wie sich allerdings aus dem theoretischen Teil gezeigt hat, werden Quantencomputer erst praktische Relevanz haben, wenn sie einen anderen Grad der Skalierung erreichen. Für kryptographische Relevanz bei aktuellen Verschlüsselungsverfahren werden Millionen Qubits benötigt, welche hoch entwickelter Fehlerkorrektur unterliegen müssen. Dafür müssen noch mehrere Entwicklungsziele erreicht werden.

Ziel ist es robuste Qubits zu entwickeln, die so lange wie möglich in Kohärenz gehalten werden, um selbst Programme mit hoher Laufzeit durchführbar zu machen. Dafür werden Fehlerkorrekturen benötigt, die unter möglichst geringem Ressourcenaufwand allen Störquellen entgegenwirken müssen, während andere Programme auf dem Quantencomputer laufen. Um dem Ziel entgegenzukommen, wird zeitgleich einer Verringerung der Störquellen angestrebt. Das beinhaltet alles von einer besseren Isolierung der Qubits bis zur Erhöhung der Genauigkeit von zentralen Quantengattern. All diese Aspekte müssen perfektioniert werden, bevor es möglich ist, Quantencomputer ansatzweise auf die für Kryptographie benötigte Komplexität zu skalieren.

Wann diese hochkomplexe Aufgabe gelöst wird, kann niemand genau sagen. Seitdem die ersten Bemühungen zur Entwicklung eines Quantencomputers ins Leben gerufen wurden, gab es die „10 Jahres Hypothese“, welche einen hochgradig komplexen und frei programmierbaren Quantencomputer in der nächsten Dekade vorsah [Dya20, S. VIII]. Natürlich hat sich diese Hypothese nie bewahrheitet, da solche Konstruktionen noch immer nicht existieren. Die wahre Antwort auf die Frage lautet daher: Man weiß es nicht. Für einen aktualisierten Ablaufplan von Quantencomputing [MP21] wurde diese Frage jedoch erneut aufgegriffen. 44 Experten sollten ihre persönlichen Einschätzungen zu diesem Thema äußern, indem sie für eine Auswahl von Zeitintervallen angaben, wie wahrscheinlich eine Entwicklung von Quantencomputern ist, die ein 2048-Bit RSA Modul brechen können. Aus dieser Umfrage geht hervor, dass die Experten für einen Zeitraum von 30 Jahren eine Wahrscheinlichkeit > 70 % zuordnen. Ein signifikanter Teil der Befragten gaben für diesen Zeitraum bereits eine Einschätzung von > 99 %. Die Hälfte

der Experten sieht bereits eine Wahrscheinlichkeit von > 50 % vor, wenn der Zeitraum auf 15 Jahre beschränkt wird. Wie auch aus den Einschätzungen deutlich wird, ist der Fortschritt nicht zu verleugnen, da schließlich bereits Quantencomputer mit über 100 Qubits existieren. Trotzdem sind etwaige Prognosen mit Vorsicht zu genießen.

Zusätzlich zum breiten Zuspruch der Entwicklung geben auch starke Kritiker ihre Meinung ab. Beispielsweise formulierte Mikhail I. Dyakonov in [Dya20] seine Bedenken bezüglich der Möglichkeit jener Konstruktion eines solchen angestrebten Quantencomputers. Im Kern behauptet er, dass sich in diesem Feld seit 25 Jahren keine bedeutenden Fortschritte ergeben haben. Er sieht daher keinen Anlass zu glauben, dass sich dies im Rahmen der nächsten 25 Jahre ändern wird. Auf der Suche nach der Antwort, ob Mikhail I. Dyakonov damit recht hat, heißt es abzuwarten.

Sollte es allerdings dazu kommen, dass dieses Wunderwerk der Ingenieurskunst vollbracht wird, sind die Anwendungsgebiete dafür zahlreich. Im Rahmen der Kryptographie muss ein „vollständiger Bruch der digitalen Sicherheit“ nicht gefürchtet werden. Durch die lange Zeitfrist, die der Entwicklung von Quantencomputern bevorsteht, sollte es problemlos möglich sein, Post-Quanten-Kryptographie zu standardisieren, um die Sicherheit und Integrität im digitalen Raum auch im Quantenzeitalter zu sichern.

Literatur

[Ala+19] *Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process*. National Institute of Standards und Technology. Jan. 2019. URL: https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8240.pdf.

[Aru+19] Frank Arute u. a. *Quantum supremacy using a programmable superconducting processor*. In: *Nature* (23. Okt. 2019). DOI: https://doi.org/10.1038/s41586-019-1666-5.

[Bea03] Stéphane Beauregard. *Circuit for Shor's algorithm using 2n+3 qubits*. In: *arXiv* (21. Feb. 2003). URL: https://arxiv.org/pdf/quant-ph/0205095.pdf.

[Ben79] Paul Benioff. *The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines*. In: *Springer* (9. Aug. 1979). URL: https://link.springer.com/content/pdf/10.1007/BF01011339.pdf.

[Ber05] Daniel J. Bernstein. *The Poly1305-AES Message-Authentication Code*. In: *Springer* (2005). URL: https://link.springer.com/content/pdf/10.1007%2F11502760_3.pdf%7D.

[Beu19] Ward Beullens. *Breaking Rainbow Takes a Weekend on a Laptop*. In: *Cryptology ePrint Archive* (7. Sep. 2019). URL: https://eprint.iacr.org/2022/214.pdf (besucht am 08. 07. 2022).

[BL17] Daniel J. Bernstein und Tanja Lange. *Post-quantum cryptography*. In: *Nature* (14. Sep. 2017). URL: https://www.nature.com/articles/nature23461.pdf.

[Bol22] J. Bolkart. *Anzahl der in Quantencomputern erreichten Qubits nach Unternehmen/Organisation von 1998 bis 2021 und Prognose bis 2023*. 2022. URL: https://de.statista.com/statistik/daten/studie/1198694/umfrage/anzahl-der-erreichten-qubits-nach-unternehmen/ (besucht am 26. 01. 2022).

[Bru19] Kate Brush. *quantum supremacy*. In: *Techtarget* (Nov. 2019). URL: https://www.techtarget.com/searchsecurity/definition/quantum-supremacy (besucht am 12. 04. 2022).

[Buc16] Johannes Buchmann. *Einführung in die Kryptographie*. Springer Spektrum, 2016.

[BZ12] Dan Boneh und Mark Zhandry. *Quantum-Secure Message Authentication Codes*. In: *Cryptology ePrint Archive* (2012). URL: https://eprint.iacr.org/2012/606.pdf.

[Car19] Sean Carroll. *Even Physicists Don't Understand Quantum Mechanics*. In: *The New York Times* (7. Sep. 2019). URL: https://www.nytimes.com/2019/09/07/opinion/sunday/quantum-physics.html (besucht am 08. 07. 2022).

[Cho19] Adrian Cho. *IBM casts doubt on Google's claims of quantum supremacy*. In: *Science* (23. Okt. 2019). URL: https://www.science.org/content/article/ibm-casts-doubt-googles-claims-quantum-supremacy (besucht am 09. 04. 2022).

[Cho20] Charles Q. Choi. *How Many Qubits Are Needed for Quantum Supremacy?* In: *IEEE Spectrum* (21. Mai 2020). URL: https://spectrum.ieee.org/qubit-supremacy (besucht am 12. 04. 2022).

[Cho22] Adrian Cho. *Worried that quantum computers will supercharge hacking, White House calls for encryption shift*. In: *Science* (5. Mai 2022). URL: https://www.science.org/content/article/worried-quantum-computers-will-supercharge-hacking-white-house-calls-encryption-shift (besucht am 06. 07. 2022).

[Com21a] The Jupyter Book Community. *Deutsch-Jozsa Algorithm*. 2021. URL: https://qiskit.org/textbook/ch-algorithms/deutsch-jozsa.html (besucht am 09. 06. 2022).

[Com21b] The Jupyter Book Community. *Shor's Algorithm*. 2021. URL: https://qiskit.org/textbook/ch-algorithms/shor.html (besucht am 14. 06. 2022).

[Cro+17] Andrew W. Cross u. a. *Open Quantum Assembly Language*. In: *arXiv* (13. Juli 2017). URL: https://arxiv.org/pdf/1707.03429.pdf.

[Dal+20] Alexander M. Dalzell u. a. *How many qubits are needed for quantum computational supremacy?* In: *arXiv* (30. Apr. 2020). URL: https://arxiv.org/pdf/1805.05224.pdf.

[DC20] Yongshan Ding und Frederic T. Chong. *Quantum Computer Systems*. Morgan & Claypool, 2020.

[Dil12] Clay Dillow. *How It Would Work: Creating a Quantum Computer*. In: *Popular Science* (11. Aug. 2012). URL: https://www.popsci.com/science/article/2012-04/unleashing-unparalleled-power-quantum-computer/ (besucht am 06. 07. 2022).

[DiV00] David P. DiVincenzo. *The Physical Implementation of Quantum Computation*. In: *arXiv* (13. Apr. 2000). URL: https://arxiv.org/pdf/quant-ph/0002077.pdf.

[DJ92] David Deutsch und Richard Jozsa. *Rapid solution of problems by quantum computation*. In: *The Royal Society Publishing* (8. Dez. 1992). URL: https://royalsocietypublishing.org/doi/epdf/10.1098/rspa.1992.0167.

[DL10] Christina Diehl und Marcel Leupp. *Komplexe Zahlen, Ein Leitprogramm in Mathematik*. Jan. 2010. URL: https://ethz.ch/content/dam/ethz/special-interest/dual/educeth-dam/documents/Unterrichtsmaterialien/mathematik/Komplexe%20Zahlen%20(Leitprogramm)/Leitprogramm.pdf (besucht am 06. 05. 2022).

[Dya20] Mikhail I. Dyakonov. *Will We Ever Have a Quantum Computer?* Springer Briefs in Physics, 2020.

[EHO21] Matthias Evers, Anna Held und Ivan Ostojic. *Pharma's digital Rx: Quantum computing in drug research and development*. In: *McKinsey & Company* (18. Juni 2021). URL: https://www.mckinsey.com/industries/life-sciences/our-insights/pharmas-digital-rx-quantum-computing-in-drug-research-and-development (besucht am 29. 04. 2022).

[FP09] William A. Fedak und Jeffrey J. Prentis. *The 1925 Born and Jordan paper "On quantum mechanics"*. In: *American Journal of Physics* (2009). DOI: https://doi.org/10.1119/1.3009634.

[Gam+21] Jay Gambetta u. a. *Driving quantum performance: more qubits, higher Quantum Volume, and now a proper measure of speed*. In: *IBM Research* (1. Nov. 2021). URL: https://research.ibm.com/blog/circuit-layer-operations-per-second (besucht am 10. 04. 2022).

[Gam20] Jay Gambetta. *IBM's roadmap for scaling quantum technology*. In: *IBM Research* (15. Sep. 2020). URL: https://research.ibm.com/blog/ibm-quantum-roadmap (besucht am 12. 04. 2022).

[GE21] Craig Gidney und Martin Ekera. *How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits*. In: *arXiv* (13. Apr. 2021). URL: https://arxiv.org/pdf/1905.09749.pdf.

[Gen21] Edd Gent. *IBM's 127-Qubit Eagle Is the Biggest Quantum Computer Yet*. In: *SingularityHub* (22. Nov. 2021). URL: https://singularityhub.com/2021/11/22/ibms-127-qubit-eagle-is-the-biggest-quantum-computer-yet/ (besucht am 09. 04. 2022).

[GMN05] Praveen Gauravaram, William Millan und Juanma Gonzalez Neito. *Some thoughts on Collision Attacks in the Hash Functions MD5, SHA-0 and SHA-1*. In: *Cryptology ePrint Archive* (2005). URL: https://eprint.iacr.org/2005/391.pdf.

[Gro96] Lov K. Grover. *A fast quantum mechanical algorithm for database search*. In: *arXiv* (19. Nov. 1996). URL: https://arxiv.org/pdf/quant-ph/9605043.pdf.

[Hau06] Philipp Hauer. *Asymmetrische Verschlüsselung - Das Verfahren sowie die Vor- und Nachteile*. 13. Dez. 2006. URL: https://www.philipphauer.de/info/info/asymmetrische-verschluesselung/ (besucht am 08. 07. 2022).

[HI21] Akinori Hosoyamada und Tetsu Iwata. *On Tight Quantum Security of HMAC and NMAC in the Quantum Random Oracle Model*. In: *Cryptology ePrint Archive* (2021). URL: https://eprint.iacr.org/2021/774.pdf.

[HPS96] Jeffrey Hoffstein, Jill Pipher und Joseph H. Silverman. *NTRU: A New High Speed Public Key Cryptosystem*. In: *Security Innovation* (13. Aug. 1996). URL: https://web.securityinnovation.com/hubfs/files/ntru-orig.pdf.

[Hro14] Juraj Hromkovič. *Theoretische Informatik, 5. Auflage*. Springer, 2014.

[IBMa] IBM. *Create your first quantum circuit*. URL: https://quantum-computing.ibm.com/composer/docs/iqx/first-circuit (besucht am 04. 06. 2022).

[IBMb] IBM. *IBM Quantum Composer*. URL: https://quantum-computing.ibm.com/composer.

[IBMc] IBM. *Shor's algorithm*. URL: https://quantum-computing.ibm.com/composer/docs/iqx/guide/shors-algorithm (besucht am 16. 06. 2022).

[Jaq+19] Samuel Jaques u. a. *Implementing Grover oracles for quantum key search on AES and LowMC*. In: *arXiv* (3. Okt. 2019). URL: https://arxiv.org/pdf/1910.01700.pdf.

[JC18] S. K. Jeswal und S. Chakraverty. *Recent Developments and Applications in Quantum Neural Network: A Review*. In: *SpringerLink* (3. Mai 2018). DOI: https://doi.org/10.1007/s11831-018-9269-0.

[Kas21] Venkateswaran Kasirajan. *Fundamentals of Quantum Computing*. Springer, 2021.

[Kni22] Ole Knief. *Einsatz von Quantencomputern zur Abschwächung klassischer Verfahren der Kryptographie*. 2022. URL: https://github.com/ole-knf/Einsatz-von-Quantencomputern-zur-Abschwaechung-klassischer-Verfahren-der-Kryptographie.

[Man20] Ryan F. Mandelbaum. *What Is Quantum Volume, Anyway?* In: *Medium - Qiskit* (20. Aug. 2020). URL: https://medium.com/qiskit/what-is-quantum-volume-anyway-a4dff801c36f (besucht am 10. 04. 2022).

[McE78] R. J. McEliece. *A Public-Key Cryptosystem Based On Algebraic Coding Theory*. In: *Jet Propulsion Laboratory* (Jan. 1978). URL: https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

[McI12] Anne McIlroy. *You don't understand quantum physics? Neil Turok will help you*. In: *The Globe and Mail* (12. Okt. 2012). URL: https://www.theglobeandmail.com/news/you-dont-understand-quantum-physics-neil-turok-will-help-you/article4610576/ (besucht am 06. 07. 2022).

[McM08] David McMahon. *Quantum Computing Explained*. John Wiley & Sons Inc., 2008.

[Mon+16] Thomas Monz u. a. *Realization of a scalable Shor algorithm*. In: *Science* (4. März 2016). URL: https://www.science.org/doi/epdf/10.1126/science.aad9480.

[MP21] Michele Mosca und Marco Piani. *Quantum Threat Timeline Report 2020*. In: *Global Risk Institute* (27. Jan. 2021). URL: https://globalriskinstitute.org/publications/quantum-threat-timeline-report-2020/ (besucht am 05. 08. 2022).

[Pre22] Richard Preston. *Applying Grover's Algorithm to Hash Functions: A Software Perspective*. In: *arXiv* (22. Feb. 2022). URL: https://arxiv.org/pdf/2202.10982.pdf.

[PZ04] John Proos und Christof Zalka. *Shor's discrete logarithm quantum algorithm for elliptic curves*. In: *arXiv* (22. Jan. 2004). URL: https://arxiv.org/pdf/quant-ph/0301141.pdf.

[Qis] Qiskit. *Open-Source Quantum Development*. URL: https://qiskit.org/ (besucht am 31. 05. 2022).

[Qis22] Qiskit. *Getting Started with Qiskit*. 30. Juni 2022. URL: https://qiskit.org/documentation/tutorials/circuits/1_getting_started_with_qiskit.html (besucht am 07. 07. 2022).

[RS18] Martin Roetteler und Krysta M. Svore. *Quantum Computing: Codebreaking and Beyond*. In: *IEEE Xplore* (Sep. 2018). URL: https://ieeexplore.ieee.org/document/8490171.

[Sch16] Wolfgang Scherer. *Mathematik der Quanteninformatik*. Springer, 2016.

[SG12] Rajeev Sobti und G. Geetha. *Cryptographic Hash Functions: A Review*. In: *ResearchGate* (März 2012). URL: https://www.researchgate.net/profile/Geetha-Ganesan/publication/267422045_Cryptographic_Hash_Functions_A_Review/links/549cf6d10cf2b8037138c35c/Cryptographic-Hash-Functions-A-Review.pdf.

[Sho96] Peter W. Shor. *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*. In: *arXiv* (25. Jan. 1996). URL: https://arxiv.org/pdf/quant-ph/9508027.pdf.

[Sic20] Bundesamt für Sicherheit in der Informationstechnik. *Entwicklungsstand Quantencomputer*. research report. Juni 2020. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Quantencomputer/P283_QC_Zusammenfassung-V_1_2.pdf?__blob=publicationFile&v=1.

[Sic22] Bundesamt für Sicherheit in der Informationstechnik. *Kryptographische Verfahren: Empfehlungen und Schlusselllängen*. 28. Jan. 2022. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile.

[Sip13] Micheal Sipser. *Introduction to the Theory of Computation, 3rd Edition*. Skills, 2013.

[ST22a] National Institute Of Standards und Technology. *NIST Announces First Four Quantum-Resistant Cryptographic Algorithms*. 5. Juli 2022. URL: https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms (besucht am 08. 07. 2022).

[ST22b] National Institute Of Standards und Technology. *Post-Quantum Cryptography - Round 3 Submissions*. 2022. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions (besucht am 07. 07. 2022).

[Tac+19] Francesco Tacchino u. a. *Quantum Computers as Universal Quantum Simulators: State-of-the-Art and Perspectives*. In: *Wiley Online Library* (19. Dez. 2019). URL: https://onlinelibrary.wiley.com/doi/full/10.1002/qute.201900052 (besucht am 29. 04. 2022).

[Tea22a] Qiskit Development Team. *AerSimulator*. 30. Juni 2022. URL: https://qiskit.org/documentation/stubs/qiskit.providers.aer.AerSimulator.html (besucht am 07. 07. 2022).

[Tea22b] Qiskit Development Team. *Grover's Algorithm and Amplitude Amplification*. 30. Juni 2022. URL: https://qiskit.org/documentation/tutorials/algorithms/06_grover.html (besucht am 07. 07. 2022).

[Tea22c] Qiskit Development Team. *Shor's algorithms*. 30. Juni 2022. URL: https://qiskit.org/documentation/tutorials/algorithms/08_factorizers.html (besucht am 07. 07. 2022).

[UK20] Quantum Computing UK. *Quantum Error Correction: Shor Code in Qiskit*. 25. Aug. 2020. URL: https://quantumcomputinguk.org/tutorials/quantum-error-correction-shor-code-in-qiskit (besucht am 27. 05. 2022).

[Wil21] Angela Wilkins. *The State of Quantum Computing*. In: *Medium – Rice Ken Kennedy Institute* (19. Juli 2021). URL: https://medium.com/ken-kennedy-institute/the-state-of-quantum-computing-a91bc0d556fa (besucht am 09. 04. 2022).

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der elektronischen Abgabe entspricht.

Ich bin damit einverstanden, dass meine Abschlussarbeit in den Bestand der Fachbereichsbibliothek eingestellt wird.

Hamburg, den 24. August 2022

Ole Knief