Leibniz Universität Hannover

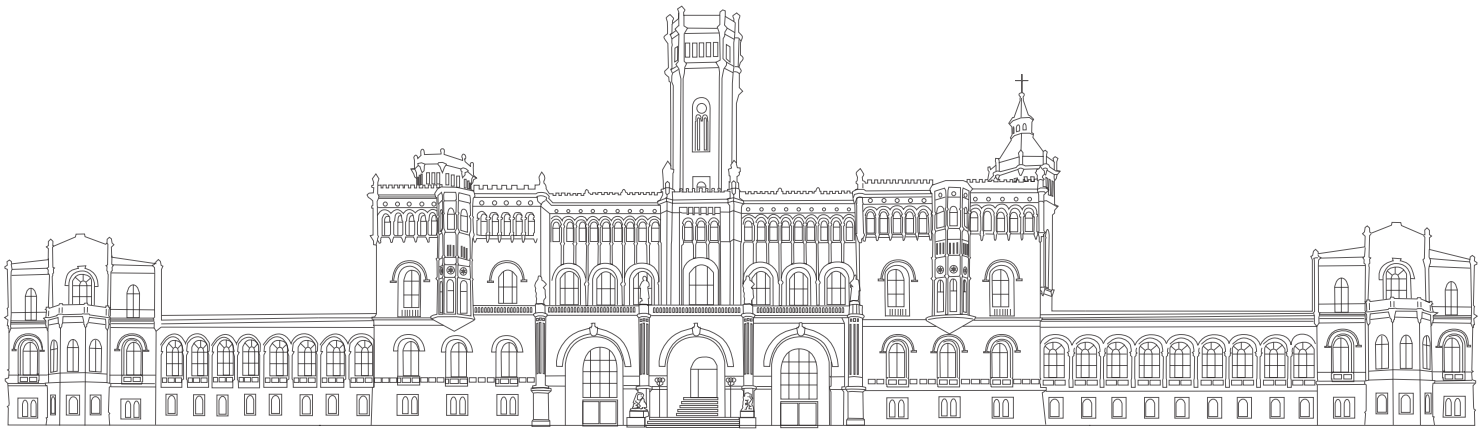
Institut für Theoretische Informatik

Bachelorarbeit im Studiengang Informatik
am Institut für Theoretische Informatik
der Leibniz Universität Hannover

Post-Quantum-Kryptographie: Hash-basierte Verfahren

Hannover, 05. August 2018

Malte Gerriet Hawich
2943360



Erstprüfer:	Prof. Dr. Heribert Vollmer
Zweitprüfer:	Dr. Arne Meier
Betreuer:	Dr. Arne Meier

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die wörtlich oder inhaltlich aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Malte Gerriet Hawich, Hannover, den 5. August 2018

Inhaltsverzeichnis

1. Einleitung

Verschlüsselung ist schon lange ein wesentlicher Bestandteil unseres täglichen Lebens. Sehr früh im Laufe der Geschichte wurden simple Verschlüsselungen benutzt. Bereits im antiken Rom wurde eine simple ROT13 genutzt, auch Cäsar Chiffre genannt, um verschlüsselte Nachrichten zu verschicken [1]. In Kriegen, insbesondere im Zweiten Weltkrieg und im Kalten Krieg, war es von besonderer Bedeutung Informationen sicher zu übertragen. Deswegen wurden verschiedene Verfahren neu erfunden oder bekannte weiter entwickelt, so wie die Enigma [2, 25].

Computer bildeten den nächsten Schritt der Kryptographie. Mit dem Aufkommen von neuen leistungsstarken Rechenmaschinen war es möglich ältere Kryptographiesysteme einfacher zu brechen. Je mehr Leistung die Computer hatten, desto mehr Verschlüsselungen konnten von ihnen gebrochen werden. Der Anspruch an Kryptographiemethoden hat sich also mit der Entwicklung von Computern verändert und es musste ein neuer Schritt in der Entwicklung von Kryptographieverfahren gemacht werden.

Aus diesem Grund wurden neue Verschlüsslungen entworfen, die auf mathematischen Problemen basieren, welche von heutigen Computern nicht effektiv gelöst werden können. Zur Zeit gibt es viele Verschlüsselungen, die unter aktuellen Bedingungen als sicher gelten. Mit der Weiterentwicklung von Quantencomputern ist es möglich, dass sich diese Bedingungen ändern. Dies würde bedeuten, dass dadurch ein weiterer Schritt in der Entwicklung von Computern getan wird und deswegen einen weiterer Schritt in der Entwicklung von Verschlüsselungsalgorithmen gemacht werden muss.

Die Quantenmechanik, auf der diese neuen Quantencomputer basieren, ist noch nicht vollständig erforscht und ihre endgültigen Fähigkeiten können daher auch nur schwer vorausgesagt werden. Diese Bachelorarbeit setzt sich jedoch nicht mit den physikalischen Begebenheiten eines Quantencomputers auseinander, sondern erklärt die Auswirkungen auf die Berechnungen und wie Quantenalgorithmen sich dieses zu Nutzen machen.

Derzeit im Einsatz befindliche Quantencomputer haben gegenüber klassischen Computern noch verhältnismäßig wenig Leistung, da die Zahl der Quantenbits mit denen gerechnet werden kann noch zu gering ist. Es gibt schon einige Veröffentlichungen darüber wie auf Quantencomputern gerechnet werden kann. So stellt Peter Shor einen

Algorithmus vor, der große Zahlen mit Hilfe eines Quantencomputers deutlich schneller faktorisieren kann, als es bisher mit klassischen Computer möglich ist [39]. Folglich werden wir prüfen, ob eine der meist genutzten Verschlüsselung RSA, benannt nach ihren Erfindern Rivest, Shamir und Adlemann [34], weiterhin als sicher angesehen werden kann.

Das Interesse an solchen Kryptographieverfahren hat sich seit der Veröffentlichung des Shor-Algorithmus deutlich erhöht und es gibt bereits einige Ansätze, die sehr vielversprechend sind. Genauso wurden bereits existierende Verschlüsselungen daraufhin geprüft, ob sie einem Angriff durch einen Quantencomputer standhalten könnten.

Um diese Algorithmen zu verstehen, setzen wir uns intensiv mit der Funktionsweise der Quantencomputer in Kapitel 3 auseinander. Um die Effizienz der Quantencomputer und ihrer Algorithmen zu zeigen, beschäftigen wir uns im Grundlagenkapitel 2 mit Komplexitätsmaßen. Die mathematischen Grundlagen und der Bezug zur Turingmaschine befinden sich im Anhang, Kapitel A und B. Nachdem wir sowohl Quantencomputer als die Algorithmen von Grover und Shor in Kapitel 3 und 4 betrachtet haben und uns ihrer Funktionsweise sicher sind, suchen wir in Kapitel 5 nach einer Alternative zu den bekannten Kryptographiesystemen in der Form von Hashverfahren.

2. Grundlagen

2.1. Komplexitätsklassen

Wir nutzen Turingmaschinen, um Algorithmen in ihrer Laufzeit und ihrem Speicherbedarf zu bewerten. Turingmaschinen werden im Anhang in Abschnitt B definiert. Algorithmen werden abhängig von diesen beiden Kriterien in Komplexitätsklassen eingeteilt. Durch diese Klassifizierung lässt sich für eine Klasse immer eine maximale Laufzeit oder ein maximaler Platzbedarf festlegen. So gilt beispielsweise bei der Klasse **P**, dass alle Algorithmen dieser Klasse in der Zeit $\mathbf{TIME}(n^{O(1)})$ ein Ergebnis liefern, also in n^k oder weniger Schritten auf einen deterministischen Turingmaschine eine Entscheidung fällt. Um die Algorithmen einzuordnen, die wir im Rahmen dieser Bachelorarbeit betrachten werden, benötigen wir nur einige gewisse Komplexitätsklassen.

Definition 1. Sei $t\colon \mathbb{N} \to \mathbb{N}$. Die Komplexitätsklasse $\mathbf{TIME}(t)$ besteht aus allen Sprachen A, für die es eine Mehrband-Turingmaschine gibt, die A entscheidet und in Zeit $O(t)$ arbeitet [28].

Definition 2. Die Komplexitätsklasse **P** beinhaltet alle Probleme, die von einer deterministischen Turingmaschine in polynomieller Zeit entschieden werden.

$$\mathbf{P} = \bigcup_{k\in\mathbb{N}} \mathbf{TIME}(n^k) = \mathbf{TIME}(n^{o(1)})$$

Oder anders formuliert **P** ist die *Klasse der effizient lösbaren Probleme*. Effizient bedeutet für uns im Rahmen dieser Bachelorarbeit immer, dass etwas in Polynomialzeit lösbar ist, da Polynome auch bei großen Eingaben noch ein überschaubares Wachstum haben. So bedeutet das immer, dass eine Sprache aus **P** in einer Anzahl von Rechenschritten entschieden werden kann, welche von einem Polynom nach oben beschränkt wird.

Definition 3. Sei $t\colon \mathbb{N} \to \mathbb{N}$. Die Komplexitätsklasse $\mathbf{NTIME}(t)$ besteht aus allen Sprachen A, für die es eine Mehrband-NTM gibt, die A entscheidet und in Zeit $O(t)$ arbeitet [28].

Die Betrachtung des Platzbedarfes der Turingmaschinen ist für uns unerheblich, sie werden jedoch in den beiden platzbedingten Komplexitätsklassen **L** $= \mathbf{SPACE}(\log(n))$ und **NL** $= \mathbf{NSPACE}(\log(n))$ definiert [28].

Da die Klassen **P** und **NP** sich sehr ähnlich sind, erfolgt die Definition zur Klasse **NP** auch analog zur Definiton 2.

Definition 4. Die Komplexitätsklasse **NP** beinhaltet die Probleme, die von einer NTM in polynomieller Zeit entschieden werden können.

$$\mathbf{NP} = \bigcup_{k \in \mathbb{N}} \mathbf{NTIME}(n^k) = \mathbf{NTIME}(n^{o(1)})$$

Ist ein Problem **NP**-schwer, so lässt sich jedes bekannte Problem in **NP** auf dieses in Polynomialzeit reduzieren. Ein Problem, das sowohl **NP**-schwer ist, als auch in **NP** liegt, wird als **NP**-vollständig bezeichnet [28].

Lösungen zu **NP**-Problemen lassen sich in **P** Zeit überprüfen, dazu kann man beispielsweise einen Algorithmus einsetzen, der nur ein Zertifikat überprüft [28]. Es ist also möglich in effizienter Zeit zu überprüfen, ob eine mögliche Lösung eine tatsächliche Lösung des Problems ist. Daher wird die Komplexitätsklasse **NP** auch als *die Klasse der effizient überprüfbaren Probleme* bezeichnet. Es wird daher angenommen, dass **P** $\subset$ **NP** gilt, auch wenn es noch nicht eindeutig bewiesen ist. Klar hingegen ist das **P** $\subseteq$ **NP** gilt.

Nichtdeterminismus lässt sich von klassischen Computern nicht simulieren, da bei jeder Projektion auf eine Potenzmenge das System komplett kopiert werden müsste.

Definition 5. Die Klasse **PSPACE** enthält alle Probleme, welche sich mit deterministisch Algorithmen berechnen lassen, die mit polynomial viel Platz in der Eingabelänge auskommen [28].

Auch wenn für uns der Platzbedarf der Algorithmen unerheblich ist, so können wir die Komplexitätsklassen mit Hilfe der Klasse **PSPACE** beschränken [36]:

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}.$$

Da aber nicht alle Algorithmen immer ein gültiges Ergebnis liefern, betrachten wir nun randomisierte Algorithmen und in welche Komplexitätsklassen diese einzuordnen sind. Ein randomisierter Algorithmus ist ein Algorithmus, der eine Reihe von Zufallsbits $(r_0 \ldots, r_{k-1})$ zur Hilfe nimmt.

Definition 6. Ein Entscheidungsproblem E liegt in der Klasse **RP**, wenn es einen randomisierten Algorithmus A gibt, sodass Folgendes gilt:

- falls $E(x) = 0$, so ist $A(x) = 0$,
- falls $E(x) = 1$, so ist $A(x) = 1$ mit Wahrscheinlichkeit größer gleich $\frac{1}{2}$, und
- die Laufzeit von A ist polynomial.

RP kürzt *randomized polynomial time* ab [22].

Ein randomisierter Algorithmus muss, um in die Klasse **RP** eingeordnet zu werden, für jede Eingabe das korrekte Ergebnis mit höherer Wahrscheinlichkeit als das Falsche liefern. Deshalb ist die Fehlerwahrscheinlichkeit höchstens $\frac{1}{2}$. Durch mehrfaches Ausführen erhalten wir also mit der Wahrscheinlichkeit $\frac{1}{2}^k$ ein richtiges Ergebnis. Da die Laufzeit des Algorithmus polynomial ist, wird die k-fache Ausführung auch nur k mal diese polynomiale Zeit benötigen und ist daher nach unserer Definition weiterhin effizient. Sollte innerhalb dieser k Ausführungen der Algorithmus auch nur ein einziges Mal eine 0 als Ergebnis zurückgeben, so ist das gesamte Ergebnis 0, ansonsten 1.

Betrachte man nun einen Algorithmus A mit der Fehlerwahrscheinlichkeit $\frac{3}{4}$. Diese ist eindeutig zu groß, um den Algorithmus in die Klasse **RP** einzuordnen. Wenn wir A nun aber statt einmal einfach pro Anwendung drei mal ausführen, so sinkt die Fehlerwahrscheinlichkeit auf $(\frac{3}{4})^3 \approx 0.42$. Damit wäre der Algorithmus A^3 durchaus in der Klasse **RP** einzuordnen. Dies geht mit jedem Algorithmus, dessen Fehler durch eine Konstante beschränkt ist. Damit zwischen Algorithmen jedoch unterschieden werden kann, definieren wir die Klasse **BPP**.

Definition 7. Ein Entscheidungsproblem E liegt in der Klasse **BPP**, wenn es einen randomisierten Algorithmus A gibt, sodass Folgendes gilt:

- einer konstanten Fehlerwahrscheinlichkeit c mit $\frac{1}{2} < c < 1$,
- falls $E(x) = 0$, so ist $A(x) = 0$ mit Wahrscheinlichkeit größer gleich c,
- falls $E(x) = 1$, so ist $A(x) = 1$ mit Wahrscheinlichkeit größer gleich c, und
- die Laufzeit von A ist polynomial.

BPP kürzt *bounded error probabilistic polynomial time* ab [19].

Aus den Definition 2, 6, und 7 lässt sich nun schließen:

$$\textbf{P} \subseteq \textbf{RP} \subseteq \textbf{BPP}\ [22].$$

In verschiedenen anderen Veröffentlichungen wurde bereits ausgiebig dargestellt, wie **BPP** in die anderen Komplexitätsklassen einzuordnen ist, es gilt nämlich:

$$\textbf{P} \subseteq \textbf{BPP} \subseteq \textbf{PSPACE}\ [19].$$

Um die klassischen Algorithmen nun in den Kontext der Quantenalgorithmen zu setzten, definieren wir die Klasse **BQP**. Hierbei ist für uns insbesondere die Komplexitätsklasse **BPP** von Bedeutung, da Algorithmen, die quantenmechanische Effekte nutzen, einer gewissen Fehlerwahrscheinlichkeit unterliegen [22].

Definition 8. Ein Entscheidungsproblem E liegt in der Klasse **BQP**, wenn es einen Quantenalgorithmus A gibt, sodass Folgendes gilt:

- einer konstanten Fehlerwahrscheinlichkeit c mit $\frac{1}{2} < c < 1$,
- falls $E(x) = 0$, so ist $A(x) = 0$ mit Wahrscheinlichkeit größer gleich c,
- falls $E(x) = 1$, so ist $A(x) = 1$ mit Wahrscheinlichkeit größer gleich c, und
- A ist durch einen Quantenschaltkreis polynomialer Größe berechenbar.

BQP steht für *bounded error quantum polynomial time* [22].

Offensichtlich unterscheidet sich die Definition der Komplexitätsklasse **BQP** nur minimal von der Definition der Klasse **BPP**, die das klassische Pendant dazu bildet.

Der Platzbedarf von Quantenalgorithmen ist für uns im Allgemeinen nicht so wichtig, wir wissen jedoch, dass der Platzbedarf von Quantenalgorithmen polynomiell wächst und somit die Größe von Quantenschaltkreisen beschränkt wird durch [22]:

$$\textbf{BQP} \subseteq \textbf{PSPACE}$$

Durch die Ähnlichkeit zwischen den Definitionen 7 und 8 und dem Umwandeln klassischer Algorithmen in Quantenalgorithmen (vgl. Abschnitt 3.6) lässt sich somit auch feststellen das gilt:

$$\textbf{BPP} \subseteq \textbf{BQP} \subseteq \textbf{PSPACE}\ [5]$$

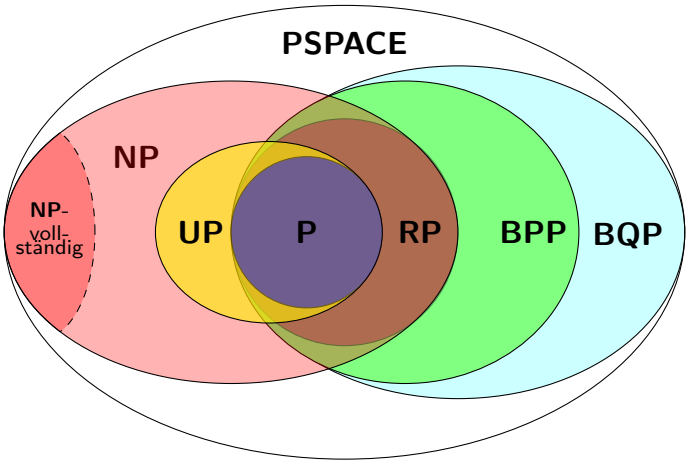


Abbildung 1: Alle eingeführten Komplexitätsklassen in der Übersicht

Um nun die Zusammenhänge zwischen all diesen Klassen ein wenig greifbarer darzustellen, ziehen wir Abbildung 1 hinzu. Dort wird eindeutig klar welche Abdeckung die Klasse **BQP** im Verhältnis zu den andern Klassen hat.

Nun haben wir ausgiebig die Komplexitätsklassen betrachtet, um Quantenalgorithmen in die bereits bekannte klassische Sicht einzuordnen. Um Hash-Verfahren aus der theoretischen Sicht genauer verstehen zu können, führen wir nun noch eindeutige Funktionen im Zusammenhang mit Turingmaschinen ein.

Definition 9. Sei eine UTM (unabmiguous Turing machine) eine NTM, die nur eine akzeptierende Belegung für jede mögliche Eingabe hat. **UP** ist die Komplexitätsklasse von Problemen, die von einer UTM gelöst werden können.
UP steht für *unambiguous non-deterministic polynomial time* [20].

Eine UTM, wörtlich übersetzt eine eindeutige Turingmaschine, ist eng mit dem Prinzip von Einwegfunktionen verwoben. Einwegfunktionen (vgl. Abschnitt 5.2), die wirklich Eins-zu-Eins abbilden sind nur möglich, wenn $\mathbf{P} \neq \mathbf{UP}$ gilt [20]. Die meisten Hashfunktionen sind allerdings nicht Eins-zu-Eins, das widerspricht schon der Idee der Berechnung von Hashwerten und der Verkleinerung des Speicherraumes. Es wird nur auf Kollisionsresistenz geprüft, Kollisionen sind aber nicht komplett auszuschließen. Dieses wird noch im Abschnitt 5.2 aufgegriffen und genauer betrachtet. Um diese Komplexitätsklasse einzusortieren, sei diese Gleichung gegeben:

$$\mathbf{P} \subseteq \mathbf{UP} \subseteq \mathbf{NP} \text{ [23]}.$$

Noch zu berücksichtigen ist, dass keine dieser hier eben vorgestellten Annahmen eindeutig bewiesen ist. Es ist immer noch möglich, dass **P** = **NP** gilt, selbst **P** = **PSPACE** bleibt weiterhin offen, da eine Teilmengenrelation bis heute nicht eindeutig bewiesen ist. Auch wenn die meisten Wissenschaftler sich einig sind, dass beides nicht zutrifft [6].

2.2. Aktuelle Methoden der Kryptographie

Symmetrische Verfahren haben einige Schwächen, je größer die Gruppe an Personen wird, die miteinander kommunizieren möchten. Dann müssen $\frac{n \cdot (n-1)}{2}$ Schlüssel ausgetauscht werden. Da es aber mittlerweile mehrere Milliarden Geräte mit Zugang zum Internet gibt, ist das keine sonderlich sinnvolle Option. Es ist auch möglich stattdessen eine zentrale Stelle einzusetzen, der dann wiederum alle Teilnehmer vertrauen müssen. Dieses Verfahren wird bereits im Rahmen der Mobiltelefonie eingesetzt [8].

2.2.1. Public und Private Key

Die Idee zu Public-Private-Key-Verfahren wurde 1976 erstmals von Diffie und Hellman vorgestellt [12]. Sie schlagen vor, dass jeder nur seinen eigenen geheimen Schlüssel d (*d*ecrypt) besitzen soll und diesen sicher verwahren muss. Die dazu korrespondierenden öffentlichen Schlüssel e (*e*ncrypt) müssen an die Personen verteilt werden, von denen man verschlüsselte Nachrichten erhalten möchte. Dies bedeutet im Normalfall, dass der Schlüssel einfach öffentlich publiziert wird. Diese beiden Schlüssel bilden das Schlüsselpaar (e, d). Es muss allerdings gewährleistet sein, dass der Schlüssel e auch authentisch ist und von der Person stammt, die die Nachricht erhalten soll. So müssen deutlich weniger Schlüssel ausgetauscht werden als es noch mit symmetrischen Schlüssel notwendig war, da jeder den selben öffentlichen Schlüssel benutzt, um einer bestimmten Person eine Nachricht zu schicken.

Das bekannteste aktuell eingesetzte Verfahren ist das RSA-Verfahren [34].

2.2.2. Digitale Signaturen

Digitale Signaturen drehen das Konzept von privaten und öffentlichen Schlüsseln um, sodass der Schlüssel zum Verschlüsseln (e) nun privat gehalten wird, der Schlüssel zum Entschlüsseln (d) hingegen wird veröffentlicht. Auf diese Art und Weise kann der Erzeuger einer Nachricht diese mit seinem geheimen Schlüssel verschlüsseln und jeder kann mit Hilfe des öffentlichen Schlüssels die Nachricht wieder entschlüsseln. Somit kann jeder

sich sicher sein, dass die Nachricht auch wirklich vom Erzeuger des privaten Schlüssels kommt, solange die Authentizität des öffentlichen Schlüssel sicher gestellt ist. Da dies nicht immer sonderlich effizient ist, wird meist nur ein Hash der Nachricht berechnet, verschlüsselt und dieser angehängt [8]. Formal definiert und näher ausgeführt wird dieses Prinzip noch in Abschnitt 5.4.

3. Quantencomputer

Dieses Kapitel basiert auf dem Buch *Quantum Computing verstehen* von Matthias Homeister [22]. Dieser Einfluss macht sich insbesondere durch die Schreibweisen einiger mathematischer Formeln bemerkbar. Ein mathematisches Grundlagenkapitel, welches das Verständnis vereinfachen sollte, befindet sich im Anhang.

Ein Quantencomputer soll all das können was ein klassischer Computer auch kann. Allerdings nutzt ein Quantencomputer die Effekte der Quantenmechanik, dadurch werden einige sonst sehr simple Operationen deutlich komplizierter [33]. Diese Bachelorarbeit setzt sich jedoch nicht mit den physikalischen Begebenheiten eines Quantencomputers auseinander, sondern erklärt die Auswirkungen auf die Berechnungen und was sich durch das Nutzen der Quantenmechanik an den Algorithmen ändert.

Der entscheidende Unterschied besteht darin, dass ein Quantencomputer nicht mit gewöhnlichen Bits arbeitet, sondern mit Quantenbits, diese werden auch Qubits genannt. Es gibt noch keinen Quantencomputer, der mit ausreichend vielen Quantenbits arbeitet, um außerhalb von Laborbedingungen relevante Berechnungen durchführen zu können [33]. Dennoch gibt es bereits Algorithmen, die auf diesen kleiner skalierten Quantencomputern nachweislich arbeiten können und sich dabei das besondere Verhalten von Quantenbits zu Nutzen machen. Die bekanntesten Algorithmen sind von Shor und Grover, diese werden in den Abschnitten 4.1 und 4.2 ausführlich betrachtet.

Ein Quantencomputer ist nicht mit einem Parallelrechner zu vergleichen. Auch wenn Parallelrechner durchaus eine Beschleunigung herbeiführen können, da sie mehr Rechenleistung zur Verfügung haben als gewöhnliche Rechner. So kann ein Parallelrechner viele Berechnungen gleichzeitig durchführen, allerdings ist jede Berechnung einzeln betrachtet in einem klassischen und damit konkreten Zustand. Bei einem Quantencomputer ist dies nicht der Fall. Der Quantencomputer ist in allen Zuständen der Berechnung zur gleichen Zeit. Dies ist eng mit dem Verhalten einer nichtdeterministischen Turingmaschine verwoben (vgl. Abschnitt B und Definition 47). Ein Quantencomputer rechnet auf seinen Quantenbits mit allen möglichen Zuständen gleichzeitig und ist so parallel in allen möglichen Endergebnissen. Das hieraus resultierende Problem ist die Ergebnisse richtig zu interpretieren, beziehungsweise das richtige Ergebnis aus den Quantenbits zu erhalten.

3.1. Funktionsweise

Ein Quantencomputer ist ein kompliziertes Konstrukt, welches einigen besonderen Regeln unterliegt. Die Einschränkungen, die für die Berechnungen relevant sind, werden wir nun betrachten. Es gibt sowohl einige Regeln, die die Funktionsweise eines Quantencomputers einschränken, als auch Regeln, die dafür sorgen, dass ein Quantencomputer besondere Arten von Rechnungen durchführen kann, die sonst nur sehr schwer durchzuführen oder zu simulieren sind [9, 22, 33]. Die zentrale Einschränkung geschieht durch das Nutzen der Quantenbits, welche in eine Superposition übergehen (vgl. Abschnitt 3.2). Dieser Zustand kann nicht genau gemessen werden, dadurch müssen einige Anpassung getroffen werden (vgl. Abschnitt 3.3). Daraus folgt somit auch, dass sich Quantenbits in einer Superposition nicht vernünftig kopieren lassen, zumindest nicht im klassischen Verständnis, dazu jedoch mehr in Abschnitt 3.5. Betrachten wir ein Bauteil einmal aus der Hardware Sichtweise. Ein klassisches boolesches AND (z. B. $0 \wedge 1 = 0$) hat zwei Kabel als Eingänge aber nur eines als Ausgang, bei der Verrechnung dieser wird ein Bit *vernichtet*. Wenn aus mehreren Bits ein Bit wird, spricht man immer von der Vernichtung der Bits, hierbei entsteht Wärme [22]. Zusätzlich lässt die Ausgabe eines AND Gatters keine eindeutigen Rückschlüsse auf die Eingabe zu. Bei Ausgabe 0 kann die Eingabe sowohl 00, 01 als auch 10 gewesen sein. Zusätzlich kann ein Quantencomputer in mehreren Zuständen gleichzeitig sein, daher muss jede Transformation, die ein Quantencomputer ausführt, umkehrbar sein. Da Quantencomputer aus verschiedenen Gründen nach Möglichkeit sehr nahe am absoluten Nullpunkt gehalten werden, sollte Wärmeentwicklung möglichst verhindert werden [33]. Dies ist insbesondere wichtig, um den äußeren Einfluss auf die Quantenbits zu reduzieren, sodass die Superposition nicht beeinflusst wird.

Die Lösung dieses Problem erfordert unitäre Transformationen, um die Zustände von Quantenbits zu verändern. Unitäre Transformationen werden in den folgenden Abschnitten näher betrachtet und in Definition 41 (vgl. Anhang) formal definiert.

3.2. Superposition

Quantenbits können in eine Superposition übergehen, sodass sie nicht länger im fest vorgeschriebenen Zustand $|0\rangle$ oder $|1\rangle$ sind. Das Quantenbit befindet sich zwar nicht länger im Zustand $|0\rangle \text{ oder } |1\rangle$, es ist jedoch auch nicht in einem Zustand dazwischen.

Definition 10. Ein Quantenbit in einer Superposition hat die Amplituden α und β , $\alpha, \beta \in \mathbb{C}$ und wird wie folgt dargestellt:

$$\alpha \cdot |0\rangle \pm \beta \cdot |1\rangle.$$

Einem Quantenbit werden zwei Amplituden zugewiesen, wobei in dieser Darstellung α immer den Anteil für 0 und β für 1 bestimmt. Diese Amplituden bestimmen die Wahrscheinlichkeiten, welches Ergebnis eine Messung liefern kann(vgl. Abschnitt 3.3). Daher ergibt sich folgende Einschränkung.

$$|\alpha|^2 + |\beta|^2 = 1 \tag{3.1}$$

An dieser Gleichung ist gut zu erkennen, dass sich ein Quantenbit somit in unendlich vielen verschiedenen Teilzuständen von $|0\rangle$ und $|1\rangle$ befinden kann. Intuitiv gesprochen befindet es sich sowohl im Zustand $|0\rangle$ als auch im Zustand $|1\rangle$. Eine andere Sichtweise betrachtet die Amplituden α und β als eine Spin eines Quantenbits. Hierbei wird das Quantenbit wie ein Elektron betrachtet, das sich sowohl in die α als auch in β dreht. Physiker beschäftigen sich schon länger mit der Bedeutung des Spin und wie das Verhalten des Elektrons davon beeinflusst wird.

Die Superposition ermöglicht, dass einige Probleme sehr effizient gelöst werden können. So kann ein Register, welches aus n Quantenbits besteht, alle 2^n Zustände gleichzeitig annehmen [22].

Definition 11. Der Zustand eines Quantenbits wird durch einen Vektor der Länge 1 in einem zweidimensionalen komplexen Vektorraum beschrieben. Die Superposition $\alpha|0\rangle + \beta|1\rangle$ wird demnach beschrieben als Vektor

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

Um die Zuordnung von α und β zu verdeutlichen, schreiben wir den Vektor als Linearkombination der zweidimensionalen Standartbasis.

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

Die nicht überlagerten Zustände $|0\rangle, |1\rangle$ bilden die Basis (vgl. Definition 31) dieses Vektoraumes. Die Superposition lässt sich dadurch als Linearkombination der Basiselemente darstellen. Aufgrund der Gleichung 3.1 sind aber nicht alle Vektoren dieses zwei-

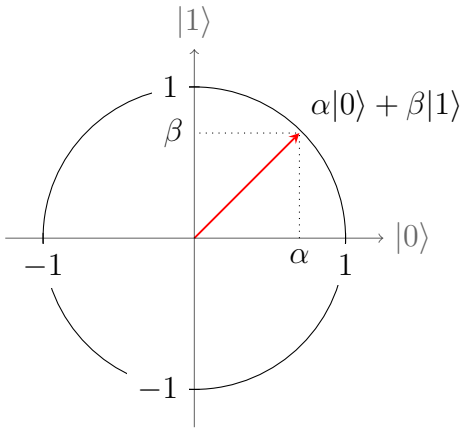


Abbildung 2: Die Superposition als Vektor [22]

dimensionalen Vektorraumes mögliche Zustände, sondern nur die der Länge 1. Es bildet sich also wie in Abbildung 2 gut zu sehen ist ein Einheitskreis auf dem alle möglichen Zustände liegen.

Da $\alpha, \beta \in \mathbb{C}$ gilt, handelt es sich bei diesem Vektorraum um einen komplexen Vektorraum (vgl. Definition 29). Dies ist in Abbildung 2 nicht ersichtlich, daher stellen wir uns den Vektorraum so vor als wäre er über den reellen Zahlen ($\mathbb{R}$) definiert. Beim Rechnen wenden wir jedoch die Regeln für komplexe Zahlen ($\mathbb{C}$) an.

Ein Quantenbit kann aus dem klassischen Zustand in eine Superposition versetzt werden, sodass es mit der gleichen Wahrscheinlichkeit im Zustand 0 wie auch im Zustand 1 ist, also $\alpha = \beta$. Dazu nutzen wir die Hadamard-Matrix [22, 26].

Definition 12. Die Hadamard-Matrix H ist definiert als:

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

Wenden wir diese Matrix auf ein Quantenbit an, welches sich im Zustand $\alpha|0\rangle + \beta|1\rangle$ befindet, so lässt sich der Folgezustand $\alpha'|0\rangle + \beta'|1\rangle$ leicht berechnen, wenn man die Matrix als Transformation anwendet (vgl. Definition 39). Das Anwenden der Matrix als Transformation schreiben wir als $\xrightarrow{H}$. Die Berechnung des Folgezustands ergibt sich dann wie folgt:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} \xrightarrow{H} \begin{pmatrix} \alpha' \\ \beta' \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix} \cdot \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

Die Hadamard-Transformation, die auf einem Quantenbit angewendet wird, welches

sich eindeutig im Zustand $|0\rangle$ oder $|1\rangle$ befindet, wird nach der Anwendung in folgendem Zustand sein:

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

Bei erneuter Anwendung der Hadamard-Transformation wechseln die Quantenbits wieder in den Normalzustand $|0\rangle$ und $|1\rangle$. So ist gezeigt, dass H zu sich selbst invers ist (vgl. Definition 37):

$$H^{-1} = H \iff HH = I.$$

Die Hadamard-Matrix beschreibt also eine umkehrbare Transformation, jetzt stellt sich nur noch die Frage, ob hierbei Bits vernichtet werden (vgl. Abschnitt 3.1). Die adjungierte Hadamard-Matrix (vgl. Definition 40) ist zu H invers, es gilt also:

$$H^{\dagger} = H^{-1}.$$

Die Hadamard-Matrix ist somit nach Definition 41 eine unitäre Matrix. Dies ist ein entscheidender Aspekt, da hierdurch die Voraussetzungen aus Abschnitt 3.1 erfüllt werden. Bits können durch das Anwenden der Hadamard-Transformation sowohl in eine Superposition hinein versetzt werden, als auch wieder in einen eindeutigen Zustand zurück. Durch das einmalige Anwenden der Hadamard-Transformation und das Messen (vgl. Abschnitt 3.3) kann ein Zufallsgenerator erstellt werden [27]. Echte Zufallszahlen spielen beim Verschlüsseln von Daten insbesondere beim One-Time-Pad eine große Rolle (vgl. Abschnitt 6).

3.3. Das Messen von Quantenbits

Quantenbits, die in einer Superposition $\alpha \cdot |0\rangle + \beta \cdot |1\rangle$ gemessen werden, nehmen mit der Wahrscheinlichkeit α^2 den Zustand $|0\rangle$ und mit der Wahrscheinlichkeit β^2 den Zustand $|1\rangle$ an (Gleichung 3.1). Durch das Messen verlässt ein Quantenbit also seine Superposition und kehrt zu einem Normalzustand zurück, in dem es einen eindeutigen Zustand hat. Vor dem Messen ist unbekannt wie groß die Amplituden α und β sind und durch das Messergebnis von $|1\rangle$ oder $|0\rangle$ können auch nur minimale Rückschlüsse darauf gezogen werden. Dieses Phänomen wird durch das Gedankenexperiment *Schrödingers Katze*

auf die wesentlichen Punkte herunter gebrochen dargestellt [22, 38], welches wir im Folgenden erläutern werden.

Beispiel 13. Eine lebendige Katze wird in eine Kiste gesteckt, hinzu kommt ein radioaktives Element, ein Geigerzähler und ein verschlossenes Gefäß mit Gift. Die Kiste wird nun versiegelt und das Experiment beginnt. Sollte das radioaktive Element zerfallen, löst der Geigerzähler aus und dadurch (mit Hilfe eines dafür vorgesehen Auslösemechanismus) wird das Gift freigesetzt. Solange die Kiste verschlossen ist, kann die Katze sowohl als *lebendig* als auch *tot* angesehen werden. Sie befindet sich mit einer gewissen Wahrscheinlichkeit im Zustand *tot* also 0, mit der Gegenwahrscheinlichkeit im Zustand *lebendig* also 1.

Bis zu dem Zeitpunkt an dem wir die Box öffnen und nachsehen, ob die Katze noch lebt, ist die Katze also in beiden Zuständen gleichzeitig. Das Ergebnis, dass wir nach einer Messung eines Quantenbits erhalten, hängt also von einer Wahrscheinlichkeit ab. Diese kann jedoch auch so verteilt sein, dass es sich bereits sicher in dem Zustand $0 \cdot |0\rangle + 1 \cdot |1\rangle$, also eigentlich im Zustand $|1\rangle$ befindet. So können wir nach der Messung nur ein Ergebnis erhalten. Auf diese Art und Weise lassen sich Berechnungen geschickt mit der Hadamard-Transformation direkt vor der Messung manipulieren.

Beispiel 14. Befindet sich ein Quantenbit $|x\rangle$ im Zustand $\frac{1}{\sqrt{2}} \cdot |0\rangle - \frac{1}{\sqrt{2}} \cdot |1\rangle$ so wird das Quantenbit $|x\rangle$ nach Anwendung der Hadamard-Transformation im Zustand

$$|x\rangle \xrightarrow{H} |x'\rangle = 0 \cdot |0\rangle - 1 \cdot |1\rangle$$

sein. Also eindeutig im Zustand $|1\rangle$. Somit kann eine Messung auch nur zu einem Ergebnis führen, einer 1.

Wenn die Messung eine 0 als Ergebnis ausgibt, wissen wir, dass das Quantenbit nicht in dem Zustand war, der in Beispiel 14 angenommen wurde. Sollte die Berechnung mit Werten ausgeführt worden sein, die wir voraussagen können, so wird durch eine geschickte Hadamard-Transformation das Quantenbit zwar weiterhin in der Superposition bleiben, jedoch eindeutig den Zustand $|1\rangle$ annehmen, es gilt daher $\alpha = 0$ und $\beta = 1$. Nach Möglichkeit wird mit Hilfe der Anwendung der Hadamard-Transformation also eine der beiden Amplituden verstärkt, die andere hingegen ausgelöscht, sowie in Beispiel 14 dargestellt. Diese Technik nennt sich auch konstruktive und destruktive Interferenz. Es ist allerdings oft nicht möglich eine Amplitude komplett auszulöschen, sodass es daher unmöglich ist einen Wert mit einer Wahrscheinlichkeit von 100% zu lesen. Durch geschickte Berechnungen und Interferenz lässt sich dann lediglich die Wahrscheinlichkeit

erhöhen das gewünschte Ergebnis zu lesen. Folglich müssen Berechnungen dann häufig mehrfach ausgeführt werden, um verlässliche Ergebnisse zu erhalten. Somit haben alle Quantenalgorithmen eine Fehlerwahrscheinlichkeit, dies erklärt die enge Verknüpfung der Komplexitätsklassen **BPP** und **BQP** (vgl. Definitionen 7 und 8).

Außer der Hadamard-Transformation gibt es noch andere Möglichkeiten geschickt mit den Quantenbits umzugehen. Die Quanten-Fourier-Transformation wird beispielsweise auch genutzt, um das Lesen von bestimmten Werten zu ermöglichen. Dieses Verfahren wird in Shors Algorithmus eingesetzt und in Abschnitt 4.1 noch etwas genauer betrachtet. Ebenso ist es möglich die Basis zu verändern, anhand welcher die Bits gemessen werden, um das Ergebnis zu unserem Vorteil zu manipulieren. Quantenalgorithmen nutzen meist die besonderen Effekte der Quantenbits und versuchen dann durch Manipulation vorm Messen, das Messen des gewünschten Ergebnisses zu begünstigen. Das werden wir noch häufiger sehen, da immer sehr sorgfältig mit den Quantenbits umgegangen werden muss, um das gewünschte Ergebnis zu erhalten [21].

3.4. Quantenregister

Ein klassisches Bit kann in zwei Zuständen sein: 0 und 1. Ein Register mit 3 Bits kann folglich in 8 verschiedenen Zuständen sein: $000, 001, \ldots, 111$. Durch ein Register mit n Bits können folglich die Zahlen $0 \text{ bis } 2^{n-1}$ binär dargestellt werden. Quantenregister können durch die Amplituden der einzelnen Quantenbits in eine Superposition übergehen, indem diese als zusammenhängende Einheit betrachtet werden. Hierbei werden die Amplituden aller Quantenbits miteinander verrechnet, um den Gesamtzustand des Registers zu beschreiben.

Beispiel 15. Man betrachte ein Quantenregister mit zwei Quantenbits $|x_1\rangle$ und $|x_2\rangle$. Das erste Quantenbit sei im Zustand $|x_0\rangle = \gamma_0|0\rangle + \gamma_1|1\rangle$, das Zweite im Zustand $|x_1\rangle = \beta_0|0\rangle + \beta_1|1\rangle$, somit ist das gesamte Register im Zustand:

$$\begin{aligned} R &= |x_1\rangle|x_0\rangle \\ &= (\beta_0|0\rangle + \beta_1|1\rangle) \cdot (\gamma_0|0\rangle + \gamma_1|1\rangle) \\ &= \beta_0\gamma_0|0\rangle|0\rangle + \beta_0\gamma_1|0\rangle|1\rangle + \beta_1\gamma_0|1\rangle|0\rangle + \beta_1\gamma_1|1\rangle|1\rangle. \end{aligned}$$

Um das Ganze zu vereinfachen werden die einzelnen Amplituden β_i und γ_j jedoch zu α_{ij} zusammengefasst und vereinheitlicht, sodass der Sinn sich direkt herauskristallisiert. Jeder mögliche Zustand des Registers ist eine Zusammensetzung jeder möglichen

Zustände der jeweiligen Quantenbits.

$$R = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

Man sollte hierbei jedoch noch im Hinterkopf behalten, dass es sich bei den Amplituden um komplexe Zahlen handelt und die Zustände der Quantenbits als Vektor der Länge 1 dargestellt werden (vgl. Abschnitt 3.2). Also hat ein Register mit *n*-Quantenbits 2^n Komponenten und folglich werden unitäre Transformationen durch $2^n \times 2^n$-Matrizen dargestellt [22]. Für das Register R gilt somit:

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Es ist allerdings nicht nur wichtig in welchem Zustand das gesamte Register sich befindet, sondern auch in welchem Zustand jedes einzelne Quantenbit ist, da es Wechselwirkungen zwischen Quantenbits geben kann. Um den Wert des gesamten Registers in der Quanten-Superposition darzustellen, muss das Tensorprodukt gebildet werden (vgl. Defintion 43). Mit Hilfe des Tensorproduktes kann der Vektorraum für das Register durch die Vektoren der einzelnen Bits zusammengesetzt werden. Das Register wird dann durch einen Vektor eines 2^n-dimensionalen Vektorraums dargestellt.

Soll nun die Hadamard-Transformation auf ein Register $|x_1 \dots x_n\rangle$ angewendet werden, so wird die Transformation $H_1 \otimes \cdots \otimes H_n$ als Tensorprodukt verrechnet. In einem zwei Bit Register $|xy\rangle$ wird also $H \otimes H = H_2$ angewendet, um $|x\rangle$ und $|y\rangle$ zu transformieren. Möchte man nur $|x\rangle$ transformieren, so muss $H \otimes I_2$ angewendet werden. Analog dazu wird bei $|y\rangle$ das Ganze andersherum $I_2 \otimes H$ angewendet. Die Reihenfolge, in der die Operationen auf die Bits angewendet werden, spielt dabei keine Rolle [22]. Neben der Hadamard-Transformation gibt es auch noch andere Operationen, die auf Registern ausgeführt werden können.

Definition 16. Die CNOT (controlled not) Operation ist das unitäre Pendant zum klassischen Exklusiven-Oder (XOR).

$$|x, y\rangle \xrightarrow{CNOT} |x, x \oplus y\rangle\rangle$$

Beispiel 17. Wendet man die CNOT-Transformation an, so folgt daraus, dass die Amplituden von $|10\rangle$ und $|11\rangle$ vertauscht werden.

$$\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} \xrightarrow{CNOT} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_2 \end{pmatrix}$$

Die CNOT-Transformation wird insbesondere dann angewendet, wenn einzelne Bits eines Quantenregisters ihre Amplitude tauschen sollen. Diese Transformation wird beim Verschränken von Quantenbits (vgl. Abschnitt 3.5) angewendet. Verschränkung ist einer der besonderen quantenmechanischen Effekte, welcher Berechnungen von Quantencomputern so besonders effizient machen. Es gibt selbstverständlich noch andere mögliche Operationen, diese sind für uns aber im Rahmen dieser Bachelorarbeit nur von geringem Interesse und werden daher nicht betrachtet.

Messen als Projektion

Um ein Quantenregister zu messen gibt es zwei Vorgehensweisen. Zum Einen die bereits erwähnte Variante, das gesamte Register anhand der Standardbasis zu messen und somit in einen klassischen Zustand zurückzuführen. Oder man wählt die zweite Variante *Messen als Projektion*, da manchmal nicht der Zustand des gesamten Registers, sondern nur der eines einzelnen Quantenbits von Bedeutung ist. Es ist tatsächlich möglich ein bestimmtes Bit mit Hilfe einer neuen Basis zu messen und dadurch die Superposition zu erhalten. Hierbei wird das mathematische Konzept von Projektionen in einen Unterraum genutzt (vgl. Definition 35). Anschaulich lässt sich das so verstehen, dass das Koordinatensystem aus Abbildung 2 gedreht wird. Zusätzlich wird ein Observable eingeführt, welches durch die Projektion in einem Unterraum aus den aufgespannten Vektorräumen des Registers zusammengesetzt wird. Ohne hier jetzt in die Tiefe zu gehen, sehen wir daher nun, dass die Superposition mit Hilfe der neuen Basis $\Big(\text{z. B.} \frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big), \frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big)\Big)$ gemessen wird und die Superposition zerstört, beziehungsweise verändert wird. Wir müssen eigentlich davon ausgehen, dass nach dem Messen die Superposition zerstört worden ist. Da wir aber eine geschickt gewählte Orthogonalbasis verwendet haben, ist das Ergebnis der *zerstörten* Superposition eine neue darauf folgende Superposition. So hat eine Messung dieser Art zwei Ergebnisse: einen Quantenfolgezustand, der genauso wieder zerstört werden kann wie der davor und die klassische Information über den Wert des Bits. Beim Messen wird nur der Wert des gemessenen Bits offenbart, nicht jedoch die Amplituden des Folgezustandes. Um überhaupt eine Superposition messen zu können,

muss eine passende orthogonale Basis gefunden werden. Da alle Zustandsvektoren die Länge 1 haben, handelt es sich sogar um Orthonormalbasen.

Durch die Superposition und das Rechnen auf Registern kann somit eine Rechnung simultan auf alle möglichen Registerzustände angewendet werden. Dies gibt Quantencomputern eine enorme Rechenleistung, auch Quantenparallelismus genannt. Nach den Berechnungen muss dann durch geschicktes Messen oder durch Transformationen vor dem Messen das gewünschte Ergebnis aus dem Register gelesen werden. Da diese Transformationen aber selten Amplituden komplett vernichten können und daher nicht eindeutige Ergebnisse liefern, unterliegen alle Quantenalgorithmen gewissen Fehlerwahrscheinlichkeiten. Dies ist der Grund dafür, dass Quantenalgorithmen in der Laufzeitklasse **BQP** liegen.

3.5. Verschränkungen

Wenn wir das Beispiel 13 von *Schrödingers Katze* aus Abschnitt 3.3 noch einmal ein wenig genauer ansehen, so lässt sich sagen: die Katze in der Kiste wurde mit dem Zerfall des radioaktiven Elementes verschränkt. Ihr Zustand ist der gleiche wie der des Elementes. Neben Katzen ist es auch möglich Bits miteinander zu verschränken. Wendet man die Hadamard-Transformation auf einem zwei Bit Quantenregister $(a_1 a_2)$ an und danach die CNOT-Transformation, so erhält man eine Verschränkung von a_1 mit a_2:

$$\begin{aligned} |00\rangle &\xrightarrow{H\otimes I_2} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle) \\ &\xrightarrow{CNOT} \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \end{aligned}$$

Das Bit a_1 einzeln betrachtet ist in dem Zustand $\frac{1}{\sqrt{2}}|0\rangle + |1\rangle$, so können beim Messen jeweils beide Ergebnisse mit den gleichen Wahrscheinlichkeiten auftreten. Ist das Bit nach der Messung im Zustand $|0\rangle$, so ist das Register im Zustand $|00\rangle$. Sollte es jedoch im Zustand $|1\rangle$ sein, so ist das Register im Zustand $|11\rangle$. Wir sehen also, dass durch die Messung des Bits a_1 das Ergebnis der Messung des zweiten Bits a_2 bereits feststeht und umgekehrt. Hierbei sei erwähnt, dass diese Verschränkung nicht durch räumliche Distanz eingeschränkt wird. So können zwei bereits verschränkte Quantenbits räumlich getrennt werden und zwei verschiedenen Personen übergeben werden. Wenn diese Personen nun unabhängig von einander ihr Quantenbit messen, so werden sie beim Vergleichen feststellen, dass beide das gleiche Ergebnis gemessen haben. Komplett unabhängig davon ist, welches Bit zuerst gemessen wurde, was bedeutet, dass Informationen

über eine beliebige räumliche Distanz geteilt wurden, sich also schneller bewegt haben als das Licht. Schon von Einstein als *Spukhafte Fernwirkung* benannt, ist dieser Effekt auch unter dem Namen *Einstein-Podolsky-Rosen-Paradoxon* bekannt [15, 22]. Dieses Phänomen wird bereits von einigen Wissenschaftlern zu deren Vorteil eingesetzt [4]. Die Verschränkung von Bits ist eines der wesentlichen Prinzipien der Quantenmechanik und spielt für viele Verfahren, insbesondere in der Quantenkryptographie, eine wichtige Rolle.

3.6. Quantenschaltkreis

Ein Quantencomputer besteht im wesentlichen auch nur aus Schaltkreisen. Diese müssen nur in der Lage sein, den Anforderungen der Quantenbits gerecht zu werden. In diesem Falle also ein Quantenschaltkreis. Da Quantencomputer auch klassische Berechnungen durchführen sollen, müssen klassische Schaltkreiselemente in unitäre Gatter überführt werden. Unitäre Gatter benötigen meist mehr Bauteile, um die gleichen Operationen auszuführen, die ein klassisches Bauteil simpel ausführen kann. Man betrachte die folgenden Definitionen zu booleschen Schaltkreisen und die daraus aufbauende Definition zu Toffoli-Gattern.

Definition 18. Sei B eine Basis über eine endliche Menge von Booleschen Funktionen und Familien Boolescher Funktionen. Ein *Boolescher Schaltkreis* über B mit n Eingängen und m Ausgängen ist ein 5-Tupel $C = (V, E, \alpha, \beta, \omega)$ [41], wobei

- (V, E) ein endlicher, gerichteter, azyklischer Graph ist,
- $\alpha\colon E \to \mathbb{N}$ eine injektive Funktion ist,
- $\beta\colon V \to B \cup \{x_1, \ldots, x_n\}$,
- $\omega\colon V \to \{y_1, \ldots, y_m\}$ eine partielle Funktion ist,

sodass die folgenden Eigenschaften gelten:

(i) Ist $v \in V$ ein Knoten mit Eingangsgrad 0, so ist $\beta(v) \in \{x_1, \ldots, x_n\}$ oder $\beta(v) = f \in B$ ist eine 0-stellige Boolesche Funktion.

(ii) Ist $v \in V$ vom Eingangsgrad $k > 0$, so ist $\beta(v)$ eine k-stellige Boolesche Funktion oder eine Familie Boolescher Funktionen.

(iii) Für $1 \leq i \leq n$ gibt es höchstens ein $v \in V$ mit $\beta(v) = x_i$.

(iv) Für $1 \leq i \leq m$ gibt es genau ein $v \in V$ mit $\omega(v) = y_i$.

Definition 19. Ein Schaltkreis aus Toffoli-Gattern über eine Basis B mit n Eingängen und m Ausgängen ist ein 5-Tupel $T = (V, E, \alpha, \beta, \omega)$, wobei

- (V, E) ein endlicher, gerichteter, azyklischer Graph ist,
- $\alpha\colon E \to \mathbb{N}$ eine bijektive Funktion ist,
- $\beta\colon V \to B \cup \{x_1, \ldots, x_n\}$,
- $\omega\colon V \to \{y_1, \ldots, y_m\}$ eine partielle Funktion ist,
- $n = m$,

sodass die Eigenschaften (i) und (ii) aus Definition 18 gelten, da die Funktion jetzt bijektiv ist, gilt statt (iii) und (iv) nun

(v) Für $1 \leq i \leq m = n$ gibt es genau ein $v \in V$ mit $\omega(v) = y_i$ und genau ein $v \in V$ mit $\beta(v) = x_i$

Die Größe eines Schaltkreises wird bestimmt durch die Anzahl der Gatter. Um unitäre Transformationen in Quantenschaltkreisen zu realisieren, werden Toffoli-Gatter eingesetzt [17, 22].

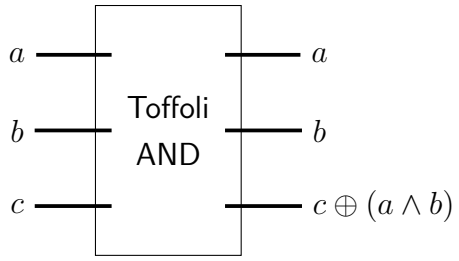


Abbildung 3: Das Toffoli-Gatter

Mit diesem Toffoli-Gatter lassen sich NOT und AND sehr leicht darstellen indem wir Bits konstant setzten. Um ein NOT darzustellen, setzen wir $c = 1$ und $b = 1$, somit gilt für den dritten Ausgang:

$$1 \oplus (a \wedge 1) = \bar{a}.$$

Um die Darstellung eines AND-Gatters zu realisieren, setzen wir $c = 0$, so gilt für den dritten Ausgang:

$$0 \oplus (a \wedge b) = a \wedge b.$$

Ein OR lässt sich durch drei Toffoli-Gatter darstellen, da wir auf Grund des Gesetzes von *de Morgan* $a \vee b$ durch $\overline{\bar{a} \wedge \bar{b}}$ darstellen können [42].

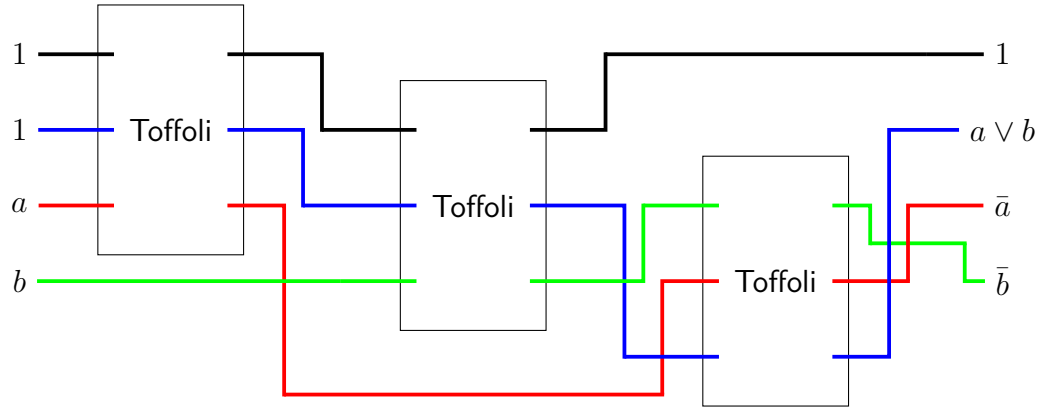


Abbildung 4: Drei Toffoli Gatter bilden ein OR

Die umkehrbaren Operationen sind also Permutationen der Eingabebits und damit unitär. Durch AND, OR und NOT lassen sich alle uns bekannten Bausteine zusammensetzten [22]. Aus den eben gezeigten Gattern und Darstellungen lässt sich also folgender Satz ableiten.

Satz 20. Klassische Schaltkreise können in unitäre Schaltkreise mit polynomiell mehr Gattern überführt werden.

Nun haben wir also sichergestellt, dass die Überführung von klassischen Schaltkreisen in Quantenschaltkreise nur polynomiell mehr Aufwand bedeutet und daher nach unserem Verständnis weiterhin *effizient* ist.

4. Quantenalgorithmen

Es gibt bereits einige Algorithmen, die sich quantenmechanische Effekte zu nutzen machen. In diesem Abschnitt werden Teile der drei Algorithmen von Shor, Grover und Simons betrachtet, die für die Kryptographie von Relevanz sind. Diese Algorithmen lösen uns bereits bekannte Probleme effizienter als es klassischen Algorithmen zur Zeit möglich ist. Um dies auch zu zeigen, betrachten wir bei Shors Algorithmus die gesamte Laufzeit des klassischen und Quantenanteils. Die Laufzeiten, der jeweiligen Teile wird ebenso aufgezeigt, um zu verdeutlichen wie schnell diese Algorithmen Berechnungen durchführen können.

4.1. Shor

Satz 21. Das Faktorisierungsproblem liegt in **NP** [24].

Das ist schon länger bekannt und viele Verschlüsselungsalgorithmen, unter anderem RSA [34], verlassen sich darauf.

Satz 22. Das Faktorisierungsproblem kann durch Quantenalgorithmen in Polynomialzeit gelöst werden und liegt somit in **BQP**.

Oder anders formuliert: Shors Algorithmus kann Zahlen in polynomieller Laufzeit faktorisieren. Um Satz 22 zu beweisen, unterziehen wir Shors Algorithmus einer sehr genauen Prüfung. Die Rechnungen, die der Quantencomputer ausführen muss, haben wir bereits in den vorherigen Kapitel 3 ausführlich betrachtet und können damit arbeiten. Shors Algorithmus nutzt ein Quantenorakel (vgl. Definitionen 8 und Abschnitt B), um die Perioden in der Funktion zu finden. Zu der Suche von Perioden kommen noch einige Operationen hinzu, die ein klassischer Computer problemlos lösen kann.

Zur besseren Übersicht betrachte man Abbildung 5 auf Seite 27. Sollte a ungünstig gewählt worden sein, so ist es unmöglich einen Teiler von n zu ermitteln. Der Algorithmus sorgt dann dafür, dass wir erneut ein a wählen müssen. Zeile 3 und 4 filtern direkt alle a mit ggT(a, n) heraus, sodass der Quantenteil übersprungen werden kann. Die restlichen

Algorithm 1 Shors Algorithmus ohne ausführlichen Quantenanteil [22]

```
1: loop
2:     a ← 2, n ..., n − 1                ▷ Wähle eine zufällige Zahl a aus 2, ..., n − 1
3:     z ← ggT(a, n)
4:     if z ≠ 1 then return z
5:     end if
6:     p ← Q(a^x mod n)                   ▷ Ermittle die Periode p mittels Quantenorakel
7:     if p mod 2 = 0 then
8:         z ← ggT(a^(p/2) − 1, n)
9:         if z ≠ 1 then return z
10:        end if
11:        z ← ggT(a^(p/2) + 1, n)
12:        if z ≠ 1 then return z
13:        end if
14:    end if
15: end loop
```

Schritte haben eine Wahrscheinlichkeit von $\frac{1}{2}$ einen Teiler zu finden [22]. Zeile 2 des Algorithmus können wir als konstant voraus setzten. Zeilen 3, 8 und 11 sind zu lösen mit dem euklidischen Algorithmus und dieser hat die Laufzeit $O\Big((\log n)^3\Big)$ (gezeigt in [8]). Zeile 6 ist der Quantenanteil und wird später betrachtet. In Zeilen 8 und 11 kann $a^{\frac{p}{2}}$ eine sehr große Zahl sein, stattdessen berechnen wir besser $a^{\frac{p}{2}} \mathit{mod} n$, was wiederum mit $O\Big((\log n)^3\Big)$ Bitoperationen möglich ist. Da bei Laufzeiten immer nur die Mächtigste von Bedeutung ist, hat der gesamte Algorithmus eine Laufzeit von $O\Big((\log n)^3\Big)$. Dieser findet jedoch nur mit einer Wahrscheinlichkeit von mindestens $\frac{1}{2}$ einen Teiler von n. Folgern wir also, dass die Wahrscheinlichkeit nach k Durchläufen noch keinen Teiler gefunden zu haben bei $\frac{1}{2}^k$ liegt. Je häufiger der Algorithmus hintereinander ausgeführt wird, desto höher ist die Wahrscheinlichkeit einen Teiler zu finden. Die zu erwartende Laufzeit ergibt sich dadurch als (vgl. auch Abbildung 6)

$$\sum_{k\geq 1} \frac{k}{2^k} \cdot O\Big((\log n)^3\Big) = O\Big((\log n)^3\Big). \tag{4.1}$$

Dies ist ohne Betrachtung des Quantenteils und daher schon länger bekannt [31].

Der schraffierte Teil in Abbildung 5 wird durch einen Quantenalgorithmus gelöst, alle anderen Punkte lassen sich durch die bereits erwähnten klassischen Operationen lösen. Der Quantenanteil des Algorithmus besteht aus sechs Schritten.

Der entscheidende Punkt ist, dass wir von Anfang an mit zwei Registern arbeiten, von denen aber nur das erste Register durch das Anwenden der Hadamard-Transformation

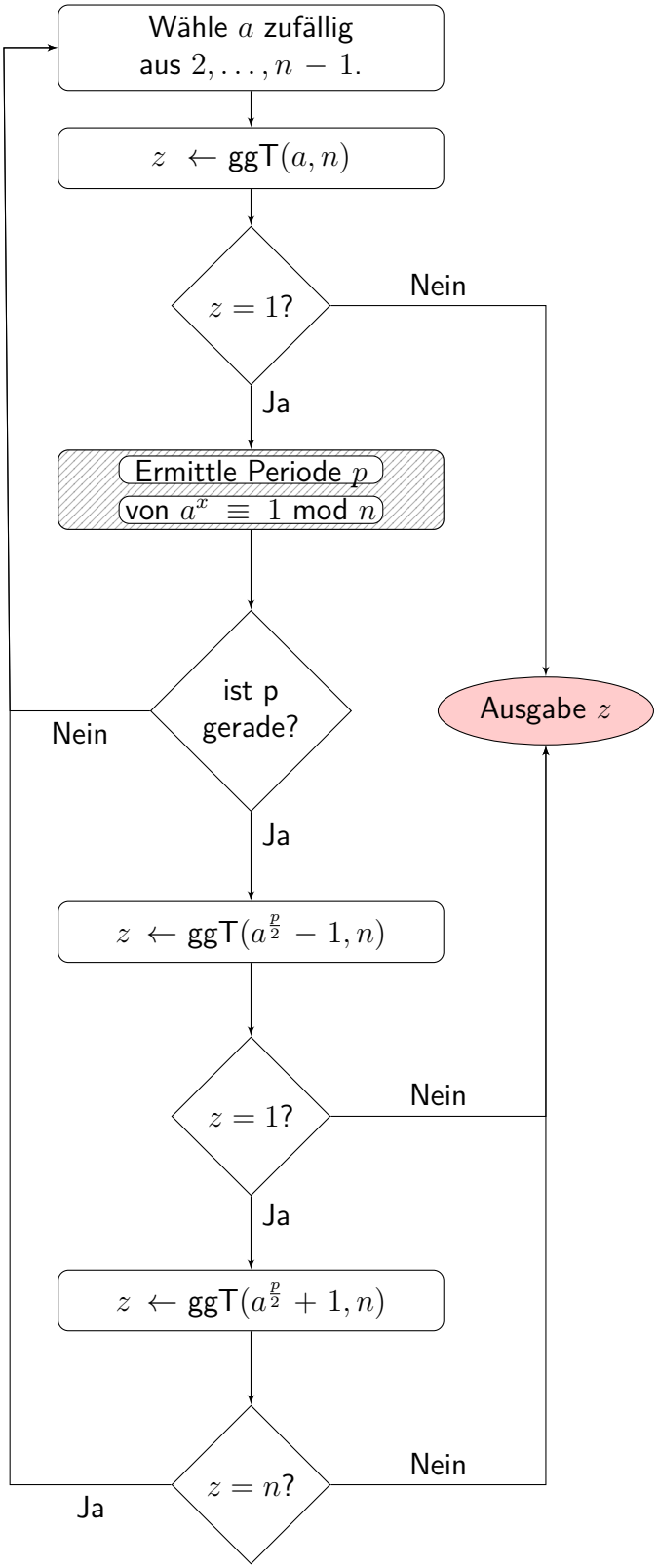


Abbildung 5: Shors Algorithmus als Diagramm [22]

Algorithm 2 Quantenanteil von Shors Algorithmus [22]

1: $R = |a\rangle|b\rangle \leftarrow |0\ldots0\rangle|0\ldots0\rangle$
2: $R \leftarrow \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle|0\ldots0\rangle$ ▷ Hadamard-Transformation nur auf a
3: $R \leftarrow U_f R = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle|f(x)\rangle$
4: Messe Register b.
5: $|a\rangle \leftarrow QFT_N|a\rangle$
6: Messe Register a und gebe den Inhalt aus

in eine Superposition versetzt wird. Neben der Anwendung des Orakels in Schritt 3 werden beide Register miteinander verschränkt, da dort $a^x \,mod\, N$ berechnet wird und das Ergebnis in Register b gespeichert wird. Durch das Messen in Schritt vier wird ein Wert $|a^{x_0} \,mod\, N\rangle$aus dem Inhalt des Registers b gemessen, welcher als x_0 immer der Kleinste ist. Durch die Verschränkung wird der Zustand in die Superposition des ersten Registers projeziert. In Schritt 5 wird die Quanten-Fourier-Transformation auf das erste Register angewendet, um die Periode heraus zu transformieren. Durch Schritt 6 wird konstanter Aufwand generiert, dieser ist vernachlässigbar gering.

Die Quanten-Fourier-Transformation nutzt die Effekte des Quantenparallelismus, um die diskrete Fourier-Transformation zu berechnen. Die QFT ist insbesondere dann hilfreich, wenn Informationen in der Periode von einer effizient berechenbaren Funktion liegen. Diese Informationen werden dann mit Hilfe der QFT extrahiert. Diese ist eng mit dem *hidden subgroup problem* (HSP) verknüpft. Das HSP beschäftigt sich mit dem Finden von Perioden in bereits gegebenen periodischen Funktionen f auf $\mathbb{Z}$, oder anders formuliert: Es soll eine *hidden subgroup* $\ell\mathbb{Z} \subset \mathbb{Z}$ mit den kleinsten Indizes gefunden werden, sodass f konstant ist mit $\ell\mathbb{Z} + a$. Durch mehrfaches Anwenden wird so die Periode gefunden, oder zumindest die Periode faktorisiert. Für unsere Zwecke soll diese Betrachtung auch ausreichen, die QFT wird bereits in einigen Veröffentlichung ganz genau betrachtet [9, 32].

Schritt 1 und 2 des Quantenanteils von Shors Algorithmus werden auf $2 \cdot 2 \cdot \log n$ Quantenbits ausgeführt, der dritte Schritt wiederum nur auf $\log n$ Bits. Die Hadamard-Transformation ist somit von $2 \cdot \log n$ und die Quanten-Fouriertransformation von $O\Big((\log n)^2\Big)$ Gattern realisierbar [22]. Am aufwändigsten ist daher Schritt 2, da es klassische Schaltkreise gibt, die modulare Potenzen mit $O\Big((\log n)^3\Big)$ Gattern realisieren können. Dementsprechend ergibt sich für den Quantenanteil von Shors Algorithmus eine Laufzeit von $O\Big(\log\log n \cdot (\log n)^3\Big)$. Das Ergebnis muss nun noch ausgewertet werden, da es das Ziel war die Periode zu bestimmen. Hierfür ist jedoch der klassische Anteil des Algorithmus

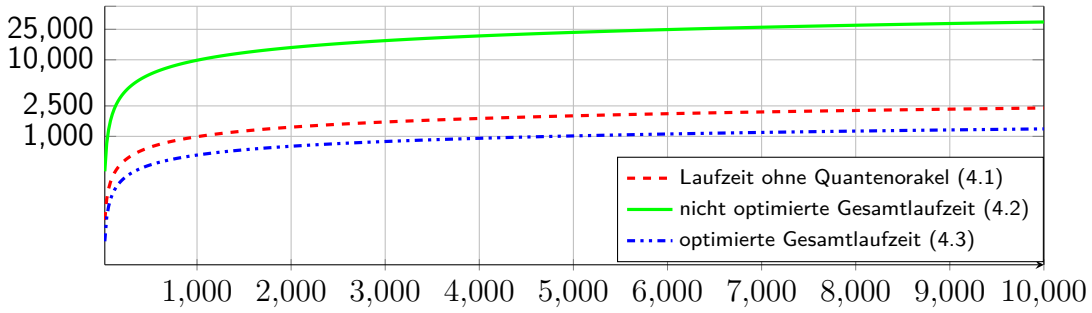


Abbildung 6: Die Laufzeit von Shors Algorithmus

wieder zuständig und es wird nicht länger ein Quantencomputer benötigt. Die Laufzeit dafür haben wir bereits bestimmt, sodass sich nun zusammengesetzt aus der Laufzeit des Quanten- und des klassischen Teils die gesamte Laufzeit des Algorithmus bestimmen lässt. Diese beläuft sich somit auf:

$$O\Big(\log\log n \cdot (\log n)^3\Big) \leq 3O\Big((\log n)^4\Big). \tag{4.2}$$

Es gibt noch ein Multiplikationsverfahren, welche die Laufzeit noch etwas senken kann, sodass sie auf

$$O\Big((\log n)^2 \cdot \log\log n \cdot \log\log\log n\Big) \tag{4.3}$$

reduziert werden kann [22]. Diese beiden Laufzeiten unterscheiden sich auf den ersten Blick nicht sonderlich, daher werden sie in Abbildung 6 noch einmal anschaulicher dargestellt. Hier wir eindeutig klar, wie stark der Exponent in die Laufzeit des Verfahrens eingeht. Durch das verbesserte Multiplikationsverfahren senken wir die Laufzeit sogar noch unter die Zeit, die der reine klassische Anteil benötigt, da auch die Multiplikationen im klassischen Teil davon profitieren. Durch die Analyse von Shors Algorithmus lässt sich nun feststellen, dass dieser eindeutig in polynomieller Laufzeit ein Ergebnis berechnen kann und daher in der Laufzeitklasse **BQP** liegen muss. Man vergleiche mit Abschnitt 2.1, dort wurde die Laufzeitklasse näher betrachtet und es hat sich gezeigt, dass $\mathbf{BQP} \subseteq \mathbf{NP}$. Also lässt sich das Faktorisierungsproblem nun in der Klasse **BQP** lösen, Probleme dieser Klasse lassen sich von Quantencomputern in polynomieller Laufzeit lösen (vgl. Definition 8). Es ist also eindeutig gezeigt, dass Satz 22 gilt.

Dies hat Auswirkungen auf einige der gängigen Verschlüsselungsverfahren wie z. B. RSA, da dieses Verfahren darauf beruht, dass die Faktorisierung von großen Zahlen ein erhebliches Problem darstellt (also in **NP** liegt) [34]. Außerdem sei noch gesagt,

dass durch einige Veränderungen ähnliches für das Berechnen des diskreten Logarithmus gilt. Auch dieser lässt sich durch einen Quantencomputer in polynomieller Laufzeit berechnen [39]. Das bedeutet allerdings nicht, dass nur Quantencomputer das Faktorisierungsproblem in polynomieller Laufzeit lösen können. Es ist durchaus möglich, dass ein klassischer Algorithmus existiert, dieser einfach nur noch nicht gefunden wurde. Es ist sogar möglich, dass zu jedem Quantenalgorithmus ein klassischer Algorithmus existiert, uns bis jetzt aber noch keine Verfahren zur Überführung bekannt sind.

4.2. Grover

Grovers Algorithmus beschäftigt sich mit der Suche nach Elementen in großen Datenmengen, vergleichbar mit der Suche nach der Nadel im Heuhaufen. So kann Grovers Algorithmus eingesetzt werden, um Einträge in einer Datenbank nach einem Kriterium zu durchsuchen, nachdem die Datenbank nicht sortiert ist. Ein verständliches Beispiel hierfür ist ein Telefonbuch, dort werden die Nummern den Namen zugeordnet, welche alphabetisch sortiert sind. Sollte man nun jedoch eine bestimmte Nummer haben und den dazugehörigen Namen suchen, so eignet sich diese Struktur dafür nicht. Genau hier kommt Grovers Suchalgorithmus ins Spiel. Dieser Algorithmus ist für die Kryptographie relevant, da diese Datenbanksuche auch als die Suche nach einem passenden Schlüssel betrachtet werden kann [8, 9]. Grovers Algorithmus besteht im Wesentlichen aus einem konstanten Anteil und der Grover Iteration.

Mathematisch gesprochen soll ein Urbild eines Wertes s einer booleschen Funktion f gefunden werden. Die Funktion f können wir zwar aufrufen, aber nicht betrachten. Hierbei wird die Funktion f mehrfach ausgewertet, diese Auswertung ist proportional zu der Kardinalitäten N (die Urbilder $f^{-1}(s)$) und M (die Größe der Datenmenge). Grovers Algorithmus ist dann dazu in der Lage die Komplexitätsklasse dieser Funktion um den Faktor $\sqrt{\frac{N}{M}}$ zu senken [9].

$$G = U_f \cdot (H^{\otimes n}(2 \cdot |0\rangle \cdot \langle 0| - 1)H^{\otimes n}) \otimes \mathit{Id}. \tag{4.4}$$

Die Grover Iteration G sorgt dafür, dass das gesuchte Element invertiert wird, da die Amplitude in das Vorzeichen verlagert wurde. G^r bedeutet somit, dass G r mal hintereinander ausgeführt wird. Beim Spiegeln am Mittelwert erfolgt dann der entscheidende Schritt. Da nur die Amplitude des gesuchten Elements negativ ist, wird auch nur diese sich durch die Spiegelung am Mittelwert aller Amplituden von den anderen abheben. Wird die Grover-Iteration nicht häufig genug angewendet, unterscheidet sich die Am-

Algorithm 3 Grovers Suchalgorithmus [9]

Input: Boolesche Funktion $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ mit den Operatoren;
$U_f\colon \mathbb{F}_2^n \times \mathbb{F}_2 \to \mathbb{F}_2^n \times \mathbb{F}_2^s\colon |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle$ und $M = \mathit{card} f^{-1}(1)$.
Output: Eins oder mehrere $y \in \mathbb{F}_2^n$ mit $f(y) = 1$.

if $M > \frac{3}{4} \cdot 2^n$ **then**
    wähle y zufällig und bilde ein uniformes F_2^n **return** y
end if
Wähle θ so, dass $\sin^2\theta = \dfrac{M}{2^n}$.
$r \leftarrow \left\lfloor \dfrac{\pi}{4 \cdot \theta} \right\rfloor$
$|0\rangle|1\rangle \xrightarrow{H\otimes(n+1)} \dfrac{1}{\sqrt{2^n+1}} \sum\limits_{x\in\mathbb{F}_n^2} |x\rangle(|0\rangle - |1\rangle) \xrightarrow{G^r} \dfrac{1}{\sqrt{2^{n+1}}} \sum\limits_{x\in\mathbb{F}_n^2} \alpha_x|x\rangle(|0\rangle - |1\rangle)$
▷ G^r ist die r-fache Grover Iteration (4.4)
Ergebnis messen und die ersten n Bit ausgeben.

plitude nicht deutlich genug von den anderen, um sie zu finden. Sollte hingegen die Grover-Iteration zu häufig angewendet werden, so wird irgendwann die Amplitude sich wieder verkleinern. Daher wird das ganze Verfahren auch gerne mit einem Soufflé verglichen, welches im Ofen zubereitet wird [22]. Bevor die Grover-Iteration angewendet werden kann, muss noch herausgefunden werden, wie oft diese ausgeführt werden sollte, dafür ist es notwendig die Größe der zu durchsuchenden Datenmenge (M) zu kennen.

Der Algorithmus kann auch eingesetzt werden, um herauszufinden, ob ein bestimmter Wert im Bild einer Funktion f liegt. Insbesondere dadurch werden Hash-Funktionen geschwächt, da dies genutzt werden kann, um nach Kollisionen von zwei Funktionen zu suchen. Zusätzlich haben sich bereits einige Wissenschaftler damit auseinandergesetzt wie man Grovers Algorithmus zum Finden von eben diesen Kollision optimieren kann. Bei einer Hashfunktion mit der Outputlänge n und der Urbildmenge N benötigt dieser angepasste Algorithmus nur $O\left(\sqrt[3]{\frac{N}{n}}\right)$ Auswertungen der Hashfunktion, um in der Lage zu sein Kollisionen zu finden [7].

4.3. Simon

Ein weiterer Quantenalgorithmus soll hier noch kurz erwähnt werden, Simons Algorithmus. Dieser Algorithmus gab die Grundidee zu Shors Algorithmus und setzt sich mit dem Finden einer Periode innerhalb einer Funktion auseinander. Auch hierbei handelt es sich um ein *hidden subgroup problem (HSP)*. Die Funktion $f\colon \{0,1\}^n \to \{0,1\}^n$ ist eine

Zwei-zu-Eins Funktion. Die dazugehörige Periode a wird deshalb im Vektorraum $(\mathbb{Z}_2)^n$ gesucht. Aufgrund dieser Zwei-zu-Eins-Funktion kann der Urbildraum in zwei gleichwertige Teile aufgeteilt werden. Die genaue Funktionsweise ist für uns hier nicht wichtig, dennoch sei gesagt, dass beim Messen nur orthogonale Superpositionen zu der gesuchten Periode gefunden werden. Dies wird mehrfach ausgeführt, um ein Gleichungsystem aufzustellen, welches schlussendlich gelöst werden kann. Da es sich hier um den Vektorraum $(\mathbb{Z}_2)^n$ handelt, sind noch einige besondere Bedingungen einzuhalten [22]. Der Algorithmus selber hat wenig praktische Anwendungen, ist nur eine allgemeine Herangehensweise an das *HSP* und kann als Grundlage für andere Algorithmen dienen.

4.4. Einfluss auf bekannte Kryptographieverfahren

Alle Kryptographieverfahren, die in Tabelle 7 aufzeigt werden, können von Quantenalgorithmen in polynomieller Zeit rückberechnet werden oder müssen sich anpassen, um weiterhin sicher zu bleiben. Eine der am meisten genutzten Verschlüsselungen (RSA) basiert auf der Annahme, dass das Faktorisieren von großen Zahlen sehr viel Rechenaufwand bedeutet und damit viel Zeit kostet [34]. Verschlüsselungen, die auf dieser Annahme basieren, können somit nicht länger als *sicher* angesehen werden. Nicht alle Verfahren versagen nur wegen Shors Algorithmus zur Faktorisierung großer Zahlen, andere auch auf Grund der Beschleunigung in der Berechnung des Diskreten Logarithmus. Wieder andere Verfahren werden bedroht durch die beschleunigte Suche möglich durch Grovers Algorithmus. Dadurch werden viele Nutzer gezwungen ihre Verfahren zu überarbeiten oder komplett auszutauschen.

Krypto	**Art**	**Ziel**	**Anpassung**
RSA ECDSA, ECDH (Elliptic Curve Crypto) DSA (Finite Field Crypto)	Öffentliche und private Schlüssel (public private key)	Signaturen und Schlüsselerstellung	Kein Anpassung möglich. Quantencomputer brechen diese Verfahren.
AES	Symmetrisch	Verschlüsselung	Größere Schlüssel
SHA-3	Hashfunktion	-	Längere Outputs

Tabelle 7: Kryptographieverfahren unter Einfluss von Quantencomputern [10].

4.4.1. Vier Ansätze für Post-Quantum Kryptographie

Dennoch gibt es Verfahren, die ohne Bedenken weiter eingesetzt werden können. Dazu gibt es vier verschiedene Ansätze, die sich zu bewähren scheinen:

1. Multivariante Verfahren
2. Code-basierete Verfahren
3. Lattice-basierete Verfahren
4. Hash-basierete Verfahren

Die ersten drei Verfahren werden im Laufe dieser Abschlussarbeit jedoch nicht weiter betrachtet. Insbesondere beschäftigen wir uns im nächsten Kapitel 5 mit den Hash-basierten Verfahren. Wir suchen dort nach Alternativen insbesondere für digitale Signaturen, die fälschungssicher bleiben sollen. Zusätzlich gibt es auch schon Ansätze, die sich nicht nur damit beschäftigen wie Kryptographieverfahren von Quantencomputern bedroht werden und was geändert werden muss, sondern eher damit wie man Quantencomputer einsetzten kann, um neue wirklich sichere Verfahren umzusetzen. Dazu werden unter anderem die Effekte der Verschränkungen, die in Kapitel 3.5 erklärt werden genutzt, genaueres dazu in Kapitel 6.

5. Hash-basierte Verfahren

Dieses Kapitel setzt sich ausführlich mit kryptographischen Hashfunktionen auseinander. Es wird sowohl geklärt welche Anforderungen an eine Hashfunktion gestellt werden, damit sie von kryptographischem Interesse ist, als auch welche Vorteile eine Hashfunktion für den kryptographischen Einsatz haben kann. Hashfunktionen werden insbesondere dann eingesetzt, wenn es darum geht etwas zu verifizieren oder zu authentifizieren. Es ist auch möglich mit Hilfe von Hashfunktionen sehr schnell die Integrität von Dateien zu überprüfen.

Beispiel 23. Alice bearbeitet ein Projekt an einem Arbeitsplatz, auf den sie nicht durchgängig Zugriff hat. Abends, bevor sie den Arbeitsplatz verlässt, erstellt sie einen Hash von ihrem Projekt mit Hilfe einer kryptographischen Hashfunktion, diesen Hash sichert sie und nimmt ihn mit. Wenn sie am nächsten Tag wieder beginnen möchte zu arbeiten, berechnet sie erneut den Hash. Wenn dieser neu berechnete Hash nun von dem Hash des Vorabends abweicht, weiß sie, dass ihr Projekt verändert wurde. Sollte er übereinstimmen kann sie sicher sein, dass ihr Projekt noch genau so vorliegt wie am Vorabend. Die Integrität des Projektes wurde somit auf die Integrität des Hashes reduziert, dessen Integrität deutlich schneller festzustellen ist.

Passwörter werden schon lange nicht mehr in Klartext abgespeichert, dies ist viel zu riskant falls ein Angreifer Zugriff auf die gesamte Passwortdatei erlangt. Damit diese Datei zwar verschlüsselt ist, aber es dennoch möglich ist zu überprüfen, ob Nutzer sich mit dem richtigen Passwort anmelden, werde die Hashwerte der Passwörter berechnet und nur diese dann gespeichert. Man sagt dann auch: die Passwörter wurden *gehashed*. Um potenziellen Angreifern die Möglichkeit zu nehmen Hashwerte für gängige Passwörter vorzuberechnen, werden heutzutage meist noch zusätzlich *Salts* verwendet. Ein Salt ist eine zufällig gewählte Zeichenfolge, die zusammen mit dem Passwort gehashed wird. Dieser Salt wird dann gemeinsam mit dem Hashwert in der Passwortdatei abgelegt. Auf diese Art und Weise generieren zwei identische Passwörter trotzdem unterschiedliche Hashwerte und das Vorberechnen von Passwörtern wird unterbunden [8].

5.1. Grundprinzip

Definition 24. Eine *Hashfunktion* h ist eine Abbildung über dem Alphabet Σ

$$h\colon \Sigma^* \to \Sigma^n, n \in \mathbb{N}$$ *[8].*

Eine Hashfunktion bildet eine beliebige Anzahl von Elementen eines Alphabetes (beliebig lange Wörter) auf eine konstante Menge an Elementen eines Alphabetes ab (Wörter konstanter Länge). Folglich kann eine Hashfunktion niemals injektiv sein.

Betrachten wir nun den Definitionsbereich D, dieser ist bei der Hashfunktion h gegeben als $D = \Sigma^*$. Damit die Funktion h kryptographisch relevant sein kann, verlangen wir, dass die Werte von $h(x)$ für alle $x \in D$ effizient berechenbar sind. Da dies eine Notwendigkeit ist, setzen wir das auch im Folgenden für alle Funktionen, die wir betrachten, voraus.

Eine *Kollision* von h ist ein Paar $(x, x') \in D^2$ von Wörtern aus Σ, für die $x \neq x'$ und $h(x) = h(x')$ gilt. Alle Hashfunktionen (und Kompressionsfunktionen) besitzen Kollisionen, weil sie nicht injektiv sind.

Die Funktion h heißt *schwach kollisionsresistent*, wenn es praktisch unmöglich ist, für ein vorgegebenes $x \in D$ eine Kollision (x, x') zu finden.

Erweitern wir nun diesen Gedanken und sagen, dass es unmöglich sein soll irgendeine Kollision (x, x') von h zu finden, so ist die Funktionen h *(stark) kollisionsresistent*.

Jede Funktion, die stark kollisionsresistent ist, ist somit auch schwach kollisionsresistent. In vielen Anwendungen, die wir hier betrachten werden, ist es notwendig eine Funktion zu nutzen, die stark kollisionsresistent ist. Insbesondere bei digitalen Signaturen ist dies von signifikanter Bedeutung (vgl. Abschnitt 5.4).

5.2. Einwegfunktionen

Die Funktion h aus Abschnitt 5.1 heißt darüber hinaus *Einwegfunktion*, wenn es „praktisch unmöglich“ ist zu einem $s \in \Sigma^n$ ein $x \in D$ mit $h(x) = s$ zu finden. Es ist sehr schwierig eindeutig zu beweisen, ob eine Funktion eine tatsächliche Einwegfunktion ist. Daher bedeutet praktisch unmöglich in diesem Zusammenhang, dass ein Problem P unlösbar bis zum Jahr a ist. Dies gilt, wenn alle probabilistischen Algorithmen A, die im Worst-Case die Laufzeit$\leq 2^{k(a)}$ haben, P höchstens mit einer Erfolgswahrscheinlichkeit von $(\frac{1}{2})^{k(a)}$ lösen.

Ob vollwertige Einwegfunktionen existieren ist noch nicht bekannt, allerdings gibt es

Funktionen, die nach heutigem Kenntnisstand als Einwegfunktionen angesehen werden können. Ihre Funktionswerte lassen sich leicht berechnen und es ist noch kein Algorithmus bekannt, der diese effizient rückberechnen kann.

Eine Einwegfunktion muss nicht zwangsläufig kollisionsresistent sein, da die Funktion für uns aber von praktischem Interesse sein muss, beachten wir ausschließlich Einwegfunktionen, die kollisionsresistent sind.

Eine nachweisliche Einwegfunktion würde alle Verfahren, die auf Hashfunktionen basieren, zu 100% sicher machen [35]. Eine Einwegfunktion, die Eins-zu-Eins abbilden würde, wäre somit die beste anzunehmende Funktion. Eins-zu-Eins bedeutet dann allerdings, dass die Funktion h jeden Wert nur ein Mal treffen darf. Da allerdings ein beliebig langer String eines Alphabetes auf einen mit konstanter Länge abgebildet wird, ist dies praktisch nicht umsetzbar und für Hashfunktionen auch nicht erwünscht. Erinnern wir uns in diesem Zusammenhang an Definition 9, dort haben wir bereits gezeigt, dass die Laufzeitklasse $\mathbf{P} \subseteq \mathbf{UP}$ ist. Damit Einwegfunktionen nun wirklich sicher sind verlangen wir somit, dass gilt: $\mathbf{P} \subset \mathbf{UP}$. Wenn $\mathbf{P} = \mathbf{UP}$ gezeigt werden sollte, lassen sich Verfahren, die auf Einweg-Hashfunktionen basieren, nicht länger als vollständig sicher ansehen. Eine Eins-zu-Eins Abbildung wäre ebenfalls nicht mehr möglich. Wenn Hashfunktionen jedoch nicht länger nur in eine Richtung berechenbar sind, kann kein Verfahren, welches sich auf die Sicherheit und Kollisionsresistenz von Hasfunktionen verlässt, weiterhin sicher sein.

5.3. MACs

Definition 25. Ein *Message-Authentication-Code-Algorithmus* oder *MAC-Algorithmus* ist ein Tupel **(K, M, S, KeyGen, Mac, Ver)** mit folgenden Eigenschaften [8]:

1. **K** ist eine Menge. Sie heißt *Schlüsselraum*. Ihre Elemente heißen *Schlüssel*.

2. **M** ist eine Menge. Sie heißt *Nachrichtenraum*. Ihre Elemente heißen *Nachrichten*.

3. **S** ist eine Menge. Sie heißt *MAC-Raum*. Ihre Elemente heißen *MACs*.

4. **KeyGen** ist ein probabilistischer Algorithmus. Er heißt *Schlüsselerzeugungsalgorithmus*.

5. **Mac** ist ein probabilistischer Algorithmus, der zustandsbehaftet sein kann. Er heißt *MAC-Erzeugungsalgorithmus*. Bei Eingabe eines Schlüssels $K \in \boldsymbol{K}$ und einer Nachricht $M \in \boldsymbol{M}$ gibt er einen MAC $S \in \boldsymbol{S}$ zurück.

6. **Ver** ist ein deterministischer Algorithmus. Er heißt *Verifikationsalgorithmus*. Er gibt bei Eingabe eines Schlüssels $K \in \boldsymbol{K}$, einer Nachricht $M \in \boldsymbol{M}$ und eines Mac $S \in \boldsymbol{S}$ einen der Werte $b \in \mathbb{Z}_2$ aus. Der Mac S heißt *gültig*, wenn $b = 1$ ist und andernfalls *ungültig*.

7. Der MAC-Algorithmus verifiziert korrekt: für alle Schlüssel $K \in \boldsymbol{K}$, jede Nachricht $M \in \boldsymbol{M}$ und jeden MAC $S \in \boldsymbol{S}$ gilt $\mathbf{Ver}(K, M, S) = 1$ genau dann, wenn S ein Rückgabewert von $\mathbf{Mac}(K, M)$ ist.

Ein MAC-Algorithmus wird eingesetzt, um über die Verifikation einer Datei auch die Authentizität einer Nachricht zu überprüfen. MACs können aus kryptographischen Hashfunktionen konstruiert werden, ein bekanntes Beispiel hierfür sind *Key-Hash Message Authentication Codes (HMAC)* [3]. Betrachten wir dazu die kryptographische Hashfunktion

$$h\colon \mathbb{Z}_2^* \to \mathbb{Z}_2^n.$$

Es ist nun möglich daraus einen MAC zu bauen. Der Schlüsselerzeugungsalgorithmus wählt zufällig und gleichverteilt einen Schlüssel $K \in \boldsymbol{K}$. Dieser Schlüssel wird nun zusammen mit einer Nachricht $M \in \mathbb{Z}_2^*$ dem MAC-Erzeugungsalgorithmus übergeben. Ist $|K| > n$ so wird K durch $h(K)$ ersetzt. Sollte $|K| < n$ sein, so muss K mit Nullen ergänzt werden, sodass die Länge n beträgt. Der Mac-Erzeugungsalgorithmus generiert dann den Ausgabewert

$$h((K \oplus \textit{opad}) \circ (K \oplus \textit{ipad}) \circ M).$$

Die dabei genutzten Variablen opad und ipad sind hexadezimale Strings der Länge n von der Form $0x5c5c5c \ldots 5c5c5c$ und $0x363636 \ldots 3636$. Diese MACs können eingesetzt werden, um zu verifizieren von wem Nachrichten kommen, wenn vorher der geheime Schlüssel $K \in \mathbf{K}$ ausgetauscht wurde. Es ist sogar möglich MACs mit Hilfe von Blockchiffren zu konstruieren. Die Nachrichten werden dann im CBC-Modus verschlüsselt und der letzte Schlüsseltextblock als MAC verwendet [8].

MACs sind nur dann *sicher*, wenn es nicht möglich ist *existentielle Fälschungen* zu konstruieren. Als existentielle Fälschung sehen wir ein Paar (M, S) an, das aus einer Nachricht M und einem gültigen MAC S für M besteht [8]. Ein solches Paar müsste erstellt werden ohne den passenden geheimen Schlüssel zu kennen. Ist das Erstellen dieser Art von Fälschung nicht möglich, so können Angreifer ohne Zugriff auf den geheimen Schlüssel auch keine MACs für vorgegebene Nachrichten erstellen. Dabei ziehe man auch in Betracht, dass selbst bei einer nicht kollisionsresistenten kryptographischen

Hashfunktion (z. B. $h\colon \mathbb{Z}_2^* \to \mathbb{Z}_2^n$) die Erstellung eines sicheren MACs auf Grundlage dieser Hashfunktion möglich ist. Das gilt beispielsweise bei der vielfach eingesetzten Hashfunktion MD5 und dem dazugehörigen MD5 MAC [8].

5.4. Digitale Signaturen

Erweitern wir nun das Modell des MAC-Algorithmus aus Abschnitt 5.3 um das Prinzip des Public-Private-Key Verfahrens und betrachten digitale Signaturen. Diese können dann mit Hilfe des privaten Schlüssels signiert und von jeder Person, die den öffentlichen Schlüssel besitzt, verifiziert werden.

Definition 26. Ein *elektronisches* oder *digitales Signaturverfahren* ist ein Tupel **(K, M, S, KeyGen, Sign, Ver)** mit folgenden Eigenschaften [8]:

1. Analog zum MAC-Algorithmus sind **K** die Schlüssel, **M** die Nachrichten und **S** die Signaturen.

2. **KeyGen** ist ein probabilistischer Polynomialzeit-Algorithmus. Er heißt *Schlüsselerzeugungsalgorithmus*. Bei Eingabe von 1^k für ein $k \in \mathbb{N}$ gibt er ein Schlüsselpaar $(e, d) \in \boldsymbol{K}$. Wir schreiben dann $(e, d) \leftarrow \boldsymbol{KeyGen}(1^k)$. Mit $\mathbf{K}(k)$ bezeichnen wir die Menge aller Schlüsselpaare, die **KeyGen** bei Eingabe von 1^k zurück geben kann. Mit $\mathbf{Pub}(k)$ bezeichnen wir die Menge aller öffentlichen Schlüssel, die als erste Komponente eines Schlüsselpaares $(e, d) \in \boldsymbol{K}(k)$ auftreten kann. Die Menge aller zweiten Komponenten dieser Schlüsselpaare bezeichnen wir mit $\mathbf{Priv}(k)$. Außerdem bezeichnet $\boldsymbol{M}(d) \subset \boldsymbol{P}$ die Menge der Nachrichten, die mit einem privaten Schlüssel d signiert werden können.

3. **Sign** ist ein probabilistischer Polynomialzeit-Algorithmus, der zustandsbehaftet sein kann. Er heißt *Signieralgorithmus*. Bei Eingabe von $1^k, k \in \mathbb{N}$, eines privaten Schlüssels $d \in \boldsymbol{Priv}(k)$ und einer Nachricht $m \in \boldsymbol{M}(d)$ gibt er eine Signatur $s \in \boldsymbol{S}$ zurück.

4. **Ver** ist analog zur MAC, nur das jetzt der öffentliche Schlüssel genutzt wird.

5. Das digitale Signaturverfahren verifiziert immer korrekt; für alle $k \in \mathbb{N}$, jedes Schlüsselpaar $(e, d) \in \boldsymbol{K}(k)$, jede Nachricht $m \in \boldsymbol{M}(d)$ und jede Signatur $s \in \boldsymbol{S}$ gilt $\mathbf{Ver}(1^k, e, m, s) = 1$ genau dann, wenn s ein Rückgabewert von $\mathbf{Sign}(1^k, d, m)$ ist.

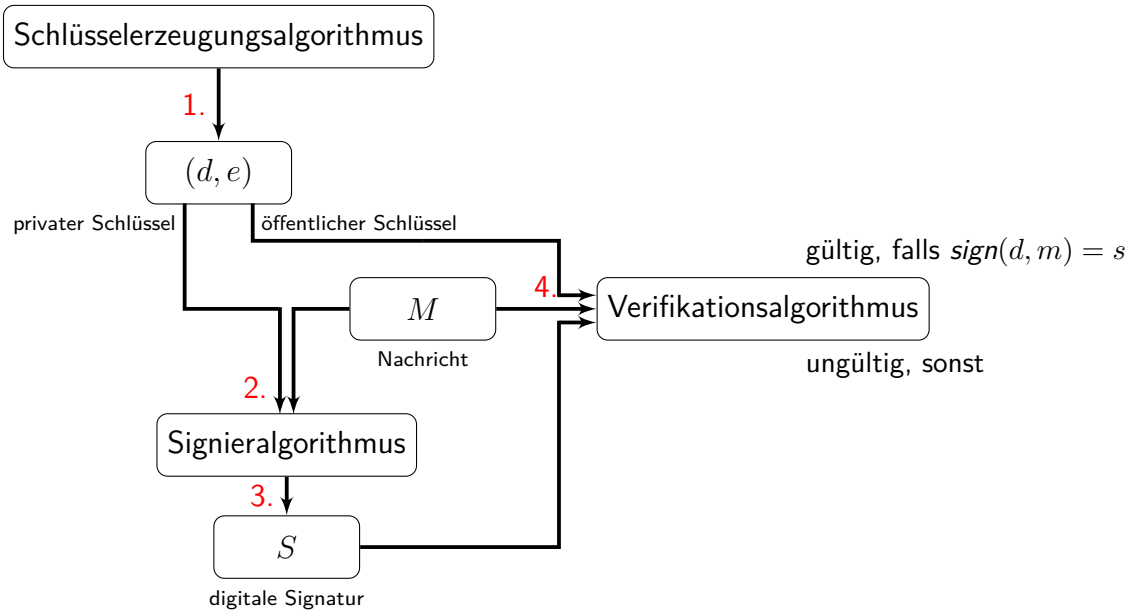


Abbildung 8: Digitales Signaturverfahren im Überblick

Das übliche Verfahren für die Erstellung einer digitalen Signatur nutzt den privaten Schlüssel, um die Signatur eines Dokumentes zu berechnen. Da das Dokument oder die Nachricht auch länger sein können, wird eine Hashfunktion eingesetzt, um davon einen Hash zu berechnen. Dieser Hash wird dann mit dem privaten Schlüssel verschlüsselt und stellt somit die Signatur dar. Diese Signatur wird dem entsprechenden Dokument angehängt. Der öffentliche Teil des Schlüssels kann nun jederzeit genutzt werden, um diese Signatur zu verifizieren. Dieses Verfahren wird in Abbildung 8 übersichtlich dargestellt. Die genutzte Hash-Funktion muss dabei die Bedingungen aus Abschnitt 5.1 erfüllen, da sonst das gesamte Verfahren deutlich angreifbarer wird [9]. Sichere digitale Signaturverfahren können nur dann wirklich sicher sein, wenn Einweg-Hashfunktionen existieren (vgl. Abschnitt 5.2).

Digitale Signaturen stellen sicher, dass signierte Dokumente auch wirklich von den Personen stammen, die diese Dokumente signiert haben. Dadurch können Informationen eindeutig bestimmten Personen zugeordnet werden. Was wiederum auch bedeutet, dass diese Dokumente später nicht vom Absender abgestritten werden können, dies ist insbesondere bei Verträgen notwendig. Außerdem wird die Integrität der Dokumente zusätzlich sicher gestellt, ebenso wie bei MACs.

Verfahren	Public-Keylänge	Signaturlänge
Lamport [12]	$2kn$	kn
Lamport* [12]	n	$2kn$
Merkles Verbesserung von Lamport [29]	$(k + \log_2(k))n$	$(\frac{k+\log_2(k)}{2})n$
Merkles Verbesserung von Lamport* [29]	n	$(k + \log_2(k))n$
Winternitz [14]	n	tn

Tabelle 9: Gängige Einmalsignaturverfahren im Vergleich [40]

Erläuterung zur Tabelle 9: n ist die von uns gewählte Schlüssellänge, k ist die Länge der Eingabe, die nötig war um ein Schlüsselpaar zu generieren und $t = t_1 + t_2$, zusammengesetzt aus der Länge der Signatur t_1 und den notwendigen Zusatzinformationen t_2 der Winternitz-Signatur [14].

5.4.1. Einmal-Signaturverfahren

Um die Sicherheit des Signaturverfahrens zu erhöhen, wurden Einmalsignaturen entwickelt. Diese sind fast unmöglich zu fälschen, denn es wird jedes mal ein neues Schlüsselpaar zum Signieren und Verifizieren verwendet. Das bedeutet aber auch, dass der Austausch der Schlüssel zu einem Problem wird. So lassen sich beispielsweise mit Hilfe eines „sicheren“ deterministischen pseudorandom number generator (PRNG) die Zufallszahlen für einen One-Time-Hash generieren. Der Seed des PRNG ändert sich dabei jedes mal, es muss unmöglich sein frühere Seeds zu generieren, sodass höchstens zukünftige aber auf keinen Fall alte Signaturen gefälscht werden können. Der Seed eines PRNG bestimmt die „Zufallszahlen“, die generiert werden. Diese können dann sowohl vom Signierer als auch vom Verifizierer identisch generiert werden und in die Berechnung mit einfließen. Der Seed wird durch jedes Signieren verändert, der Verifizierer muss also genau wissen welche Nachricht er überprüfen möchte und welcher Seed dieser Nachricht zugeordnet ist.

5.4.2. Lamport-Diffie

Da das Einsetzen von solchen PRNGs nicht sonderlich effizient ist, wurden auch andere Verfahren entwickelt. Die Lamport-Diffie Signatur untersuchen wir nun ein wenig genauer, denn sie wird auch später noch zusammen mit den Merkle Bäumen betrachtet (vgl. Abschnitt 5.5). Für die Erstellung des Schlüsselpaares wird eine Einwegfunktion f und

eine kryptographische Hashfunktion g benötigt [8, 9].

$$f\colon \{0,1\}^n \to \{0,1\}^n, n \in \mathbb{N},$$
$$g\colon \{0,1\}^n \to \{0,1\}^n, n \in \mathbb{N}.$$

Der Schlüssel zum Signieren besteht dann aus $2n$ Bitstrings der Länge n gewählt aus:

$$X = (x_{n-1}[0], x_{n-1}[1], \ldots, x_1[0], x_0[1], x_0[0], x_0[1]) \in_{\mathbb{R}} \{0,1\}^{(n,2n)}.$$

Der Verifikationschlüssel ist analog dazu:

$$Y = (y_{n-1}[0], y_{n-1}[1], \ldots, y_1[0], y_0[1], y_0[0], y_0[1]) \in \{0,1\}^{(n,2n)}$$

und unterliegt folgender Einschränkung

$$y_i[j] = f(x_i[j]), 0 \leq i \leq n-1, j = 0, 1.$$

Die Funktion f wird zum Erstellen des Schlüssels daher $2n$ mal berechnet.

Ein Dokument $M \in \{0,1\}^*$ wird signiert mit dem Schlüssel X, sodass $g(M) = d = (d_{n-1}, \ldots, d_0)$ der Hash die gekürzte Nachricht M ist. Die Lamport-Diffe Einmalsignatur σ_{LDots} besteht dann aus [8, 9]:

$$\sigma_{LDots} = (x_{n-1}[d_{n-1}], \ldots, x_1[d_1], x_0[d_0]) \in \{0,1\}^{(n,n)}.$$

Wie sich an diesen Gleichungen sehr leicht erkennen lässt, steht und fällt die Sicherheit der Lamport-Diffie Einmalsignatur mit der Sicherheit der Hashfunktion und der Einwegfunktion. Die Einwegeigenschaft ist eine maßgebliche Anforderung, da es ohne Einwegfunktion unmöglich ist sichere digitale Signaturen zu erzeugen [9, 35].

Das Lamport-Diffie Verfahren wurde von Ralph Merkle noch ein wenig verbessert [29], insbesondere was die Erstellung von Schlüsseln betrifft.

5.4.3. Begrenzte Menge an Signaturen

Einmal-Signtaturverfahren haben ein offensichtliches Problem, die Signaturen lassen sich nur einmal einsetzen. Daher muss vorab immer eine ausreichend große Menge an Signaturen generiert werden. Es gibt bereits einige Ansätze dieses Problem zu umgehen, unter anderem die sogenannten Merkle Bäume (vgl. nächster Abschnitt 5.5). Durch verschiedene Verfahren sind mehr Signaturen möglich als ursprünglich angenommen, sogar

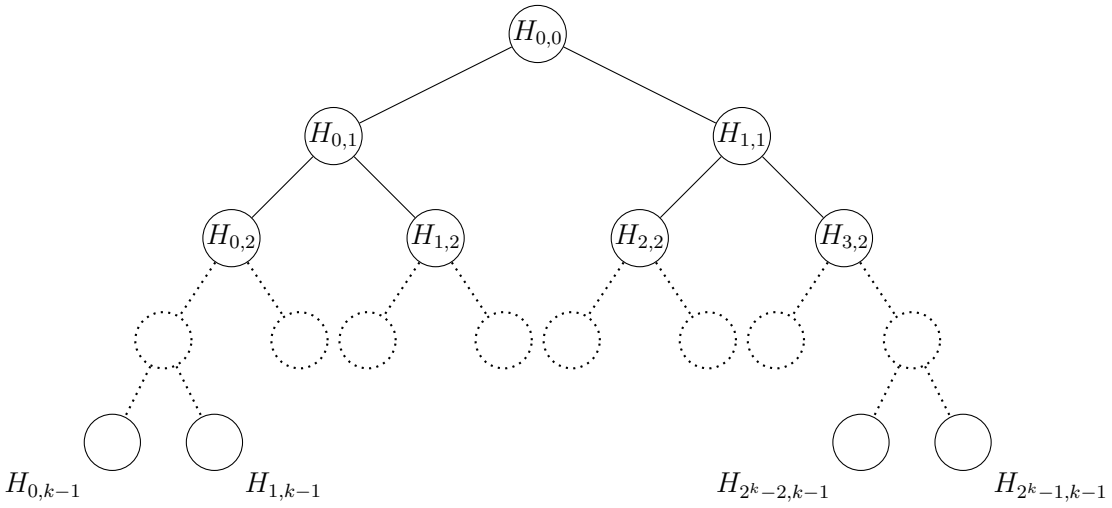


Abbildung 10: Aufbau eines Merkle Baumes der Höhe $H = k$

annähernd unendlich, allerdings wächst dadurch auch die Länge der Signatur [10]. Eine längere Signatur bedeutet dabei immer, dass sowohl das Erstellen des Schlüsselpaares, als auch das Signieren und Verifizieren deutlich mehr Zeit benötigt. Viele Verfahren arbeiten jedoch noch auf der Basis der RSA-Verschlüsselung, die, wie wir bereits in Abschnitt 4.1 gesehen haben, von Shors Algorithmus in Polynomialzeit auf einem Quantencomputer gelöst werden kann. Diese Signaturen können somit nicht mehr als sicher angesehen werden, da Angreifer in der Lage sind ihre eigenen Dokumente zu signieren, sodass sie vom öffentlichen Schlüssel und den dazu gehörigen Verifikationsalgorithmus als gültig erkannt werden. Es wird daher nach Verfahren gesucht, die ähnlich effizient sind wie das RSA-Verfahren, aber gegenüber Quantencomputern als sicher angesehen werden. Ein viel versprechendes Verfahren ist das Merkle Signaturverfahren basierend auf Merkle Bäumen.

5.5. Merkle Bäume

Merkles Signaturverfahren ist ein besonderes Verfahren zum Erstellen und Verifizieren von digitalen Signaturen, hierbei wird ein sogenannter Merkle Tree, ein spezieller Binärbaum, eingesetzt. Dieses Verfahren wurde von Ralph C. Merkle entwickelt und veröffentlicht [30]. Es basiert auf der Grundlage von Einmalsignaturverfahren, durch die Erweiterung um den Merkle Tree können solche Verfahren jedoch mehrfach eingesetzt werden. Es muss vorher allerdings festgelegt werden wie viele Signaturen überprüft werden können [9]. Merkle selbst hat in seiner Veröffentlichung [30] die Sicherheit seines

Verfahrens nicht eindeutig bewiesen. Andere Gruppen [18] haben sich damit jedoch ausführlich auseinander gesetzt und neben dem Beweis für die Sicherheit auch das Problem mit der vorab zu wählenden Menge an Signaturen gelöst. So sind 2^N Signaturen möglich und die Zeit für die Verifikation ist für $N = 40$ in einem angemessenen Zeitrahmen, daher ist dieses Verfahren eine pragmatische Alternative für RSA-Signaturen [18]. Der einmalige Aufwand zum Erstellen eines solchen Schlüssels liegt zwar noch bei knapp vier Stunden, ist jedoch zum Standardverfahren, welches 235 Jahre benötigen würde, um einiges verbessert worden. Es muss eine gesunde Balance zwischen dem Zeit- und Speicherbedarf dieser Signaturmethode gefunden werden.

Viele Algorithmen zum Erstellen und Speichern des Binären Baumes sowie zum Verifizieren der Signaturen sind bereits durch verschiedene Gruppen analysiert und verbessert worden [9, 13, 18].

Schlüssel generieren

Die Hauptaufgabe der Berechnung der Schlüssel ist die Berechnung und das Veröffentlichen des Wurzel (*root*) Knotens. Hierbei muss die Person, die eine Signatur erstellen möchte, 2^H Schlüsselpaare $(X_j, Y_j), 0 \leq j \leq 2^H$ für die Einmalsignaturen erstellen. Wobei X_j der Schlüssel zum Signieren und Y_j der Schlüssel zum Verifizieren ist. Die Blätter des binären Merkle Baumes sind die Ziffern $g(Y_j), 0 \leq j \leq 2^H$. Die inneren Knoten werden aus ihren Nachfolgern berechnet. Ein Eltern-Knoten ist immer die Verkettung seiner beiden Kinder-Knoten. Der Wurzel-Knoten ist der öffentliche Schlüssel des Signaturverfahrens. Folglich benötigt dieser ganze Prozess die Berechnung von 2^H Einmalsignaturschlüsselpaaren und $2^{H+1} - 1$ Auswertungen der Hashfunktion. Das ursprüngliche Verfahren speichert den ganzen Baum einfach als einen Stack ab. Um den Wert von Knoten zu aktualisieren, müssen dann häufig ganze Stacks durchgegangen werden. Da dies extrem die Effizienz des Verfahrens beeinflusst gibt es verschiedene andere Ansätze dieses Problem zu umgehen. Der Binärbaum wird in kleinere Teilbäume zerlegt, hierbei sind die Wurzeln eines Unterbaumes immer Blätter eines übergeordneten Baumes.

Zum Verifizieren wird nun eine kleinere Menge an Knoten benötigen, da der Pfad zur Wurzel nur über einen Teil der Unterbäume geht und daher nicht mehr jeder Knoten einzeln gespeichert werden muss. Teilbäume werden nur solange gespeichert wie sie auch in Verwendung sind, so kann es sein, dass beim Generieren der Signatur alte Bäume verworfen, neue erstellt oder sogar bereits verworfene neu berechnet werden. Dies scheint auf den ersten Blick nicht sonderlich effizient, beschleunigt aber die einzelnen Laufzeiten

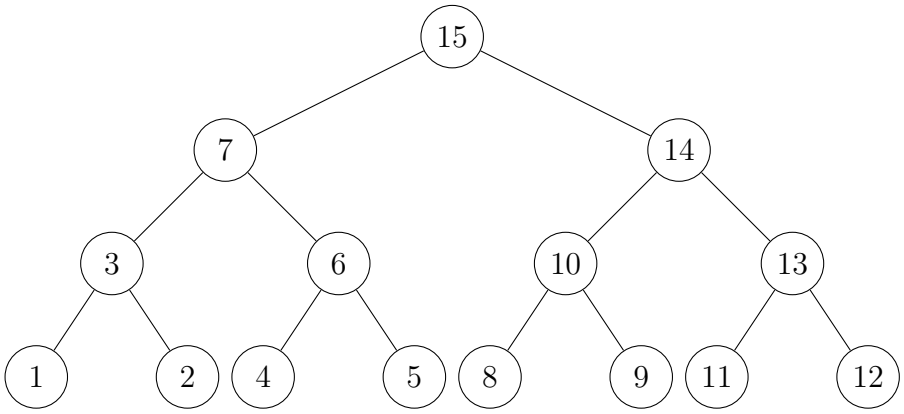


Abbildung 11: Reihenfolge der Knotenberechnung

der Signaturerstellung, da der zu durchlaufende Baum kleiner ist. Außerdem sinkt der Speicherbedarf immens bei diesem Vorgehen. Je größer die Subbäume, desto schneller läuft der Algorithmus und desto mehr Platz benötigt er auch.

$$\mathit{Time}_{max} = 2(L-1) < 2\frac{H}{h}$$
$$\mathit{Space}_{max} < 2 \cdot L \cdot 2^{h+1} + H \cdot \frac{L}{2}$$

Wobei L die Anzahl der Subbäume, H die Gesamthöhe des Baumes und h die Höhe der Subbäume ist.

Verifikation

Die Merkle Signatur beinhaltet die Einmalsignatur σ_{OTS} und der dazu gehörige einmalige Verifikationsschlüssel (Y_s). Um die Integrität des Schlüssel Y_s zu bestätigen, wird auch noch der Index s und der Verfikationsspfad für den Schlüssel Y_s angehängt. Der Verifikationsspfad ist eine Sequenz von Knoten in dem binären Merkle Baum $A_s = (a_0, \ldots a_{H-1})$, sodass sich folgende Struktur für die Signatur σ_s ergibt:

$$\sigma_s = (s, \sigma_{OTS}, Y_s, (a_0, \ldots, a_{H-1})).$$

Durch diesen Pfad und den Index kann von einer Person, die die Signatur verifizieren möchte, ein Pfad vom Blatt des Merkle Baumes $g(Y_s)$ bis hin zur Wurzel des Baumes erstellt werden (vgl. Abbildung 12). Die Wurzel (ganz oben im Baum) fungiert als öffentlicher Schlüssel, um die Echtheit der Einmalsignatur zu verifizieren. Dieses Verfahren ist stark abhängig davon in welcher Form der Merkle Baum gespeichert wurde.

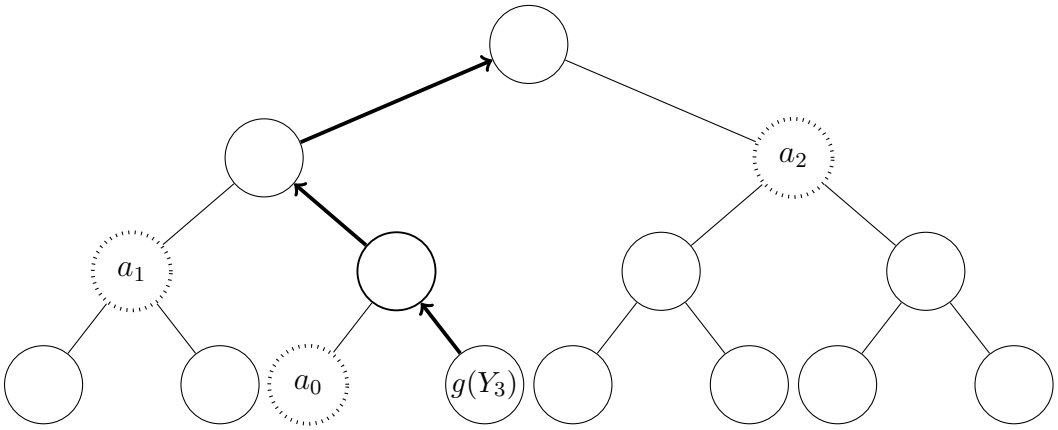


Abbildung 12: Konstruktion des Verifikationspfades

Die Speicherung der Verifikationspfade kann deutlich effizienter gestaltet werden als alle einzeln abzulegen, da viele Pfade sich an einigen Stellen überlappen. So unterscheidet sich der gezeigte Pfad von $g(Y_3)$ aus Abbildung 12 nur in a_0 von dem Pfad des Knotens links davon, also $g(Y_2)$. Die Speicherung kann dann beispielsweise so erfolgen, dass statt eines komplett neuen Pfades, nur die Änderungen zum Pfad davor gespeichert werden, dies hat meist nur wenige Änderungen. So wird auch dieses Verfahren immens beschleunigt, wenn nicht der gesamte Baum, sondern nur die notwendigen Unterbäume durchlaufen werden müssen [9]. Es ist möglich, dass Unterbäume nach einiger Zeit nicht mehr benutzt werden somit verfallen und keinen Speicherplatz mehr besetzen. Diese Subbäume werden mit Hilfe von Markierungen berechnet, sodass die Berechnungen der folgenden Pfade durch diese Subbäume effizienter werden können [9].

5.6. Sicherheit gegenüber Quantencomputern

Die Sicherheit von Hashfunktionen gegenüber Quantencomputern wird maßgeblich von Grovers Algorithmus (vgl. Abschnitt 4.2) in Frage gestellt. Dieser Algorithmus weist eine Beschleunigung bei der Suche von bestimmten Daten in großen Datenmengen auf. So kann auch die Suche nach den Kollisionen einer Hashfunktion durch die Beschleunigung des Grovers Algorithmus profitieren. Bereits die Wissenschaftler Brassard, Høyer und Tapp [7] haben gezeigt, dass es möglich ist dieses Verfahren noch zu beschleunigen.

In Abschnitt 5.1 haben wir bereits gefordert, dass Hashfunktionen, die für Signaturen eingesetzt werden, kollisionsresistent ist, da alle Verfahren nur so sicher sein können wie die den Verfahren zugrundegelegeten Hashfunktionen [9]. Allerdings sind Kollisionen in Hashfunktionen nicht vollständig auszuschließen, daher müssen die Funktionen so

aufgebaut sein, dass die Auswertung einer Hashfunktion eine gewisse Zeit in Anspruch nimmt. Diese Zeit sollte lange genug sein, sodass Vorberechnungen von sehr vielen Werten nicht mehr praktikabel sind, aber dennoch das einzelne Auswerten zum Signieren oder Verifizieren noch in einer angemessen Zeit geschehen kann.

Je länger eine Signatur ist, desto größer ist auch der Suchraum für mögliche Kollisionen. Da Grovers Algorithmus, selbst die verbesserte Version, nur über eine begrenzte Beschleunigung verfügt, muss die Länge der Signatur soweit angepasst werden, dass diese Beschleunigung wieder negiert wird. Meist reicht dafür eine Verdopplung der Signaturlänge aus. Selbst eine Forschungsgruppe des NIST (*National Institute of Standards and Technology*) hat bestätigt, dass die klassischen Hashfunktionen SHA-2 und SHA-3 zur Zeit nicht als Post-Quantum sicher eingestuft werden können, jedoch durch längere Outputs dies zu erreichen wäre [10].

Das Sicherheitslevel b eines Verschlüsslungsverfahrens gibt an wie viele Rechenoperationen ein Angreifer mindestens ausführen muss, um dieses Verfahren zu brechen. So bedeutet ein Sicherheitslevel von 33, dass 2^{33} Rechenschritte benötigt werden.

Satz 27. Das Sicherheitslevel b der Merkle Signatur kombiniert mit der Lamport-Diffie Einmalsignatur ist gegenüber klassischen Computern mindestens

$$b = \frac{n}{2} - 1,$$

wenn die Höhe des Merkle Baumes höchstens $H \leq \frac{n}{3}$ und die Länge des Outputs der Hashfunktion mindestens $n \geq 87$ ist [9].

Satz 28. Das Sicherheitslevel b der Merkle Signatur kombiniert mit der Lamport-Diffie Einmalsignatur ist gegenüber Quantencomputern mindestens

$$b = \frac{n}{3} - 1,$$

wenn die Höhe des Merkle Baumes höchstens $H \leq \frac{n}{4}$ und die Länge des Outputs der Hashfunktion mindestens $n \geq 196$ ist [9].

Diese beiden Sätze stammen aus dem Werk *Post Quantum Kryptographie* von Buchmann, Dahmen und Szydlo [9] und werden auch dort bewiesen. Daher können wir davon ausgehen, dass das Merkle Verfahren in Kombination mit dem Lamport-Diffie Verfahren auch nach dem erfolgreichen Einsatz von Quantencomputern sicher bleibt. Dies ist gegeben durch die Struktur des Baumes und die Art und Weise wie sich die Knoten berechnen. Die uns bereits bekannten Quantenalgorithmen (vgl. Abschnitt 4) bieten nur minimale

Länge des Outputs n	128	160	224	256	384	512
Klassischer Fall						
bit security b	63	79	111	127	191	255
Maximaler Wert für H	42	53	74	85	128	170
Quanten Fall						
bit security b	-	-	73	84	127	169
Maximaler Wert für H	-	-	56	64	96	128

Tabelle 13: Sicherheitslevel der Merkle Signatur kombiniert mit dem Lamport-Diffie Einmalsignaturverfahren in Bits

Vorteile gegenüber klassischen Algorithmen zur Berechnung des Merkle Baumes. Die maximale Höhe H des Merkle Baumes wird maßgeblich durch die Länge des Outputs der Hashfunktion bestimmt, vgl. Tabelle 13. Anhand dieser Tabelle sieht man auch eindeutig, dass das Merkle Verfahren selbst mit heutigen Signaturen und Hashfunktionen gegenüber dem Quantencomputer als sicher angesehen werden können.

Die Verbesserung des Merkles Signaturverfahrens und der Beweis, dass es sicher gegen Chosen-Message-Angriffe ist, wurde gezeigt von García [18]. Bei diesem Beweis wird der Kollisionsalgorithmus von Brassard, Høyer und Tapp [7] als optimale Lösung zum Finden von Kollisionen angenommen.

Das Merkle Signaturverfahren ist demnach in der Lage ausreichend viele verschiedene Signaturen zu erstellen und Dokumente auch in kurzer Zeit zu signieren und verifizieren. Einzig das einmalige Erstellen des öffentlichen Schlüssel, des Wurzelknotes des Merkle Baumes, stellt noch eine zeitliche Belastung da.

Das Merkle Signaturverfahren hält nach dem aktuellen Kenntnisstand einem Angriff durch Quantencomputer stand. Das Verfahren selber benötigt noch einige Optimierungen im Hinblick auf den Zeitbedarf der Signaturerstellung, aber ansonsten wäre es bereits heute praktisch einsetzbar [9].

6. Ausblick und Zusammenfassung

Stand der Technik

Quantencomputer beschäftigen die Wissenschaft schon eine ganze Weile, überwiegend handelt es sich dabei sowohl um die Theorie wie ein Quantencomputer am Besten rechnen sollte als auch das Erstellen und Bauen von Quantencomputern. Einige namenhafte Unternehmen sind maßgeblich an der Forschung und Entwicklung von Quantencomputern beteiligt. Auch wenn zur Zeit Quantencomputer mit sehr wenigen Quantenbits existieren so gibt es einen 50 Bit Quantencomputer der kurzzeitig seinen Zustand halten kann. Ein 20 Bit Quantencomputer ist sogar per Webportal verfügbar, um einige kleine Test damit durchzuführen [16]. Die Dekohärenz (vlg. Abschnitt 3.2) ist zur Zeit noch problematisch. Die Wechselwirkungen zwischen den Quantenbits stellen zur Zeit noch ein großes Problem dar, die einige Forscher noch eine Weile beschäftigen wird. Auch wenn das Grundkonzept Anwendung findet, so bildet auf Grund der eben beschriebenen Dekohärenz die Skalierung von Quantencomputer noch eine relevante Problemstellung. Dennoch wurde ein Quantencomputer vorgestellt, der wohl eine Beschleunigung von 10^8 bei der Lösung von Optimierungsproblemen aufweist [11].

Auswirkungen auf die Kryptographie

In Abschnitt 4 sind wir bereits auf die hochpotenten Algorithmen eingegangen, die nun durch den Quantencomputer tatsächlich ihre Anwendung finden können. Diese Algorithmen bilden den Anreiz Kryptographiesysteme zu entwickeln und zu evaluieren, die eben diesen neuen Algorithmen stand halten können. Genau zu diesem sehr neuen Forschungsgebiet bilden sich in der jüngeren Vergangenheit nun neue Konferenzen und Gemeinschaften, die sich aktiv darüber austauschen.

Da es immer noch möglich ist, dass neue Quantenalgorithmen, oder sogar klassische Algorithmen entdeckt werden, die Probleme lösen, die nach heutiger Ansicht als *schwierig* gelten, sollten Kryptographieverfahren nach Möglichkeit mehr als ein solch schwieriges

Problem als Grundlage nutzen. So kann Shors Algorithmus (vgl. Abschnitt 4.1) auf einen schon sehr lange bekannten Teil und ein Quantenorakel aufgeteilt werden. Dieses Quantenorakel gibt uns eine Lösung zu einem Problem. In dem Fall von Shors Algorithmus ist nun nicht länger ein Quantenorakel notwendig, da nun klar ist wie die Periode von einem Quantencomputer errechnet werden kann. Es ist folglich möglich Probleme, die zur Zeit nur mit Hilfe von Orakeln zu lösen sind, in Zukunft (eventuell sogar sehr simpel) zu lösen und dadurch Verschlüsselungsverfahren, die auf diesen Problemen basieren, obsolet zu machen. Ein gutes Verschlüsselungsverfahren zeichnet sich also dadurch aus, dass mehrere Orakel einer Turingmaschine (vgl. Abschnitt B) angewendet werden müssen, um die Verschlüsselung rückzuberechnen.

Klar ist allerdings bereits jetzt schon, dass die RSA-Verschlüsselung und alle anderen Systeme, die auf dem Faktorisierungsproblem oder dem Lösen von Diskreten Logarithmen beruhen, nicht länger als sicher angesehen werden können. Viele unserer heutzutage eingesetzten Verfahren nutzen aktiv das RSA Public-Private-Key oder RSA-Signatur Verfahren, so zum Beispiel die Übermittlung von Informationen bei der Zahlung mit einer EC-Karte. Dieser Vorgang muss mit der Existenz und dem praktischen Einsatz von Quantencomputern zweifelsfrei ausgetauscht werden. Auch andere Verfahren die sich in unserem täglichen Leben etabliert haben, werden sich verändern müssen.

Quantenkryptographie

Anstatt sich immer nur damit auseinanderzusetzen wie sehr die Quantencomputer die moderne Kryptographie beeinflussen und einige sehr gängige Verfahren obsolet machen, betrachten wir nun kurz wie wir Quantencomputer einsetzen können, um mit ihrer Hilfe neue Kryptographiesysteme zu bauen.

Ein bereits bekanntes und nachweislich zu 100% sicheres Verfahren ist das *One-Time-Pad* [22]. Es handelt sich hierbei um einen symmetrischen Schlüssel, der nur aus Einsen und Nullen besteht. Das Verschlüsseln und Entschlüsseln geschieht hierbei durch das Anwenden der XOR-Operation. Hierbei ist es extrem wichtig, dass der Schlüssel bei jeder Nachricht getauscht wird, da es bereits bei der zweiten Verwendung möglich ist, den Schlüssel rückzuberechnen und die Nachricht von Dritten entschlüsselt werden kann. Der Schlüssel muss im klassischen Fall also beiden Parteien bekannt sein.

Der Quantencomputer eröffnet nun neue Möglichkeiten des Schlüsselaustausches. Der Schlüssel kann mit Hilfe von Quantenbits über einen unsicheren Kanal ausgetauscht werden. Um das Verständnis zu erleichtern betrachten wir wieder einmal das klassische Beispiel von Alice, die eine Nachricht an Bob schicken möchte, sowie Eve, die diese

Nachricht abfangen möchte. So müssen zuerst eine Anzahl an Quantenbits miteinander verschränkt werden, diese eine Hälfte der verschränkten Bits behält Alice, die andere schickt sie an Bob. Durch die Verschränkung und die Superposition der Quantenbits werden so immer neue zufällige Bitfolgen generiert. Sobald Bob alle Bits erhalten hat, teilt ihm Alice die Basis mit anhand der sie gemessen hat. Alice und Bob messen nun alle ihre Bits und werden auf Grund der Verschränkung dieser Bits exakt die gleiche Bitfolge messen.

Alice und Bob vergleichen nun weiterhin über den unsicheren Kanal ein paar Bits, sollten diese alle identisch sein können sie mit sehr hoher Sicherheit davon ausgehen, dass niemand ihren Schlüsselaustausch belauscht hat. Sollten allerdings die Messergebnisse von einander abweichen, so wissen Alice und Bob, dass etwas nicht stimmt. So kann Eve die Quantenbits unterwegs abgefangen und gemessen haben, da ihre Messung der Bits deren Superposition verändert hätte. Da Alice die Basis, anhand welcher sie die Quantenbits gemessen hat, Bob erst mitteilt, nachdem dieser den Eingang aller Quantenbits bestätigt hat, musste Eve die Basis zum Messen erraten und die Wahrscheinlichkeit, dass sie dabei richtig liegt, ist minimal.

Um nun herauszufinden, ob Eve versucht hat mitzuhören, können Alice und Bob einige Messergebnisse miteinander vergleichen. Je mehr Bits sie miteinander vergleichen, desto höher ist die Wahrscheinlichkeit, dass niemand versucht hat den Schlüsselaustausch zu belauschen. Sollte jedoch auch nur ein Bit nicht identisch sein, so muss davon ausgegangen werden, dass die Bits gemessen beziehungsweise manipuliert wurden. Ist der Schlüssel erfolgreich ausgetauscht worden, so kann die Nachricht verschickt werden und ist für Dritte auf Grund der Sicherheit des Verfahrens unmöglich zu berechnen [22].

Der Stand der Technik ist allerdings noch nicht so weit, dass Quantenbits verschickt werden können und dabei sicherstellen zu können, dass die Superposition der Bits unverändert bleibt. Eben das ist aber unbedingt notwendig für den Erfolg dieses Verfahrens, daher ist auch dieses Verfahren noch nicht praktisch umsetzbar.

Zusammenfassung

Quantencomputer sind durch die Nutzung der Quantenbits an unitäre Transformationen gebunden. Unitäre Gatter können alle unsere bekannten Gatter darstellen und benötigen dabei polynomiell mehr Gatter, sind daher also weiterhin *effizient*. Die Nutzung der Quantenbits bringt einige Vorteile mit sich, so können Quantenbits in einer Superposition übergehen, in der sie sogar miteinander verschränkt werden können. Hinzu kommt der Quantenparallelismus, welcher auf Grund der Superposition der Quantenbits die Berech-

nungen auf allen möglichen Zuständen gleichzeitig ausführen kann. Diese beiden Effekte sorgen für eine sehr schnelle Berechnung, wenn sie mit geschicktem Messen kombiniert werden. Shors Algorithmus macht sich das Prinzip der Verschränkung von Quantenbits zu Nutzen, um die Periode von Funktionen effizient zu berechnen. Durch diesen neuen Algorithmus ist es möglich zu zeigen, dass das Problem der Faktorisierung nun nicht länger nur in **NP**, sondern auch in **BQP** liegt (vgl. Abschnitt 4.1). Durch diesen Fortschritt ist gezeigt, dass die wohl gängigste Verschlüsselung, RSA [34], nicht mehr als sicher angesehen werden kann.

Daher haben wir uns im Bereich der Hash-basierten Verfahren auf die Suche nach Alternativen begeben. Hierbei lag der Hauptfokus auf fälschungssicheren digitalen Signaturen. Um die Sicherheit von solcher Verfahren zu garantieren, werden einige Anforderungen an die unterliegende Hashfunktion gestellt, unter anderem müssen Hashfunktionen Einwegfunktion sein, die auch noch kollisionsresistent sind. Daher setzen wir auf Einmalsignaturverfahren. Diese gelten als besonders sicher, aber leider auch nicht sonderlich effizient, da sie wie der Name schon sagt, nur einmal eingesetzt werden können. Um diese Verfahren dennoch mehrfach einzusetzen, wird das Verfahren der Merkle Bäume eingesetzt. Diese stellen durch ihren Aufbau eine sehr hohe Menge an Einmalsignaturen mit dazugehörigen Schlüsselpaaren zur Verfügung. Jetzt muss nur noch die Authentizität des Einmalschlüssels überprüft werden, genau hier kommt der Merkle Baum ins Spiel.

Das Verfahren von Merkle in Kombination mit der Lamport-Diffie Einmalsignatur ist nachweislich sicher gegenüber Quantencomputern (vgl. Abschnitt 5.6), sodass hier eine Alternative zu der aktuell sehr häufig eingesetzten RSA-Signatur besteht [9].

Bis zu dem Zeitpunkt an dem ein Quantencomputer gebaut wird, der mehrere Tausende Quantenbits hat, ist es jedoch nicht möglich die RSA-Verschlüsselung mit zur Zeit eingesetzten Schlüssellängen rückzuberechnen. Auch wenn nicht eindeutig klar ist, ob es überhaupt möglich wird Quantencomputer von dieser Größe zu bauen, so wird dadurch ein dunkler Schatten auf die aktuellen Kryptographiesysteme geworfen. Wenn man möchte, dass sensible Informationen, die bereits heute verschickt werden weiterhin vertraulich bleiben, so sollte man bereits heute auf Post-Quantum-Kryptographie setzen.

Anhang

A. Mathematische Grundlagen

Um mit Quantenalgorithmen und Quantenbits umgehen zu können, sind einige Grundlagen aus dem Bereich der Algebra von Nöten. Wir möchten hier jedoch keine komplette Einführung geben, sondern nur die Teile betrachten, die für uns im Rahmen dieser Bachelorarbeit notwendig sind.

Vektorräume

Wir betrachten in dem Zusammenhang mit Quantenalgorithmen nur komplexe Vektorräume, die aus einzelnen komplexen Vektoren bestehen:

$$v = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{n-1} \end{pmatrix}.$$

Komplexe Vektoren bestehen aus n untereinander angeordneten komplexen Zahlen, auch n-Tupel genannt. Jeder Spaltenvektor kann transponiert werden, sodass sich ein Zeilenvektor $v^T = (\alpha_0, \ldots, a_{n-1})$ ergibt.

Wollen wir nun von einzelnen Vektoren zum Raum $\mathbb{C}^n$ übergehen, so können wir das

kartesische Produkt verwenden:

$$\begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{n-1} \end{pmatrix} + \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_{n-1} \end{pmatrix} = \begin{pmatrix} \alpha_0 + \beta_0 \\ \vdots \\ \alpha_{n-1} + \beta_{n-1} \end{pmatrix}.$$

In einem solchen $\mathbb{C}^n$ Raum kann auch multipliziert werden.

$$a \cdot \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{n-1} \end{pmatrix} = \begin{pmatrix} a \cdot \alpha_0 \\ \vdots \\ a \cdot \alpha_{n-1} \end{pmatrix}$$

Hierdurch werden nun alle Vektorraumaxiome erfüllt. Diese Axiome besagen nämlich nur, dass man lediglich Vektoren addieren kann und Skalarmultiplikation möglich sein muss. Somit haben wir soeben komplexe Vektorräume vollständig definiert.

Definition 29. Sei $M = \{|0\rangle, |1\rangle, \ldots, |n-1\rangle\}$ eine Menge mit n Objekten. Eine Summe der Form

$$\alpha_0 \cdot |0\rangle + \alpha_1 \cdot |1\rangle + \cdots + \alpha_{n-1}|n-1\rangle$$

mit α_i aus $\mathbb{C}$. Als lineare Hülle einer Menge M über $\mathbb{C}$ bezeichnet man die Menge aller solchen Linearkombinationen [22]. So ist der dazugehörige *Vektorraum*:

$$\mathrm{Span}_{\mathbb{C}} M = \{\alpha_0 \cdot |0\rangle + \alpha_1 \cdot |1\rangle + \cdots + \alpha n - 1|n-1\rangle; \alpha_0, \ldots, \alpha_{n-1} \in \mathbb{C}\}\ [22].$$

Hierbei können die Faktoren $\alpha_0, \ldots, \alpha_{n-1}$ als n-Tupel aufgefasst und somit als Vektor dargestellt werden.

Definition 30. Eine Menge S von Vektoren ist genau dann *linear abhängig*, wenn gilt: Ein Vektor x aus S kann von anderen Vektoren aus S erzeugt werden:

$$x \in \mathrm{Span}(S \setminus x).$$

In einer linear abhängigen Menge sind also Vektoren enthalten, die durch andere erzeugt werden können. Diese sind in der Linearkombination somit nicht nötig. Linear unabhängige Mengen sind minimal, sollte man also ein Teil der Menge entfernen, können auch nur noch weniger Vektoren erzeugt werden.

Definition 31. Vektoren $v_1, \ldots, v_m$ eines Vektorraumes V heißen *Basis* von V, wenn sie linear unabhängig sind und ihre lineare Hülle ganz V ergibt:

$$\mathrm{Span}\{v_1, \ldots, v_m\} = V.$$

Alle möglichen Basen eines Vektorraumes haben die gleiche Größe, diese nennt man daher auch Dimension des Vektorraumes.

Definition 32. Eine Basis, deren Vektoren orthogonal zueinander sind, nennt man *Orthogonalbasis*. Sollten alle Vektoren die Länge $|1|$ haben, so ist es eine *Orthonormalbasis*.

Definition 33. Eine Menge U von Vektoren aus V heißt *Unterraum* von V, wenn U alle Linearkombinationen von Vektoren aus U enthält, wenn also gilt:

$$\mathrm{Span}U = U$$

Unterräume können erzeugt werden, indem z. B. einzelne Elemente der Basis von V ausgewählt werden und ihre lineare Hülle betrachtet wird.

Mit dem Skalarprodukt lässt sich nun die Projektion eines Vektors auf einen Unterraum definieren. Zum leichteren Verständnis beginnen wir mit der Projektion auf die Koordinatenachsen.

Beispiel 34. Der Vektor

$$\begin{pmatrix} a \\ b \end{pmatrix}$$

entspricht einer komplexen Zahl z. Die *Projektion* auf die x-Achse ist dann nur der Vektor

$$\begin{pmatrix} a \\ 0 \end{pmatrix},$$

analog dazu ist die *Projektion* auf die y-Achse [22]

$$\begin{pmatrix} 0 \\ b \end{pmatrix}.$$

Definition 35. Sei $b_1, \ldots, b_m$ die Basis eine Orthonormalbasis eines Unterraumes E. Die *Projektion* eines Vektors v auf diesen Unterraum ist der Vektor:

$$b_1 \cdot \langle b_1|v\rangle + b_2 \cdot \langle b_2|v\rangle \cdot b_2 + \cdots + b_m \cdot \langle b_m|v\rangle \ [22].$$

Matrizen

Um die Transformationen in Abschnitt 3.2 nachvollziehen zu können, werden nun einige Definitionen zu Matrizen betrachtet.

Definition 36. Eine Matrix, deren Einträge auf der Hauptdiagonalen 1 sind und ansonsten 0, nennt man *Einheitsmatrix* I^n. Eine 2×2 *Einheitsmatrix* wird demnach als I^2 bezeichnet.

$$I^n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

Definition 37. Eine Matrix, die mit der Matrix A multipliziert wird, sodass die Einheitsmatrix entsteht, nennt man *Inverse* der Matrix A also A^{-1}.

$$A^{-1}A = AA^{-1} = I.$$

Definition 38. Eine Matrix $A = (a_{ij})$ mit komplexen Einträgen kann zur *komplex konjugierten* Matrix $\bar{A} = A^*$ überführt werden, indem die Einträge a_{ij} durch $\bar{a}_{ij}$ ersetzt werden [22]. Sodass gilt:

$$A = (a_{ij}) \in \mathbb{C}^{n \times m} \qquad A^* = \bar{A} = (\bar{a}_{ij}) \in \mathbb{C}^{n \times m}$$

$$A = \begin{pmatrix} a_{00} & \cdots & a_{0(n-1)} \\ \vdots & \ddots & \vdots \\ a_{(m-1)0} & \cdots & a_{(m-1)(n-1)} \end{pmatrix} \quad A^* = \begin{pmatrix} \bar{a}_{00} & \cdots & \bar{a}_{0(n-1)} \\ \vdots & \ddots & \vdots \\ \bar{a}_{(m-1)0} & \cdots & \bar{a}_{(m-1)(n-1)} \end{pmatrix}$$

Definition 39. Eine Matrix $A = (a_{ij})$ kann zur *transponierten* Matrix A^T überführt werden, indem die Einträge a_{ij} mit a_{ji} getauscht werden [22].

$$A = (a_{ij}) \in C^{n \times m} = \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0(n-1)} \\ a_{10} & a_{11} & \cdots & a_{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(m-1)0} & a_{(m-1)1} & \cdots & a_{(m-1)(n-1)} \end{pmatrix}$$

$$A^T = (a_{ji}) \in C^{m \times n} = \begin{pmatrix} a_{00} & a_{10} & \cdots & a_{(m-1)0} \\ a_{01} & a_{11} & \cdots & a_{(m-1)1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{0(n-1)} & a_{1(n-1)} & \cdots & a_{(n-1)(m-1)} \end{pmatrix}$$

Setzt man nun die beiden Definitionen 38 und 39 zusammen erhält man die Definition zu adjungierten Matrizen.

Definition 40. Eine Matrix $A = (a_{ij})$ kann zur *adjungierten* Matrix

$$\bar{A}^T = (A^*)^T = A^\dagger$$

überführt werden, indem die Matrix zuerst komplex konjugiert und anschließend transponiert wird, also die Einträge a_{ij} mit $\bar{a}_{ji}$ getauscht werden [22].

Definition 41. Sei A eine $n \times n$-Matrix, deren Einträge komplexe Zahlen sind. A heißt *unitär*, falls

$$A^\dagger = A^{-1}.$$

Eine unitäre Matrix beschreibt immer eine unitäre Transformation [22].

Quadratische Matrizen (deren Einträge komplexe Zahlen sind) entsprechen Abbildungen f von n-dimensionalen komplexen Vektorräumen in sich selbst.

Definition 42. Ein Vektor v wird mittels Matrixmultiplikation auf sein Bild $w = Av$ abgebildet:

$$f \colon \mathbb{C}^n \to \mathbb{C}^n, v \mapsto Av.$$

Tensorprodukt

Das Tensorprodukt wird insbesondere benötigt, wenn man mit Quantenregistern arbeitet (vgl. Abschnitt 3.4). Das Tensorprodukt kann auch eingesetzt werden um Matrizen als Transformation einzusetzen.

Definition 43. Seien V_1 und V_2 zwei Vektorräume. Sei $e_0, \ldots, e_{m-1}$ eine Basis von V_1 und $f_0, \cdot, fn-1$ eine Basis von V_2. Das Tensorprodukt $V_1 \otimes V_2$ dieser Räume ist ein $(m \cdot n)$-dimensionaler Vektrorraum. Seine Bassisvektoren bezeichnen wir mit

$$\begin{array}{cccc} e_0 \otimes f_0, & e_0, \otimes f_2, & \cdots & e_0 \otimes f_{n-1}, \\ e_2 \otimes f_0, & e_2, \otimes f_2, & \cdots & e_2 \otimes f_{n-1}, \\ \vdots & \vdots & \ddots & \vdots \\ e_{m-1} \otimes f_0, & e_{m-1} \otimes f_2, & \cdots & e_{m-1} \otimes f_{n-1}. \end{array}$$

Beispiel 44. Die Vektoren

$$v_1 = \alpha_0 e_0 + \cdots + \alpha_{m-1} e_{m-1} = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{m-1} \end{pmatrix}$$

$$v_2 = \beta_0 f_0 + \cdots + \beta_{m-1} f_{m-1} = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_{m-1} \end{pmatrix}$$

seien Vektoren aus den V_1 und V_2, so ist ihr Tensorprodukt wie folgt:

$$v_1 \otimes v_2 = (\sum_{i=0}^{m-1} \alpha_i e_i) \otimes (\sum_{j=0}^{n-1} \beta_j f_j) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \alpha_i \beta_j (e_i \otimes f_j).$$

Ein Tensorprodukt nutzt die Vektoren der Ausgangsräume, um damit einen neuen Raum zu erschaffen. Je mehr Ausgangsräume mit einander verrechnet werden, desto stärker wächst das Tensorprodukt. Die folgenden Beispiele veranschaulichen das starke Wachstum des Tensorproduktes.

Beispiel 45. So gilt, dass das Tensorprodukt der Einheitsmatrix I_2 mit sich selbst berechnet wird durch:

$$I_2 \otimes I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes I_2 = \begin{pmatrix} I_2 & 0 \\ 0 & I_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = I_4$$

Beispiel 46. Um dies noch zu verdeutlichen, betrachte man die Verrechnung von drei 2×2 Einheitsmatrizen.

$$I_2 \otimes I_2 \otimes I_2 = \begin{pmatrix} I_4 & 0 \\ 0 & I_4 \end{pmatrix} = I_8 = \begin{pmatrix} I_2 & 0 & 0 & 0 \\ 0 & I_2 & 0 & 0 \\ 0 & 0 & I_2 & 0 \\ 0 & 0 & 0 & I_2 \end{pmatrix}$$

B. Turingmaschine

Einer der zentralen Ansätze der theoretischen Informatik beschäftigt sich mit der Turingmaschine. Eine Turingmaschine ist ein sehr simples Modell eines Computers. Dieses Modell bricht auch komplizierte Algorithmen auf ihre Grundbestandteile herunter. Eine Turingmaschine wird genutzt um Entscheidungen zu treffen, hierbei geht es meistens darum, ob ein eingegebenes Element x Teil einer Sprache A ist oder nicht ($x \in A$?). Diese Sprachen können verschieden definiert sein. Wir interessieren uns allerdings eher dafür, ob eine Turingmaschine ein Problem in einer gewissen Zeit lösen kann. Sollte dies nicht möglich sein, dann versuchen wir herauszufinden, ob eine mögliche Lösung zu einem Problem von einer Turingmaschine verifiziert oder abgelehnt werden kann.

Es wird unterschieden zwischen deterministischen und nichtdeterministischen Turingmaschinen, diese werden auch im folgenden formal definiert (Definition 47). Eine Turingmaschine unterliegt bestimmten Regeln, daher können wir mit einer Turingmaschine immer feste Aussagen über die Laufzeit und den Speicherbedarf eines Algorithmus treffen, somit lässt sich auch die Effizienz eines Algorithmus über diese beiden Parameter bestimmen. Wir betrachten dabei insbesondere wie viele Rechenschritte abhängig von der Eingabegröße notwendig sind, um zu einem Ergebnis zu gelangen. Es gibt auch Einschränkungen bezüglich des Platzbedarfes einer Turingmaschine, diese sind für uns innerhalb dieser Bachelorarbeit aber von geringerem Interesse und werden daher auch nicht weiter betrachtet.

Es ist auch möglich, dass eine Turingmaschine zu keinem Ergebnis kommt [37]. Dies ist das sogenannte Halteproblem, für uns jedoch nicht weiter von Bedeutung.

Spricht man von einer Turingmaschine ist meist eine deterministische Turingmaschine (DTM) gemeint. Diese muss zuerst formal definiert werden.

Definition 47. Eine *Turingmaschine* ist gegeben durch ein 7-Tupel [37]

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E).$$

Hierbei sind:

- Z die endliche *Zustandsmenge*
- Σ das *Eingabealphabet*
- $\Gamma \supset \Sigma$ das *Arbeitsalphabet* auch Bandalphabet genannt
- $\delta\colon Z \times \Gamma \to Z \times \gamma \times \{L, R, N\}$ im deterministischen Fall (bzw. $\delta\colon Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{L, R, N\})$ im nichtdeterministischen Fall) die *Überführungsfunktionen*
- $z_0 \in Z$ der *Startzustand*
- $\square \in \Gamma$-Σ das *Blank*
- $E \subseteq Z$ die Menge der *Endzustände*

Eine deterministische Turingmaschine wird laut dieser Definition bei zwei identischen Eingaben auch exakt den gleichen Weg zur Berechnung des Ergebnisses wählen. Deterministisches Verhalten lässt sich immer sehr gut voraussagen bzw. berechnen.

Analog zur DTM gibt es die NTM, eine nichtdeterministische Turingmaschine. Sie wird durch das gleiche 7-Tupel definiert und unterscheidet sich wie in Definition 47 zu sehen ist nur durch δ, die Überführungsfunktionen. Dort wird nun in eine Potenzmenge überführt, folglich kann eine NTM in mehreren Zuständen gleichzeitig sein und somit auch verschiedene Wege wählen.

Orakel-Turingmaschine

Orakel-Turingmaschinen (OTM) sind Turingmaschinen mit einem speziellen Orakelband. Wenn die Turingmaschine auf das Orakelband zugreifen will, wechselt sie in einen dafür vorgesehenen *Orakelzustand*, sodass sie nach nur einem Zeitschritt ein Ergebnis erhält. Dies lässt sich auch so verstehen, dass die Turingmaschine die Möglichkeit hat ein Problem ohne Aufwand zu lösen. Eine Turingmaschine, die Zugriff auf ein solches Orakel hat, wird dann als M^O bezeichnet.

Es kann durch Forschung (insbesondere an und mit Quantencomputern) dazu kommen, dass ein Algorithmus gefunden wird, der exakt eine bereits verwendete Orakelfunktion erfüllt. Dies würde bedeuten, dass ein Problem, dass bis dahin nur nur von einer OTM gelöst werden kann, ab diesem Zeitpunkt nun von einer DTM berechenbar ist. So

können zur Zeit Algorithmen geschrieben werden, die Zugriff auf ein solches Orakel haben und mit Hilfe dieses Orakels Verschlüsselungen brechen können. Dies bleibt solange nur ein theoretisches Konstrukt, bis ein Algorithmus gefunden wird, der exakt das tut, was das Orakel bis dahin für den Algorithmus übernommen hat. Kann die Orakelfunktion nun *effizient* berechnet werden, ist somit auch möglich diese in den fertigen Algorithmus einzusetzen und somit die besagte Verschlüsselung zu brechen.

Im Bezug auf Verschlüsselungsverfahren bedeutet das, je mehr Orakelfunktionen notwendig sind um eine Verschlüsselung rückzuberechnen, desto tendenziell sicherer ist ein Verfahren für die weitere Zukunft.

Quantenalgorithmen liegen in der Klasse **BQP** und sind Algorithmen, die mit Quantenbits rechnen und Zugriff auf ein sogenanntes Quantenorakel haben, ähnlich zu den Orakel-Turingmaschinen.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] F. L. Bauer. *Entzifferte Geheimnisse: Methoden und Maximen der Kryptologie, 3. Auflage*. Springer, 2000.

[2] F. L. Bauer. *Historische Notizen zur Informatik*. Springer Berlin Heidelberg, 2008.

[3] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996.

[4] C.H. Bennett and S.J. Wiesner. Communication vio one- and two-particle operators on einstein-podoslky-rosen channels. *Phys. Rev. Letters 69*, pages 2881–2884, 1992.

[5] E. Bernstein and U. V. Vazirani. Quantum complexity theory. *SIAM J. Comput.*, 26(5):1411–1473, 1997.

[6] Daniel P. Bovet and Pierluigi Crescenzi. *Introduction to the theory of complexity*. Prentice Hall international series in computer science. Prentice Hall, 1994.

[7] G. Brassard, P. Høyer, and A. Tapp. Quantum cryptanalysis of hash and claw-free functions. In C. L. Lucchesi and A. V. Moura, editors, *LATIN '98: Theoretical Informatics, Third Latin American Symposium, Campinas, Brazil, April, 20-24, 1998, Proceedings*, volume 1380 of *Lecture Notes in Computer Science*, pages 163–169. Springer, 1998.

[8] J. A. Buchmann. *Einführung in die Kryptographie, 6. Auflage*. Springer, 2016.

[9] J. A. Buchmann, E. Dahmen, and M. Szydlo. Hash-based digital signature schemes. In Daniel J. Bernstein, Johannes A. Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, volume 1, pages 35–93. Springer, 2009.

[10] L. Chen, S. Jordan, Y. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone. Report on post-quantum cryptography. *NISTIR*, 2016.

[11] V. S. Denchev, S. Boixo, S. V. Isakov, N. Ding, R. Babbush, V. Smelyanskiy, J. Martinis, and H. Neven. What is the computational value of finite range tunneling? 2016.

[12] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[13] D.Naor, A. Shenhav, and A. Wool. One-time signatures revisited: Have they become practical? *IACR Cryptology ePrint Archive*, 2005:442, 2005.

[14] C. Dods, N. P. Smart, and M. Stam. Hash based digital signature schemes. In N. P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *Lecture Notes in Computer Science*, pages 96–115. Springer, 2005.

[15] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev. Letters 47*, pages 777–780, 1935.

[16] Stefan Filipp. Quantencomputer: Der Beginn der kommerziellen Quanten-Ära, 2018. https://www.ibm.com/de-de/blogs/think/2018/02/23/quantencomputer/.

[17] Fredkin and Toffoli. Conservative logic. *Int. Journal Theoretical Physics 21*, pages 219–253, 1982.

[18] L. C. C. García. On the security and the efficiency of the merkle signature scheme. 2005.

[19] J. Gill. Computational complexity of probabilistic turing machines. *SIAM J. Comput.*, 6(4):675–695, 1977.

[20] J. Grollmann and A. L. Selman. Complexity measures for public-key cryptosystems. *SIAM J. Comput.*, 17(2):309–335, 1988.

[21] J. Gruska. Quantum challenges. In *SOFSEM '99, Theory and Practice of Informatics, 26th Conference on Current Trends in Theory and Practice of Informatics, Milovy, Czech Republic, November 27 - December 4, 1999, Proceedings*, pages 1–28, 1999.

[22] M. Homeister. *Quantum Computing versthen, 1. Auflage*. Vieweg, 2005.

[23] D. S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 67–161. 1990.

[24] C. Karpfingfen and H. Kiechle. *Kryptologie*. Vieweg+Teubner, 2010.

[25] R. Kippenhahn. *Verschlüsselte Botschaften*. rowohlt-Verlag, 2012.

[26] H. O. Kunz. On the equivalence between one-dimensional discrete walsh-hadamard and multidimensional discrete fourier transforms. *IEEE Trans. Computers*, 28(3):267–268, 1979.

[27] M. Luby. *Pseudorandomness and cryptographic applications*. Princeton computer science notes. Princeton University Press, 1996.

[28] A. Meier and H. Vollmer. *Komplexität von Algorithmen*. Mathematik für Anwendungen, 4. Auflage. Lehmanns Media, 2015.

[29] R. C. Merkle. *Secrecy, authentication, and public key systems*. Ann Arbor, Mich. : UMI Research Press, 1982. Revision of the author's thesis (Ph. D.–Stanford University, 1979).

[30] R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.

[31] G. L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13(3):300–317, 1976.

[32] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Inforamtion*, volume 10th Aniversy Edition. Cambride University Press, 2000.

[33] E. G. Rieffel and W. Polak. An introduction to quantum computing for non-physicists. *ACM Comput. Surv.*, 32(3):300–335, 2000.

[34] R.L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[35] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In H. Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 387–394. ACM, 1990.

[36] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[37] U. Schöning. *Theoretische Informatik kurz gefaßt, 5. Auflage*. Spektrum Akademischer Verlag, 2009.

[38] E. Schrödinger. Die gegenwärtige Situation in der Quantenmechanik. *Die Naturwissenschaften*, 23(49):823–828, 1935.

[39] P.W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134, 1994.

[40] V. G. Umaña. *Post-Quantum Cryptography*. Dissertation, Technical University of Denmark, 2011.

[41] H. Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999.

[42] J. E. Whitesitt. *Boolesche Algebra und ihre Anwendungen*. Logik und Grundlagen der Mathematik, 2. Auflage. Vieweg+Teubner, 1970.