Schuljahr: 2019/20

Jahrgangsstufe: Q1.2

Facharbeit

aus dem Fach

Mathematik

Thema: Kryptographie in der Vergangenheit, heute und in der Zukunft mit Anwendung des RSA-Kryptosystems und besonderer Betrachtung des Shor-Algorithmus.

Verfasser: Nin0

Kurs: Mathematik Leistungskurs

https://github.com/N-I-N-0/Facharbeit-Kryptographie/blob/master/Kryptographie.pdf

Inhaltsverzeichnis

1 Einführung

1.1 Vorwort

Bei diesem Dokument handelt es sich um eine Facharbeit über die Kryptographie mit expliziter Betrachtung des RSA-Kryptosystems und des Shor-Algorithmus. Dabei sollen erst ein paar geschichtliche Entwicklungen und die Grundlagen schrittweise beschrieben werden bis schlussendlich ein Verständnis für das Gesamtbild entsteht. Der Zusammenhang einzelner Themen wird also erst zum Ende ersichtlich, was die Möglichkeit bieten soll auch ohne Vorkenntnisse ein Verständnis für das Gesamtbild zu erreichen.
Diese Facharbeit ist in zwei Teile aufgeteilt, wobei der erste Teil hauptsächlich um die geschichtliche Entwicklung der Kryptographie bis hin zum RSA-Kryptosystems, welches aus mathematischer Sicht beschrieben werden soll, gehen soll. Der zweite Teil soll Grundlagen von Quantencomputern vermitteln und den Shor-Algorithmus, welcher die Sicherheit vom RSA-Kryptosystems gefährdet, in Grundzügen erklären.

1.2 Begriffserklärung Kryptographie

Im einfachsten Sinne bedeutet Kryptographie so viel wie Geheimschrift, also eine Information für andere unkenntlich zu machen und somit geheim zu halten. Der Kryptographie steht die Kryptoanalyse gegenüber, welche das Ziel verfolgt, verschlüsselte Texte zu entschlüsseln. Man spricht dabei auch davon kryptographische Verfahren zu brechen beziehungsweise zu knacken.
Der Begriff Kryptologie wird alltagssprachlich oft mit Kryptographie gleichgesetzt, beschreibt jedoch zum Beispiel beim US-Militär als Oberbegriff zugleich die Kryptographie und die Kryptoanalyse.[1]

Die Kryptographie ist von der Steganographie abzugrenzen. Bei dieser geht es nicht um das unkenntlich machen einer Information, sondern um das Verbergen des Kanals, über welchen kommuniziert wird. Dritte erfahren hierbei nicht nur

[1] https://de.wikipedia.org/wiki/Kryptographie#Terminologie

keinen Inhalt einer Nachricht, sie wissen im Erfolgsfall nicht einmal, dass kommuniziert wurde.[2]

Man unterscheidet im allgemeinen zwischen der modernen und der klassischen Kryptographie. Die moderne Kryptographie verfolgt dabei folgende vier Ziele:

1. Vertraulichkeit — Dritte sollen die zu übermittelnden Daten nicht mitlesen können
2. Integrität — Die Daten müssen nachweislich vollständig und unverändert sein
3. Authentizität — Der Urheber der Daten muss eindeutig bestimmbar sein
4. Verbindlichkeit — Die Urheberschaft darf nicht bestreitbar sein[3]

Zu der klassischen Kryptographie zählt man Verfahren zur Verschlüsselung von Nachrichten, welche vor dem Zeitalter der Computer entstanden und verwendet wurden. Bei solchen Verfahren werden stets gesamte Buchstaben oder Buchstabengruppen ersetzt. Durch die heutige Kryptoanalyse gelten diese Verfahren als veraltet und unsicher. Klassische Kryptographie baute vor allem auf die folgenden zwei Methoden:

1. Transposition: Neuanordnung der Buchstaben einer Botschaft
2. Substitution: Ersetzung der Buchstaben mit anderen Buchstaben oder Symbolen

Bei der Substitution sei zwischen der monoalphabetischen und der polyalphabetischen Substitution zu unterscheiden.[4]

2 Kryptographische Verfahren

2.1 Symmetrische Verschlüsselung

Die klassische Kryptographie baute auf symmetrische Verschlüsselung. Es gibt auch moderne symmetrische Verfahren, die in der Praxis aber meistens nur

[2] https://de.wikipedia.org/wiki/Kryptographie#Abgrenzung_zur_Steganographie

[3] https://de.wikipedia.org/wiki/Kryptographie#Ziele_der_Kryptographie

[4] https://de.wikipedia.org/wiki/Kryptographie#Methoden_der_Kryptographie

noch in Kombination mit asymmetrischen Verfahren verwendet werden. Symmetrische Verschlüsselungen verwenden den selben Schlüssel zum Ver- und Entschlüsseln. An klassischen kryptographischen Verfahren soll das Prinzip solcher Verschlüsselungen gezeigt werden.[5]

2.1.1 Skytale

Ein Beispiel für ein auf Transposition basierendes Verfahren ist die Skytale, welche bereits vor ungefähr 2500 Jahren von den Spartanern verwendet worden sein soll. Bei diesem Verfahren wird ein beschreibbarer Streifen um einen Stab mit bestimmten Durchmesser gewickelt und anschließend seitlich beschrieben. Anschließend wird der Streifen abgezogen und ist vertikal nun nicht mehr zu lesen. Erst nach umwickeln um einen Stab mit identischem Durchmesser, kann dann die Nachricht wieder gelesen werden.[6]

2.1.2 Caesar-Verschlüsselung

Ein Beispiel für die monoalphabetische Substitution ist die Caesar-Verschlüsselung. Bei dieser wird das Alphabet um eine gegebene Anzahl an Buchstaben zyklisch gedreht, wodurch ein Geheimalphabet entsteht. Anhand dieses neuen Alphabets lässt sich dann der Text verschlüsseln. Gleichermaßen kann mit der gegeben Anzahl an Rotationen zwischen dem originalen und dem neuen Alphabet, aus einer verschlüsselten Botschaft wieder der originale Text erschaffen werden. Eine mathematische Beschreibung dieser Verschlüsselung für ein klassisches lateinisches Alphabet mit 26 Buchstaben sieht wie folgt aus:

[P] = Klartextbuchstabe

[K] = Verschiebungsanzahl

[C] = Geheimtextbuchstabe

$$encrypt_K(P) = (P + K) \bmod 26$$

$$decrypt_K(C) = (C - K) \bmod 26$$[7]

---

[5] https://www.kryptowissen.de/symmetrische-verschluesselung.html

[6] http://www.mathe.tu-freiberg.de/~hebisch/cafe/kryptographie/skytale.html

[7] https://de.wikipedia.org/wiki/Caesar-Verschlüsselung

Während, wie man sehen kann, bei der monoalphabetischen Substitution jeder Buchstabe stets mit dem selben Ersatzbuchstaben getauscht und nur ein Geheimalphabet verwendet wird, gibt es bei Verfahren der polyalphabetischen Substitution mehrere Geheimalphabete. Ein weitreichend bekanntes Beispiel für dieses Verfahren ist die Enigma.

2.1.3 Enigma

Die Enigma wurde im Zweiten Weltkrieg in Deutschland zur verschlüsselten Kommunikation verwendet. Sie baut auf mehrere Buchstabenringe, welche hintereinander einen jeden Buchstaben substituieren. Nach jedem substituierten Buchstaben drehen sich jedoch diese Buchstabenringe, wodurch ein neues Geheimalphabet entsteht für den nächsten Buchstaben. Erst nach 26 Drehungen ist die Startstellung eines Buchstabenringes wieder erreicht, wodurch Muster die unter Anwendung einer monoalphabetischen Substitution noch erkennbar gewesen wären, nicht mehr vorhanden sind. Ein Buchstabe wird also nicht immer zum selben Ersatzbuchstaben substituiert.[8]

2.2 Asymmetrische Verschlüsselung

Asymmetrische Verfahren gelten als sicher aufgrund der Tatsache, dass sie auf sogenannten Falltürfunktionen basieren. Eine Falltürfunktion ist eine Einwegfunktion, welche unter bestimmten Zusatzbedingungen (die Falltür) auch rückwärts leicht zu berechnen ist, während dies eigentlich nicht möglich ist. Eine Einwegfunktion beschreibt eine Funktion, die in eine Richtung leicht und in die andere Richtung schwer zu berechnen ist.[9]

Bei solchen Verfahren unterscheiden sich Chiffrierschlüssel (Public-Key) und Dechiffrierschlüssel (Private-Key) voneinander. Im Erfolgsfall lässt sich aus dem einen Schlüssel der andere nicht berechnen. Während bei symmetrischen Verfahren für jeden neuen Teilnehmer ein neuer Schlüssel gegenüber jedem anderen, bereits existierenden Teilnehmer, erschaffen werden muss, reicht bei

---

[8] https://de.wikipedia.org/wiki/Enigma_(Maschine)#Prinzip

[9] https://de.wikipedia.org/wiki/Einwegfunktion#Einwegfunktionen_mit_Falltür_(Trapdoor-Einwegfunktionen)

asymmetrischen Verfahren das Hinzufügen des Public-Keys in die Liste aller Public-Keys im System, welcher dann von allen verwendet werden kann, um eine Nachricht so zu verschlüsseln, dass nur die Person mit dem zugehörigen Private-Key die Nachricht wieder entschlüsseln kann.[10] [11]

2.2.1 RSA

RSA ist ein asymmetrisches Public-Key-Kryptosystem, welches 1977 von Ronald Rivest, Adi Shamir und Leonard Adleman entwickelt wurde. Es war das erste veröffentlichte asymmetrische Verschlüsselungsverfahren.[12]
Im folgenden sollen die mathematischen Grundlagen des Verfahrens beschrieben werden. Anschließend folgt die Zusammensetzung der Grundlagen zum fertigen Algorithmus.

2.2.1.1 Teilbarkeit und Primzahlen

Bei asymmetrischen Verfahren stellt der Private-Key die Falltür der Falltürfunktion da. Das RSA Verfahren nutzt dabei aus, dass das Multiplizieren zweier Primzahlen schnell möglich ist, während die Primfaktorzerlegung zum heutigen Zeitpunkt für große Primzahlen nicht effizient gelöst werden kann.[13]

Zur Teilbarkeit einer Zahl sei folgendes zu sagen:

Angenommen a und b seien ganze Zahlen, so teilt die Zahl a die Zahl b genau dann, wenn für eine ganze Zahl k gilt: $b = a \cdot k$

Man schreibt $a \mid b$ und sagt: a ist ein Teiler von b.

Unter anderem gelten folgende Sätze zur Teilbarkeit:

1. $a \mid b \Rightarrow a \mid cb$, Beweis: $b = ak \Rightarrow bc = a(ck)$
2. $a \mid b \wedge a \mid c \Rightarrow a \mid (b \pm c)$, Beweis:
   $b = ak_1 \wedge c = ak_2 \Rightarrow (b \pm c) = (ak_1 \pm ak_2) = a(k_1 \pm k_2)$

[10] https://www.kryptowissen.de/asymmetrische-verschluesselung.html

[11] https://www.zum.de/Faecher/Inf/RP/infschul/kr_rsa.html

[12] https://de.wikipedia.org/wiki/RSA-Kryptosystem#Geschichte

[13] https://de.wikipedia.org/wiki/RSA-Kryptosystem#Einwegfunktionen

3. $a \mid (b \pm c) \wedge a \mid b \Rightarrow a \mid c$, Beweis:
   $(b \pm c) = ak_1 \wedge b = ak_2 \Rightarrow ak_1 = ak_2 \pm c \Rightarrow c = a(k_1 \pm k_2)$
4. $a \mid b \wedge b \neq 0 \Rightarrow |a| \leq |b|$, Beweis:
   aus $ak = b \Rightarrow |a| \cdot |k| = |b|$, aus $b \neq 0 \Rightarrow k \neq 0$,
   somit gilt $|k| \geq 1 \Rightarrow |a| \leq |b|$ [14]

Das RSA Verfahren funktioniert nur, da es eine sogenannte Eindeutigkeit der Primfaktorzerlegung gibt. Soll heißen: eine Zahl kann nur auf eine einzige Art durch Primzahlen zerlegt werden. Ohne die Eindeutigkeit der Primfaktorzerlegung gebe es mehrere mögliche Entschlüsselungen zu einer verschlüsselten Nachricht. Das RSA Verfahren funktioniert auch mit Carmichael-Zahlen, da sie das Produkt aus mindestens drei quadratfreien Primzahlen sind und somit ebenfalls eindeutig geteilt werden können.[15]

2.2.1.2 Größter gemeinsamer Teiler

Ein gemeinsamer Teiler t von zwei ganzen Zahlen a und b wird wie folgt definiert: $t \mid a \wedge t \mid b$.
Aus dem vierten in 2.2.1.1 beschriebenen Satz folgt, dass es nur eine endliche Anzahl an natürlichen Teilern für jede natürliche Zahl gibt. Dies wiederum erlaubt die Suche nach einem größten gemeinsamen Teiler (abgekürzt ggT). Man schreibt $t = \mathrm{ggT}(a, b)$.
Jede Zahl lässt sich durch die 1 teilen, weshalb gilt: $\mathrm{ggT}(a, b) \geq 1$. Für den Fall, dass $\mathrm{ggT}(a, b) = 1$ gilt, sagt man auch, dass a und b relativ prim zueinander sind.[16] Für eine Primzahl p gilt für jede Zahl $a < p$: $\mathrm{ggT}(a, p) = 1$.

2.2.1.3 Euklidischer Algorithmus

Der euklidische Algorithmus wird genutzt, um den größten gemeinsamen Teiler einer Zahl zu bestimmen. Ist von keiner der beiden Zahlen die Primfaktorzerle-

[14] Yimin Ge, Die Mathematik von RSA (August 2005), Seite 4-5; PDF auf Anfrage verfügbar

[15] https://mathepedia.de/Existenz_und_Eindeutigkeit.html

[16] Yimin Ge, Die Mathematik von RSA (August 2005), Seite 6; PDF auf Anfrage verfügbar

gung bekannt, so ist der euklidische Algorithmus das schnellste bekannte Verfahren zur Berechnung des größten gemeinsamen Teilers.
In der ursprünglichen Form wurde solange die größere der beiden Zahlen des jeweiligen Schrittes von der kleineren Zahl abgezogen, bis beide Zahlen den selben Wert erreichten oder in anderer Variante die eine der beiden Zahlen den Wert 0 annahm (einen Subtraktionsschritt weitergehend). Die übrig bleibende Zahl ist dann der größte gemeinsame Teiler der beiden ursprünglichen Zahlen.
Das Verfahren beruht auf der Tatsache, dass der größte gemeinsame Teiler zweier Zahlen sich nicht ändert, wenn von der größeren Zahl die kleinere abgezogen wird.
Mathematisch wie folgt ausgedrückt: $\mathrm{ggT}(a, b) = \mathrm{ggT}(b, a - kb)$ mit $k \in \mathbb{Z}$.
Bei der heutzutage verwendeten Variante wird das wiederholte Subtrahieren durch eine Division mit Rest ersetzt. In jedem Schritt wird mit dem Divisor und dem Rest des vorherigen Schrittes weitergerechnet, bis eine Division aufgeht, und keinen Rest hat. Der Divisor der letzten Division ist bei diesem Verfahren dann der größte gemeinsame Teiler.[17]

2.2.1.4 Erweiterter euklidischer Algorithmus

Nach dem Lemma von Bézout gilt für den $\mathrm{ggT}(a, b)$, dass dieser sich als Linearkombination von a und b mit ganzzahligen Koeffizienten darstellen lässt. Dies wird wie folgt ausgedrückt: $\mathrm{ggT}(a, b) = s \cdot a + t \cdot b$
Der erweiterte euklidische Algorithmus ist dazu in der Lage neben dem ggT auch die Faktoren s und t zu berechnen.
Die Funktionsweise des Algorithmus sei an folgendem Beispiel für die Zahlen 42 und 57 erklärt:

$$57 = 1 \cdot 42 + 15$$
$$42 = 2 \cdot 15 + 12$$
$$15 = 1 \cdot 12 + 3$$
$$12 = 4 \cdot 3 + 0$$

[17] Yimin Ge, Die Mathematik von RSA (August 2005), Seite 6-7; PDF auf Anfrage verfügbar

Bis hierhin handelt es sich um den euklidischen Algorithmus unter Verwendung von Division mit Rest. Die Zahl 3 ist somit der $\mathrm{ggT}(42,57)$.
Die dabei verwendeten Gleichungen können nun genutzt werden um rekursiv die Koeffizienten s und t zu berechnen. Die Berechnung der Koeffizienten beginnt mit den vorläufigen Werten $s = 1$ und $t = 0$. Dies kann wie folgt am ersten Schritt des Beispiels erklärt werden:

$t = s_{vorher} = 1$

$s = t_{vorher} - s_{vorher} \cdot ((a - a \bmod b)/b) = 0 - 1 \cdot (15 - 3)/12 = -1$

Im ersten Schritt wird kurz gefasst s aus der Multiplikation von -1 mit der 1 aus $15 = 1 \cdot 12 + 3$ berechnet, sprich: $s = -1 \cdot 1 = -1$. Aus dem Koeffizienten der vorletzten Gleichung des rekursiven Abstiegs berechnet sich, nicht nur im Beispiel, durch die Multiplikation mit -1 der erste, beim rekursiven Aufstieg verwendete, vorläufige Wert für s. Anschließend werden die anfänglichen rekursiven Abstiege des Beispiels wie folgt wieder aufgestiegen:

$3 = 1 \cdot 15 - 1 \cdot 12 \qquad \Rightarrow s = -1$ und $t = 1$

$3 = 1 \cdot 15 - 1 \cdot (42 - 2 \cdot 15) = 3 \cdot 15 - 1 \cdot 42 \qquad \Rightarrow s = 3$ und $t = -1$

$3 = 3 \cdot (1 \cdot 57 - 1 \cdot 42) - 1 \cdot 42 = 3 \cdot 57 - 4 \cdot 42 \qquad \Rightarrow s = -4$ und $t = 3$

Man setzt hier also die Restdarstellungen der vorherigen Schritte ineinander ein. Beim rekursiven Abstieg des Algorithmus wird der ggT berechnet und beim rekursiven Aufstieg die Koeffizienten s und t, die in diesem Fall $s = -4$ und $t = 3$ sind, sodass gilt: $\mathrm{ggT}(42,57) = 3 = -4 \cdot 42 + 3 \cdot 57$.[18]

2.2.1.5 Die eulersche φ-Funktion

Die eulersche φ-Funktion zu einer natürlichen Zahl n, gibt die Anzahl positiver, ganzer Zahlen kleiner gleich n an, die zu n teilerfremd sind:

$$\varphi(n) := \left| \{a \in \mathbb{N} \mid 1 \leq a \leq n \wedge \mathrm{ggT}(a,n) = 1\} \right|$$ [19]

[18] https://de.wikipedia.org/wiki/Erweiterter_euklidischer_Algorithmus

[19] https://de.wikipedia.org/wiki/Eulersche_Phi-Funktion

Beispiel:

$\varphi(12) = 4$, da nur für $a \in \{1,5,7,11\}$ die Bedingung $\mathrm{ggT}(a,12) = 1$ gilt.

Da Primzahlen nur durch 1 und sich selbst geteilt werden können, gilt für eine Primzahl p: $\varphi(p) = p - 1$.

Die φ-Funktion ist multiplikativ für teilerfremde natürliche Zahlen.[20] Somit gilt: $\varphi(p \cdot q) = \varphi(p) \cdot \varphi(q)$, falls $\mathrm{ggT}(p, q) = 1$

Ein Beispiel: $\varphi(12) = \varphi(3) \cdot \varphi(4) = 2 \cdot 2 = 4$

Ein Gegenbeispiel durch zwei Zahlen p und q mit gemeinsamen Primfaktoren:

$\varphi(2 \cdot 6) = \varphi(12) = 4 \neq \varphi(2) \cdot \varphi(6) = 1 \cdot 2 = 2$

Da Primzahlen zwangsweise zueinander teilerfremd sind, gilt für zwei Primzahlen p und q: $\varphi(p \cdot q) = (p - 1) \cdot (q - 1)$.

2.2.1.6 Multiplikative Gruppe modulo n

Im weiteren Verlauf wird die multiplikative Gruppe modulo n benötigt. Deswegen seien zunächst einmal Gruppen in der Mathematik wie folgt definiert:

„Eine Menge G zusammen mit einer inneren Verknüpfung $\circ$, geschrieben $(G, \circ)$, heißt Gruppe, sobald die Verknüpfung assoziativ ist und es sowohl ein neutrales Element e gibt, dass für alle $a \in G$ die Gleichung $a \circ e = e \circ a = a$ erfüllt, als auch zu jedem $a \in G$ ein inverses Element $a^{-1} \in G$ existiert mit $a \circ a^{-1} = a^{-1} \circ a = e$.“[21]

Die multiplikative Gruppe modulo n besteht aus der Menge $\mathbb{Z}_n^*$, der 1 als neutrales Element und der Multiplikation modulo n als Verknüpfung. $\mathbb{Z}_n^*$ ist wie folgt definiert: $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \mathrm{ggT}(a, n) = 1\}$, mit $n \in \mathbb{N}$.

Die Multiplikation zweier Zahlen a und b modulo n wird als $(a \cdot b) \bmod n$ ausgedrückt.

---

[20] https://mathepedia.de/Eulersche_Phi-Funktion.html

[21] https://repositorium.uni-muenster.de/document/miami/92fac8b4-ce4d-4643-8c83-f8826518e856/bachelor_mandrysch_2017.pdf, Seite 8

·	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

[22]

Die Verknüpfungstafel für die Multiplikation modulo 5.

Da die 5 eine Primzahl ist, gilt für $n = 5$: $\mathbb{Z}_n^* = \mathbb{Z}_n$, denn die Menge $\mathbb{Z}_n^*$ besteht aus allen zu n teilerfremden Elementen aus $\mathbb{Z}_n$.

Die Anzahl der Elemente in einer multiplikativen Gruppe modulo n kann durch die eulersche φ-Funktion bestimmt werden: $\varphi(n) = |\mathbb{Z}_n^*|$.[23]

2.2.1.7 Das multiplikative Inverse

Das multiplikative Inverse a^{-1} einer Zahl a in der Gruppe $\mathbb{Z}_n^*$ ist das eindeutig bestimmte Element, für das gilt: $a \cdot a^{-1} = a^{-1} \cdot a = 1$, wobei 1 das neutrale Element der Gruppe ist.
Ein Beispiel: In $\mathbb{Z}_9^*$ ist 2 das inverse Element zu 5 und umgekehrt, da gilt:

$$2 \cdot 5 \equiv 5 \cdot 2 \equiv 10 \equiv 1 \pmod 9$$

Das inverse Element eines Elements $a \in \mathbb{Z}_n^*$ lässt sich mit dem erweiterten euklidischen Algorithmus oder durch modulare Exponentiation auf Grundlage des Satzes von Euler berechnen. Im angefügten Programm wird der erweiterte euklidische Algorithmus verwendet, weshalb nur dieser Weg weiter erklärt werden soll:

Bei dem erweiterten euklidischen Algorithmus gilt, wie zuvor erklärt, dass es zwei ganze Zahlen s und t geben muss, sodass gilt: $\mathrm{ggT}(a, b) = s \cdot a + t \cdot b$.
Da die multiplikative Gruppe $\mathbb{Z}_n^*$ nur aus Elementen von $\mathbb{Z}_n$, die zu n teilerfremd sind, besteht, gilt für jedes $a \in \mathbb{Z}_n^*$: $\mathrm{ggT}(a, n) = 1$.

---

[22] https://www.math.tu-dresden.de/~ganter/inf0708/folien/inf07-Modulo.pdf, Seite 9

[23] https://www.inf.hs-flensburg.de/lang/krypto/grund/gruppezn.htm

Wie in 2.2.1.4 erklärt, beschreiben wir nun die 1, nach dem Lemma von Bézout, als ganzzahlige Linearkombination von a und b beziehungsweise a und n:

$$1 = s \cdot a + t \cdot n$$

Dies kann modulo n gerechnet werden:

$$1 \equiv s \cdot a + t \cdot n \equiv s \cdot a \pmod{n}$$

Die Multiplikation mit a^{-1} ergibt zuletzt:

$$a^{-1} \equiv s \pmod{n}$$

Für das inverse Element von a in $\mathbb{Z}_n^*$ gilt damit:

$$a^{-1} = s \bmod n$$[24]

2.2.1.8 Der Satz von Euler

Die Gruppentheorie besagt, dass für eine endliche Gruppe $(G, \circ, e)$ mit e als neutralem Element für alle $a \in G$ gilt: $a^{|G|} = e$.

Der Satz von Euler besagt für zwei Zahlen $a, n \in \mathbb{N}$:

$$\mathrm{ggT}(a, n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1 \pmod{n}$$[25]

Beweis:

Es sei $\mathbb{Z}_n^* = \{r_1, \ldots, r_{\varphi(n)}\}$ die Menge der multiplikativ modulo n invertierbaren Elemente oder anders ausgedrückt also die Menge der zu n teilerfremden Zahlen. Es gelte zudem $\mathrm{ggT}(a, n) = 1$.

Alle zu n teilerfremde Zahlen wären damit: $r_1, r_2, \ldots, r_{\varphi(n)}$.

Multipliziert man diese mit a, so erhält man: $ar_1, ar_2, \ldots, ar_{\varphi(n)}$.

Da a wie zu Beginn definiert ebenfalls zu n teilerfremd ist, sind auch die hier entstandenen Produkte weiterhin teilerfremd zu n. Also: $\mathrm{ggT}(ak_i, n) = 1$.

Die Menge $\{ar_1, ar_2, \ldots, ar_{\varphi(n)}\}$ ist hierbei eine Permutation der Menge $\{r_1, r_2, \ldots, r_{\varphi(n)}\}$. Beispiel für $n = 8$: $\mathbb{Z}_8^* = \{1, 3, 5, 7\}$.

---

[24] https://www.inf.hs-flensburg.de/lang/krypto/grund/inverses-element.htm

[25] https://www.inf.hs-flensburg.de/lang/krypto/grund/fermat-euler.htm, Satz von Euler

Mit $a = 3$ multipliziert modulo n entsteht die Menge:
$\{1 \cdot 3 \bmod 8, 3 \cdot 3 \bmod 8, 5 \cdot 3 \bmod 8, 7 \cdot 3 \bmod 8\} = \{3, 1, 7, 5\}.$

Es entstehen also keine neuen Restklassen. Dies kann trivial bereits damit begründet werden, dass die Menge $\mathbb{Z}_n^*$ alle zu n teilerfremden Zahlen und somit alle möglichen Restklassen beinhaltet.

Es kann zudem nicht mehrfach das gleiche r_i für verschiedene ar_i herauskommen, sprich: $ar_i \not\equiv ar_j \pmod{n}$. Der Beweis hierfür:
Angenommen es gelte: $ar_i \equiv ar_j \pmod{n}$, so dürfte man, da a und n teilerfremd sind, beide Seiten durch a teilen. Übrig bliebe dann: $r_i \equiv r_j \pmod{n}$.
Da aber nun die Menge $\mathbb{Z}_n^*$ so definiert wurde, dass sie alle zu n teilerfremden Zahlen je einmal enthält, können zwei verschiedene Zahlen r_i und r_j mit der selben Zahl a multipliziert auch nur zwei verschiedene Zahlen als Produkt besitzen. Damit beweist der Widerspruch: $ar_i \not\equiv ar_j \pmod{n}$.

Da nun aber die Mengen $\{r_1, r_2, \ldots, r_{\varphi(n)}\}$ und $\{ar_1, ar_2, \ldots, ar_{\varphi(n)}\}$, wie zuvor bewiesen, die selben Elemente besitzen, gilt durch das Kommutativgesetz:
$r_1 \cdot r_2 \cdots r_{\varphi(n)} \equiv ar_1 \cdot ar_2 \cdots ar_{\varphi(n)} \pmod{n}$

Da alle r_i nach Definition teilerfremd zu n sind, darf man nun jedoch durch alle r_i dividieren. Damit ist der Satz von Euler bewiesen, denn übrig bleibt:
$1 \equiv a^{\varphi(n)} \pmod{n}$[26]

Die bis hierhin erklärte Form des Satzes von Euler kann theoretisch zur Findung des multiplikativen Inversen eines Elementes aus $\mathbb{Z}_n^*$ verwendet werden.

[26] https://www.youtube.com/watch?v=DU082wcr40A

Zur Entschlüsselung einer Nachricht dient jedoch eine abgeänderte Form des Satzes von Euler:

Da a zu n teilerfremd ist, lässt sich $1 \equiv a^{\varphi(n)} \pmod{n}$ mit a multiplizieren:

$a \equiv a^{\varphi(n)+1} \equiv a^{k\cdot\varphi(n)+1} \pmod{n}$ mit $k \in \mathbb{N}_0$.

Diese Form des Satzes von Euler soll im Gegensatz zur vorherigen für beliebige $a \in \mathbb{N}_0$ gelten.[27]

2.2.1.9 Die Implementation (in Ruby)

Die mathematischen Grundlagen des RSA-Algorithmus sind nun erklärt. Die Implementation erfordert jedoch noch zwei weitere Verfahren: Ein zeitlich effizienter Primzahltest sowie ein Verfahren zur zeitlich akzeptablen modularen Exponentiation. Ohne diese wäre der RSA-Algorithmus nicht nützlich anwendbar, da zu viel Zeit zum Ver- und Entschlüsseln benötigt werden würde, sodass die zu übertragende Nachricht in den meisten Fällen ihre Bedeutung verlieren würde, bevor sie verschlüsselt, verschickt und wieder entschlüsselt wäre.

Vorab: Die vollständige Implementation, programmiert in der Programmiersprache Ruby, befindet sich im Anhang.

2.2.1.9.1 Miller-Rabin-Test

Der RSA-Algorithmus benötigt zwei, zufällig gewählte, große Primzahlen mit einer üblichen Länge von einigen hundert Bits ($\geq 2^{100}$). Um eine Primzahl zu bestimmen, kann man ganz prinzipiell erst einmal für eine mögliche Primzahl p für jede Zahl $a < p$ den $\mathrm{ggT}(a, p)$ berechnen. Gilt dabei für irgendein a $\mathrm{ggT}(a, p) \neq 1$, so ist p keine Primzahl, da die Definition einer Primzahl ja gerade ist, dass sie nur durch sich selbst und 1 teilbar ist.

Tatsächlich würde es bereits reichen den $\mathrm{ggT}(a, p)$ für alle $a \leq \sqrt{p}$ zu berechnen, da für jedes $a \geq \sqrt{p}$ gilt, dass falls dieses p teilt, der andere Faktor $\leq \sqrt{p}$ sein müsste und somit bereits vor dem Erreichen eines $a \geq \sqrt{p}$ feststünde, dass p keine Primzahl sein kann.

---

[27] https://www.inf.hs-flensburg.de/lang/krypto/grund/fermat-euler.htm, Modifizierter Satz von Euler

Nichtsdestotrotz bleiben für eine Primzahl $p = 2^{100}$ somit bereits eine Anzahl von $\sqrt{2^{100}} = 2^{50} = 1125899906842624$ Testwerten für a, was schnell zeigt, dass der intuitive und primitive Primzahltest nicht effizient genug aussagen kann, ob eine Zahl eine Primzahl ist.

Hier kommt der probabilistische Miller-Rabin-Primzahltest ins Spiel. Der Test verwendet zufällige Werte auf eine hier nicht weiter erklärte Art und Weise und kann dann mit einer bestimmten Wahrscheinlichkeit sagen, ob eine Zahl prim ist oder, dass eine Zahl definitiv nicht prim ist. Der Test kann jedoch nicht garantieren, dass eine Zahl prim ist. Dies ist jedoch nicht weiter dramatisch, da die Wahrscheinlichkeit, dass eine Zahl wirklich prim ist mit der Anzahl an Durchläufen k des Algorithmus steigt, solange nicht bewiesen werden kann, dass die gegebene Zahl nicht prim ist. Die Wahrscheinlichkeit, dass eine zusammengesetzte Zahl nicht als solche erkannt wird beträgt nämlich andersherum betrachtet $< \frac{1}{2^k}$ und wird somit mit jedem Durchlauf des Algorithmus geringer[28].

In der Implementation wird $k = 20$ verwendet, was eine Irrtumswahrscheinlichkeit von $< \frac{1}{2^{20}} = 0.00000095367431640625$ übrig lässt.

2.2.1.9.2 Effiziente modulare Exponentiation

Für eine effiziente modulare Exponentiation macht man sich zum Vorteil, dass $z = x^4$ auch über den Umweg $y = x^2$ berechnet werden kann: $z = y^2$. Für die Berechnung von $z = x^4 = x \cdot x \cdot x \cdot x$ werden drei Multiplikationen benötigt, für $z = y^2 = y \cdot y$ mit $y = x^2 = x \cdot x$ hingegen nur zwei.

Um nun das Ergebnis einer Exponentiation mit großem Exponenten unter möglichst geringem Zeitaufwand zu berechnen, verwendet man die binäre Darstellung des Exponenten: $[21]_{10} = [10101]_2$.
Somit gilt: $m^{21} = m^{16 \cdot 1 + 8 \cdot 0 + 4 \cdot 1 + 2 \cdot 0 + 1 \cdot 1} = m^{16} \cdot m^4 \cdot m^1$

[28] https://www.inf.hs-flensburg.de/lang/krypto/algo/primtest.htm

Eine minimale Erklärung des programmierten Codes:

```
def modulare_exponentiation(basis, exponent, modul)
   ergebnis = 1
```

Zunächst werden die Basis, der Exponent und das Modul übergeben. Zudem wird eine Variable „ergebnis“ mit dem Anfangswert 1 erstellt.

```
   while exponent > 0
```

Dann beginnt eine Schleife, die solange läuft, bis der Exponent auf 0 reduziert ist. Dabei sei beachtet, dass in jedem Durchlauf sich der Exponent halbiert.

```
      if exponent % 2 == 1
         ergebnis = (ergebnis * basis) % modul
      end
```

Endet der Exponent im derzeitigen Durchlauf mit einer 1, wird „ergebnis“ mit dem derzeitigen Wert aus „basis“ multipliziert und das Ergebnis modulo „modul“ gerechnet und in „ergebnis“ gespeichert.

```
      basis = (basis * basis) % modul
```

In jedem Durchlauf wird die Basis quadriert und modulo „modul“ gerechnet. In diesem Schritt liegt die Reduktion der Anzahl benötigter Multiplikationen.

```
      exponent >>= 1;
```

Der Exponent wird bitweise um eins nach rechts verschoben. Sprich: die binäre Darstellung der Zahl verschiebt sich um eins nach rechts, was das Dividieren durch 2 bedeutet.

```
   end
   return ergebnis
end
```

Zum Schluss wird nur noch das berechnete Ergebnis zurückgegeben.

Einen Beweis für die Korrektheit dieser Methode konnte ich nicht auffinden. Jedoch gehört diese Methode nicht zu den mathematischen Grundlagen des Algorithmus selbst, da prinzipiell auch einfach $\text{basis}^{\text{exponent}} \bmod \text{modul}$ gerechnet werden könnte, was nur gegebenenfalls sehr viel Zeit beanspruchen würde. Ich wage jedoch die Behauptung aufzustellen, dass die Webseite der Hochschule

Flensburg[29] als eine vertrauenswürdige Quelle anerkannt werden kann, sodass ein weiterer Beweis trivial ist.

2.2.1.9.3 Der Ablauf des Algorithmus

Nun folgt schlussendlich der Ablauf des RSA-Algorithmus:

1. Es werden zwei Primzahlen $p \neq q$ mit dem Miller-Rabin-Test bestimmt.
2. Es wird das RSA-Modul N berechnet: $N = p \cdot q$
3. Es wird die eulersche φ-Funktion von N berechnet:
   $\varphi(N) = \varphi(p) \cdot \varphi(q) = (p-1) \cdot (q-1)$
4. Es wird der Verschlüsselungsexponent e festgelegt, standardmäßig:
   $e = 2^{16} + 1 = 65537$
5. Es wird der Entschlüsselungsexponent d berechnet als multiplikatives Inverses von e in Bezug zum Modul $\varphi(N)$. Dies geschieht wie in 2.2.1.7 beschrieben mit Hilfe des erweiterten euklidischen Algorithmus mit e und $\varphi(N)$ als Übergabewerten. Dabei sei angemerkt, dass hierbei aus $\mathrm{ggT}(e, \varphi(N)) = d \cdot e + k \cdot \varphi(N) = 1$ folgt: $d \cdot e \mod \varphi(N) = 1$.
   (d und k sind im erweiterten euklidischen Algorithmus als s und t bezeichnet worden.)[30]

Der öffentliche Schlüssel ist das Zahlenpaar (e, N) und der private Schlüssel das Zahlenpaar (d, N).

Im Program ist der Verschlüsselungsexponent $e = 65537$ bereits festgelegt.

Das RSA-Modul N wird während des Verschlüsselns aus der Multiplikation zweier zufälliger Primzahlen erstellt. Es wird von der Person, die die Nachricht empfangen möchte bereitgestellt.

Der Entschlüsselungsexponent d bleibt geheim und kann nur berechnet werden, wenn $\varphi(N)$ bekannt ist, was erfordert die Primzahlen p und q zu kennen.

---

[29] https://www.inf.hs-flensburg.de/lang/krypto/algo/modexp2.htm, Program (Python Vorlage)

[30] https://de.wikipedia.org/wiki/RSA-Kryptosystem#Erzeugung_des_öffentlichen_und_privaten_Schlüssels

Diese aus N zu berechnen ist bisher für große Zahlen nicht effizient möglich. Es wird hierbei auch vom RSA-Schlüsselproblem gesprochen. Dies ist eine der zwei Grundlagen der Sicherheit vom RSA-Algorithmus.

Eine Nachricht m kann nun verschlüsselt werden, indem sie in eine eindeutig zurück umwandelbare Zahl umgewandelt wird, mit der dann wie folgt gerechnet wird: $c = m^e \mod N$, wobei c den verschlüsselten Text als Zahl darstellt.
Wie sich sehen lässt, ließe sich theoretisch aus dem öffentlichen Schlüssel (e, N) der Klartext m berechnen. In der Praxis ist dies jedoch nicht möglich aufgrund der Schwierigkeit die e-te Wurzel modulo N zu bestimmen. Es handelt sich hierbei um das sogenannte RSA-Problem, was die zweite Grundlage der Sicherheit vom RSA-Algorithmus darstellt.[31]

Eine Nachricht c kann nun wiederum zum Klartext m entschlüsselt werden, indem folgende Rechnung durchgeführt wird: $m = c^d \mod N$.
Der Beweis dafür, dass so der ursprüngliche Text errechnet werden kann, sieht wie folgt aus:

$$c^d \mod N = m^{e^d} \mod N = m^{e \cdot d} \mod N$$

Wie anfangs, unter 5. erwähnt, gilt:

$$d \cdot e \mod \varphi(N) = 1 \Rightarrow d \cdot e = k \cdot \varphi(N) + 1$$

Wie in 2.2.1.8 beschrieben, gilt nach dem Satz von Euler zudem:

$$a \equiv a^{\varphi(n)+1} \equiv a^{k \cdot \varphi(n)+1} \pmod{n} \text{ mit } k \in \mathbb{N}_0$$

Aus diesen beiden Gegebenheiten folgt schlussendlich:

$$m^{e \cdot d} \mod N = m^{k \cdot \varphi(N)+1} \mod N = m$$

Somit ist bewiesen, dass es mit dem Entschlüsselungsexponenten d möglich ist den Klartext aus der verschlüsselten Nachricht zu berechnen.[32]
Die dabei berechnete Zahl muss zuletzt nur noch in einen Text zurück umgewandelt werden und die entschlüsselte Nachricht kann gelesen werden.

---

[31] https://de.wikipedia.org/wiki/RSA-Kryptosystem#Beziehung_zwischen_RSA_und_dem_Faktorisierungsproblem

[32] https://www.inf.hs-flensburg.de/lang/krypto/protokolle/rsa.htm, Entschlüsselung

3 Shor-Algorithmus

Der Shor-Algorithmus ist ein Faktorisierungsverfahren, welches 1994 von Peter Shor veröffentlicht wurde. Das Besondere an dem Verfahren ist dabei, dass es in sogenannter polynomieller Laufzeit nicht triviale Teiler einer Zahl finden kann. Anders ausgedrückt, kann mit dem Shor-Algorithmus das Faktorisierungsproblem effizient gelöst werden. Dies stellt gerade für das zuvor vorgestellte RSA-Kryptosystem ein großes Problem dar, da somit die Primfaktoren p und q zurückberechnet werden können aus der Zahl N, welche Teil des öffentlichen Schlüssels (e, N) ist. Kennt man erst einmal p und q, kann man anschließend auch $\varphi(N) = \varphi(p \cdot q) = (p-1) \cdot (q-1)$ berechnen und damit wiederum d des privaten Schlüssels (d, N) mit Hilfe des erweiterten euklidischen Algorithmus: $\mathrm{ggT}(e, \varphi(N)) = d \cdot e + k \cdot \varphi(N) = 1$.
Man kann also durch den Shor-Algorithmus effizient den privaten Schlüssel des RSA-Kryptosystems aus dem öffentlichen Schlüssel berechnen, was jegliche Sicherheit des Systems zunichte macht.[33]

Der Algorithmus funktioniert jedoch nur mit Hilfe eines Quantencomputers. Ein solcher besitzt analog zu den Bits eines normalen Computers sogenannte Qubits. Eine Anzahl von n Bits kann 2^n Werte darstellen. Das selbe gilt grundsätzlich auch für Qubits, jedoch mit mehreren Besonderheiten: Ein einzelner Qubit kann mehr als die zwei Werte 0 und 1 annehmen, mehrere Qubits können miteinander verschränkt sein und auf n verschränkte Qubits können durch Quantenparallelismus unter Anwendung eines Gatters 2^n Manipulationen zeitgleich durchgeführt werden, wie später genauer erklärt wird. Ein klassisches Gatter verarbeitet binäre Eingangssignale zu einem binären Ausgangssignal. Das simpelste dürfte dabei das Nicht-Gatter sein, welches beispielsweise aus einer Bitfolge 101101 die Bitfolge 010010 macht. Für jedes Bit der beispielhaften Bitfolge muss eine eigenständige Manipulation mit dem Nicht-Gatter geschehen. In diesem Fall also 6 Operationen für die gegebenen 6 Bits.

[33] https://de.wikipedia.org/wiki/Shor-Algorithmus

Um näher erklären zu können, wie in einem Quantencomputer mehrere Zustände gleichzeitig manipuliert werden können, müssen zunächst imaginäre Zahlen, komplexe Zahlen, die Bra-Ket-Notation sowie die Quantenverschränkung erklärt werden.

3.1 Imaginäre Zahlen

In der Mathematik ergab sich in der Vergangenheit die folgende Frage: Was ist die Quadratwurzel einer negativen Zahl?

Im reellen Zahlenbereich $\mathbb{R}$ hat $\sqrt{-1}$ keine Lösung. Daraus schlussfolgerte man letztendlich, dass Zahlen nicht linear auf einer geraden anzuordnen sind, sondern auf einer Ebene mit einem sogenannten Realteil und einem Imaginärteil. Der Imaginärteil besteht dabei aus eine reellen Zahl, die mit $\sqrt{-1}$ multipliziert wird. Um dabei nicht jedes mal $\sqrt{-1}$ schreiben zu müssen, legte man folgende Schreibweise fest: $\mathrm{i} = \sqrt{-1} \Leftrightarrow \mathrm{i}^2 = -1$.[34] (Manchmal auch j statt i.)

Durch die Produktregel gilt hier als Beispiel: $\sqrt{-9} = \sqrt{9} \cdot \sqrt{-1} = 3\mathrm{i} \vee -3\mathrm{i}$.

3.2 Komplexe Zahlen

Jede imaginäre Zahl ist eine komplexe Zahl $\mathbb{C}$ mit einem Realteil $a = 0$. Eine komplexe Zahl c besteht aus einem Realteil und einem Imaginärteil und wird wie folgt definiert: $c = a + b\mathrm{i}$, mit $a = \mathrm{Re}(a + b\mathrm{i})$ und $b = \mathrm{Im}(a + b\mathrm{i})$.

Wie in Abbildung 1 im Anhang zu sehen ist, stellt man komplexe Zahlen auf einer komplexen Zahlenebene dar. Später fand man heraus, dass es noch weitere Zahlen neben den komplexen Zahlen gibt, für die nochmals weitere Dimensionen zur Darstellung nötig sind. Diese sollen jedoch keine weiter Rolle in dieser Facharbeit spielen.

Eine konjugierte komplexe Zahl zu $c = a + b\mathrm{i}$ wird zudem wie folgt beschrieben: $\bar{c} = a - b\mathrm{i}$.[35]

---

[34] https://de.wikipedia.org/wiki/Imaginäre_Zahl

[35] https://de.wikipedia.org/wiki/Konjugation_(Mathematik)

3.3 Bra-Ket-Notation

Die Bra-Ket-Notation, auch Dirac-Notation genannt, wird in der Quantenmechanik zur Beschreibung quantenmechanischer Zustände verwendet. Es gibt dabei sogenannte Bra-Ausdrücke, $\langle v|$, und Ket-Ausdrücke, $|v\rangle$. Diese Ausdrücke lassen sich auch als vertikale beziehungsweise horizontale Vektoren darstellen in einem m-dimensionalen Vektorraum, wobei für einen Vektor v in diesem gilt: $v \in \mathbb{C}^m$.

$$|v\rangle \doteq \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_m \end{pmatrix}$$

Der Ket-Ausdruck und der Vektor sind nicht als das selbe mathematische Objekt aufzufassen, was im Folgenden aber keine weitere Rolle spielt.

Der dazugehörige Bra-Ausdruck hat zum Ket-Ausdruck konjugierte Werte und wird wie folgt dargestellt: $\langle v| \doteq \begin{pmatrix} v_1^* & v_2^* & v_3^* & \dots & v_m^* \end{pmatrix}$.

Für ein Bra $\langle\phi|$ und ein Ket $|\psi\rangle$ erhält man folgende Schreibweise für das Skalarprodukt: $\langle\phi|\psi\rangle := \big(\langle\phi|\big) \cdot \big(|\psi\rangle\big)$. Man spricht von einer Anwendung des Bras auf das Ket. Für zwei komplexe Zahlen c_1 und c_2 gilt:

$$\langle\phi| \left(c_1|\psi_1\rangle + c_2|\psi_2\rangle \right) = c_1\langle\phi|\psi_1\rangle + c_2\langle\phi|\psi_2\rangle.$$[36]

3.4 Qubits

Wie zuvor angeschnitten, kann ein Qubit prinzipiell ebenfalls die Werte 0 und 1 annehmen. Jedoch kann ein einzelner Qubit noch weitere Zustände annehmen und liegt zunächst in einer sogenannten Superposition der beiden Zustände $|0\rangle$ und $|1\rangle$. Erst beim Messen des Wertes eines Qubits wird der Wert endgültig festgelegt.

Ein Qubit $|\phi\rangle$ wird deshalb wie folgt beschrieben in Bra-Ket-Notation:

$$|\phi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$$

[36] https://de.wikipedia.org/wiki/Dirac-Notation

In dieser Form stellt $|\alpha_0|^2$ die Wahrscheinlichkeit, dass der Zustand $|0\rangle$ gemessen wird und $|\alpha_1|^2$ die Wahrscheinlichkeit, dass der Zustand $|1\rangle$ gemessen wird, dar. Somit ergibt sich: $|\alpha_0|^2 + |\alpha_1|^2 = 1$.

In Abbildung 2 im Anhang findet sich eine Veranschaulichung, wie diese Superposition durch Kugelkoordinaten realisiert ist. Der Zustand des Qubits wird in diesem Fall konkreter wie folgt beschrieben:

$$|\phi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$$[37]

Für die Beschreibungen mehrerer Qubits gibt es folgende Schreibweisen:

$$|1\rangle|0\rangle = |10\rangle = |1\rangle \otimes |0\rangle = |2\rangle$$

Ein Register eines klassischen Computers ist als die gegebene Bitfolge zu verstehen. Ein Quantenregister hingegen beschreibt die Superposition aller möglich Kombinationen einzelner Qubits. Für ein 2-Qubit Quantenregister R gilt:

$$R = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$$

Dabei gilt auch hier: $|\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 = 1$.[38]

3.5 Quantenverschränkung

Die Verschränkung von Quantenobjekten bedeutet, dass ihr zusammengesetztes Ganzes zwar wohldefiniert werden kann, dies aber nicht für die einzelnen Teile des Systems der Fall ist. Teile des Systems sind dadurch unteilbar oder eben: verschränkt.

Ein unverschränkter Zustand, der in eine Form, die alle Qubits einzeln beschreibt, überführt werden kann, sieht beispielhaft wie folgt aus:

$$\frac{1}{\sqrt{2}}(|10\rangle + |11\rangle) = |1\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

---

[37] https://www.tfp.kit.edu/downloads/lehre_2013_ss/aaa_HSS13_Grundlagen.pdf, Seite 7

[38] https://itp.tugraz.at/LV/arrigoni/projektpraktikum/gasser/ver.1.todel/Bachelorarbeit.pdf, Seite 4

Ein verschränkter Zustand, der nicht anders ausgedrückt werden kann, wäre hingegen zum Beispiel der folgende:

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Wie sich sehen lässt, lassen sich die einzelnen Qubits nicht genauer Beschreiben, was zur Folge hat, dass durch Messung an einem der beiden Qubits der Zustand beider Qubits festgelegt wird.[39] [40]

3.6 Quantenparallelismus

Wie zuvor angesprochen, ist es in einem Quantencomputer möglich mehrere verschränkte Zustände gleichzeitig zu manipulieren. Ein einfaches Beispiel für die Anwendung des Nicht-Gatters auf einen Zustand:

$|\phi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$ sieht wie folgt aus:

$|\phi'\rangle = \alpha_0|11\rangle + \alpha_1|10\rangle + \alpha_2|01\rangle + \alpha_3|00\rangle.$

Für n verschränkte Qubits können, kurz gefasst, 2^n Manipulationen durch ein einziges Gatter durchgeführt werden. Dies liefert einen entscheidenden Vorteil gegenüber klassischen Computern.[41]

3.7 Mathematik des Shor-Algorithmus

„Die grundlegende Idee ist, dass man die Faktorisierung auf die Bestimmung der Ordnung zurückführen kann. Diese Bestimmung lässt sich mit Hilfe der Quanten-Fouriertransformation effektiv durchführen. Man teilt den Algorithmus deshalb häufig in einen klassischen Teil zur Reduzierung des Problems und einen Quantenteil, der das Restproblem effizient löst.“[42]

Im folgenden sei nur der klassische Teil des Shor-Algorithmus im Detail erklärt, da der Quantenteil die Quanten-Fouriertransformation beinhaltet, welche auf

[39] https://itp.tugraz.at/LV/arrigoni/projektpraktikum/gasser/ver.1.todel/Bachelorarbeit.pdf, Seite 5

[40] https://de.wikipedia.org/wiki/Quantenverschränkung

[41] https://de.wikipedia.org/wiki/Quantenparallelismus

[42] https://de.wikipedia.org/wiki/Shor-Algorithmus#Ablauf

der diskreten Fouriertransformation basiert. All dies zu erklären würde den Rahmen dieser Facharbeit sprengen, weshalb nur das Ergebnis des Quantenteils beschrieben werden soll.
Um nun aber zunächst auf das Finden nichttrivialer Teiler einer Zahl zurückzukommen, muss die Ordnung einer Zahl definiert werden.

3.7.1 Die Ordnung einer Zahl

„Sei G eine (multiplikative) endliche Gruppe mit neutralem Element 1. […] Die Ordnung eines Elements $a \in G$ ist [definiert als:]
$\mathrm{ord}_G(a) := \min\{i \in \mathbb{N} \mid a^i = 1\}$."[43]

3.7.2 Die Ordnung modulo m

Seien a und m zwei teilerfremde natürliche Zahlen, so gibt es eine Zahl $i \in \mathbb{N}$, für die gilt: $a^i \equiv 1 \pmod{m}$. Die Ordnung von a ist dann i an dieser Stelle.[44]

3.7.3 Klassischer Teil

Es sei n eine Zahl, deren nichttrivialen Teiler bestimmt werden sollen. Der Shor-Algorithmus kann dann in folgende Schritte aufgeteilt werden, die in Abbildung 3 im Anhang noch einmal anschaulich zu sehen sind:

1. Eine Zahl a bestimmen für die gilt: $1 < a < n$.
2. Den $\mathrm{ggT}(a, n)$ bestimmen. Falls $\mathrm{ggT}(a, n) \neq 1$ gilt wurde ein nichttrivialer Teiler gefunden und der Algorithmus kann beendet werden.
3. Durch den Quantenteil die Ordnung r von a in $\mathbb{Z}_n^*$ bestimmen.
4. Zum ersten Schritt zurückgehen, im Falle, dass r ungerade ist oder gilt: $a^{r/2} \equiv -1 \pmod{n}$
5. $\mathrm{ggT}(n, a^{r/2} - 1)$ und/oder $\mathrm{ggT}(n, a^{r/2} + 1)$ als Lösung zurückgeben.[45]

---

[43] http://www.crypto.ruhr-uni-bochum.de/imperia/md/content/may/dima08/16_neben.pdf, Seite 1

[44] https://www.spektrum.de/lexikon/mathematik/ordnung-modulo-m/7554

[45] https://de.wikipedia.org/wiki/Shor-Algorithmus#Klassischer_Teil

Für eine genauere Erklärung, wie durch die Ordnung r einer Zahl a die Teiler einer Zahl n bestimmt werden können, möchte ich auf Seite 3 des folgenden Dokuments verweisen, auf welcher sich eine leicht verständliche Ausführung des Ganzen findet: http://page.mi.fu-berlin.de/alt/vorlesungen/sem02/shors_algorithm.pdf. Von einer eigenen Ausführung sehe ich an dieser Stelle ab, da ich es selber nicht besser erklären könnte und erneut ein gewisser Rahmen für diese Facharbeit einzuhalten ist.

3.7.4 Quantenteil

Der Quantenteil des Shor-Algorithmus berechnet mit Hilfe der Quanten-Fouriertransformation die Ordnung r von a in $\mathbb{Z}_n^*$. Dabei sei zur Veranschaulichung eine Funktion f definiert: $f(k) = a^k \bmod n$. In Abbildung 4 im Anhang befindet sich ein entsprechender Graph mit den Werten $a = 7$ und $n = 15$. Wie sich leicht erkennen lässt, wiederholen sich die Werte der Balken periodisch. Die Periode entspricht der Ordnung: $r = 4$.
Die Quanten-Fouriertransformation kann diese Periode aufgrund des Quantenparallelismus effizient bestimmen.

3.8 Notwendigkeit eines Quantencomputers

Da genannter Parallelismus nicht auf normalen Computern genutzt werden kann und eine Simulation von einer Vielzahl an Qubits enorme Rechenleistungen erfordern würden, kann der Shor-Algorithmus nur auf einem Quantencomputer effizient angewendet werden. Da diese jedoch bisher aus einer unzureichenden Anzahl an Qubits bestehen, kann der Algorithmus praktisch noch nicht angewandt werden.
Die größte mit dem Shor-Algorithmus faktorisierte Zahl ist im Augenblick deshalb nur die recht kleine Zahl 21.[46]

[46] https://www.tfp.kit.edu/downloads/lehre_2013_ss/aaa_HSS13_Shor.pdf, Seite 16

4 Schlusswort

Festhalten lässt sich, dass das RSA-Kryptosystem zu den aktuellsten Verschlüsselungsverfahren gehört und grundsätzlich eine hohe Sicherheit bietet. Neben dem Shor-Algorithmus gibt es zwar noch weitere Gefahren für die Sicherheit einer mit dem RSA-Kryptosystems verschlüsselten Nachricht, jedoch lassen sich diese durch kleinere Anpassung, des in dieser Facharbeit vorgestellten grundlegenden Verfahrens, ausräumen. Weitere Angriffsmöglichkeiten und entsprechende Lösungen, um diese auszuräumen können folglich als weitere interessante Themen aufgeführt werden.

Zu dem Shor-Algorithmus lässt sich anmerken, dass dieser das Potenzial besitzt in der Zukunft die Sicherheit des RSA-Kryptosystem zunichte zu machen. Im Augenblick ist dies jedoch auszuschließen, da Quantencomputer dafür bisher nicht ansatzweise genügend Qubits besitzen.

Ebenfalls interessante Themen für die Zukunft sind von daher, in Bezug auf den Shor-Algorithmus, die genauere Betrachtung der Quanten-Fouriertransformation, die Realisierungsmöglichkeiten durch die Entwicklung größerer Quantencomputer und die Frage, wie Nachrichten in der Zukunft stattdessen sicher verschlüsselt werden können.

5 Anhang

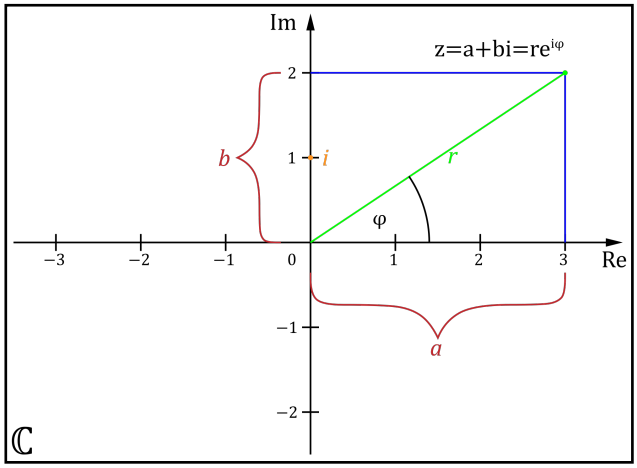

Abbildung 1[47]

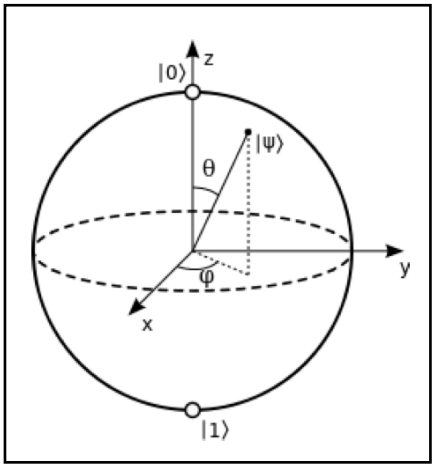

Abbildung 2[48]

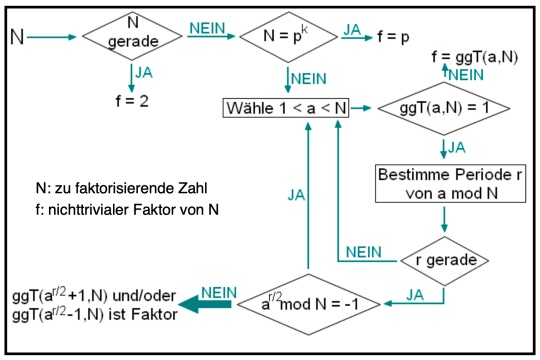

Abbildung 3[49]

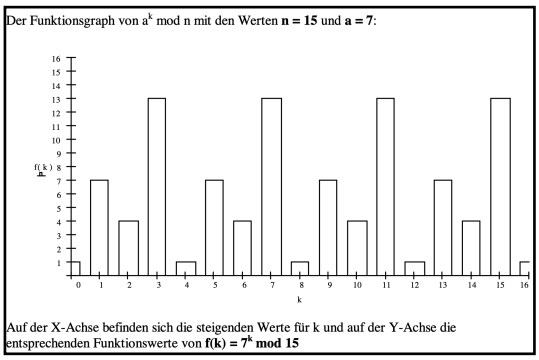

Abbildung 4[50]

RSA Implementation in Ruby:

```
class Integer
   def prim?
      p = self.abs()
      return true if p == 2
      d = p - 1
      s = 0
      while d % 2 == 0
         d /= 2
         s += 1
      end
      20.times do
         a = 2 + rand(p - 4)
         x = modulare_exponentiation(a, d, p)
```

[47] https://commons.wikimedia.org/wiki/File:Gaußsche_Zahlenebene.svg

[48] https://www.tfp.kit.edu/downloads/lehre_2013_ss/aaa_HSS13_Grundlagen.pdf, Seite 7

[49] https://www.tfp.kit.edu/downloads/lehre_2013_ss/aaa_HSS13_Shor.pdf, Seite 10

[50] http://page.mi.fu-berlin.de/alt/vorlesungen/sem02/shors_algorithm.pdf, Seite 4

```
            if not (x == 1 || x == p - 1)
               (s-1).times do
                  x = modulare_exponentiation(x, 2, p)
                  break if x == p - 1
               end
               return false if x != p - 1
            end
         end
         return true
      end

      def text!
         text=""
         i=self
         while i>0
            text = (i&0xff).chr + text
            i >>= 8
         end
         return text
      end
   end

   class String
      def zahl!
         zahl = 0
         self.each_byte do |byte|
            zahl = zahl*256+byte
         end
         return zahl
      end
   end

   def modulare_exponentiation(basis, exponent, modul)
      ergebnis = 1
      while exponent > 0
         if exponent % 2 == 1
            ergebnis = (ergebnis * basis) % modul
         end
         basis = (basis * basis) % modul
         exponent >>= 1
      end
      return ergebnis
   end

   def zufällige_zahl(länge)
      mitte = ""
      (länge-2).times do
         mitte += rand(2).to_s
      end
```

```
    binär_text = "1" + mitte + "1"
    return binär_text.to_i(2)
end

def zufällige_primzahl(länge)
    while true
        zahl = zufällige_zahl(länge)
        if zahl.prim?
            return zahl
        end
    end
end

def erweiterter_euklidischer_algorithmus(a, b)
    if a % b == 0
        return [0,1]
    end
    t,s = erweiterter_euklidischer_algorithmus(b, a % b)
    return [s, t-s*(a / b)]
end

def berechne_d(p,q,e)
    phi = (p-1)*(q-1)
    s,t = erweiterter_euklidischer_algorithmus(e,phi)
    if s < 0
        s += phi
    end
    return s
end

def verschlüsseln
    puts "Gebe die Bitlänge der Primzahlen ein (übliche Werte: 256, 512, 1024):"
    i = gets.to_i
    p = zufällige_primzahl(i)
    q = zufällige_primzahl(i)
    while p==q
        q = zufällige_primzahl(i)
    end

    n = p*q
    e = 65537

    d = berechne_d(p,q,e)

    puts "Primzahl p:\n#{p}"
    puts "Primzahl q:\n#{q}"
    puts "Exponent:\n#{e}"
    puts "Öffentlicher Schlüssel N:\n#{n}"
    puts "Privater Schlüssel d:\n#{d}"
```

```
    puts "Gebe die zu verschlüsselnde Nachricht ein:"
    m = gets.chomp.zahl!
    if m.to_s(2).bytesize > n.to_s(2).bytesize
        puts "Nachricht ist zu lang. (Versuche gegebenenfalls eine größere Bitlän-
ge der Primzahlen oder Teile die Nachricht in kleinere Blöcke auf)"
        return
    end

    c = modulare_exponentiation(m,e,n)
    puts "Nachricht als Zahl:\n#{m}"
    puts "Verschlüsselt als Zahl:\n#{c}"

    puts "Verschlüsselte Nachricht: "
    puts c.text!
end

def entschlüsseln
    puts "Gebe den öffentlichen Schlüssel N ein:"
    n = gets.to_i
    puts "Gebe den privaten Schlüssel d ein:"
    d = gets.to_i
    puts "Gebe die zu entschlüsselnde Nachricht als Zahl ein:"
    c = gets.to_i
    a = modulare_exponentiation(c,d,n)
    puts "Entschlüsselte Nachricht:"
    puts a.text!
end

nicht_beenden = true
while nicht_beenden
    puts "Gebe die \"1\" ein zum Verschlüsseln, die \"2\" zum Entschlüsseln oder
die \"3\" zum Beenden:"
    case gets.to_i
    when 1
        verschlüsseln
    when 2
        entschlüsseln
    when 3
        nicht_beenden = false
    end
    puts "\n\n"
end
```

Unter folgendem Link findet sich eine kommentierte Version des Codes:

https://github.com/N-I-N-0/Facharbeit-Q1-Kryptographie/blob/master/rsa%20-mit%20Kommentaren.rb

6 Quellen

Quellen

1. https://de.wikipedia.org/wiki/Kryptographie#Terminologie (04.02.2020 17:33)
2. https://de.wikipedia.org/wiki/Kryptographie#Abgrenzung_zur_Steganographie (04.02.2020 17:33)
3. https://de.wikipedia.org/wiki/Kryptographie#Ziele_der_Kryptographie (04.02.2020 17:33)
4. https://de.wikipedia.org/wiki/Kryptographie#Methoden_der_Kryptographie (04.02.2020 17:33)
5. https://www.kryptowissen.de/symmetrische-verschluesselung.html (04.02.2020 19:10)
6. http://www.mathe.tu-freiberg.de/~hebisch/cafe/kryptographie/skytale.html (05.02.2020 17:11)
7. https://de.wikipedia.org/wiki/Caesar-Verschlüsselung (05.02.2020 17:54)
8. https://de.wikipedia.org/wiki/Enigma_(Maschine)#Prinzip (06.02.2020 18:03)
9. https://de.wikipedia.org/wiki/Einwegfunktion#Einwegfunktionen_mit_Falltür_(Trapdoor-Einwegfunktionen) (06.04.2020 14:24)
10. https://www.kryptowissen.de/asymmetrische-verschluesselung.html (04.02.2020 19:41)
11. https://www.zum.de/Faecher/Inf/RP/infschul/kr_rsa.html (10.04.2020)
12. https://de.wikipedia.org/wiki/RSA-Kryptosystem#Geschichte (06.04.2020 14:10)
13. https://de.wikipedia.org/wiki/RSA-Kryptosystem#Einwegfunktionen (06.04.2020 14:10)
14. Yimin Ge, Die Mathematik von RSA (August 2005), Seite 4-5; PDF auf Anfrage verfügbar
15. https://mathepedia.de/Existenz_und_Eindeutigkeit.html (24.02.2020 16:05)
16. Yimin Ge, Die Mathematik von RSA (August 2005), Seite 6; PDF auf Anfrage verfügbar
17. Yimin Ge, Die Mathematik von RSA (August 2005), Seite 6-7; PDF auf Anfrage verfügbar

18. https://de.wikipedia.org/wiki/Erweiterter_euklidischer_Algorithmus (06.04.2020 16:27)
19. https://de.wikipedia.org/wiki/Eulersche_Phi-Funktion (07.04.2020 13:58)
20. https://mathepedia.de/Eulersche_Phi-Funktion.html (07.04.2020 15:02)
21. https://repositorium.uni-muenster.de/document/miami/92fac8b4-ce4d-4643-8c83-f8826518e856/bachelor_mandrysch_2017.pdf, Seite 8 (07.04.2020 17:41)
22. https://www.math.tu-dresden.de/~ganter/inf0708/folien/inf07-Modulo.pdf, Seite 9 (07.04.2020 19:07)
23. https://www.inf.hs-flensburg.de/lang/krypto/grund/gruppezn.htm (07.04.2020 18:05)
24. https://www.inf.hs-flensburg.de/lang/krypto/grund/inverses-element.htm (08.04.2020 15:56)
25. https://www.inf.hs-flensburg.de/lang/krypto/grund/fermat-euler.htm, Satz von Euler (09.04.2020 13:32)
26. https://www.youtube.com/watch?v=DU082wcr40A (09.04.2020 16:10)
27. https://www.inf.hs-flensburg.de/lang/krypto/grund/fermat-euler.htm, Modifizierter Satz von Euler (09.04.2020 13:32)
28. https://www.inf.hs-flensburg.de/lang/krypto/algo/primtest.htm (10.04.2020 11:02)
29. https://www.inf.hs-flensburg.de/lang/krypto/algo/modexp2.htm, Program (Python Vorlage) (12.04.2020 16:40)
30. https://de.wikipedia.org/wiki/RSA-Kryptosystem#Erzeugung_des_öffentlichen_und_privaten_Schlüssels (15.04.2020 14:51)
31. https://de.wikipedia.org/wiki/RSA-Kryptosystem#Beziehung_zwischen_RSA_und_dem_Faktorisierungsproblem (16.04.2020 17:39)
32. https://www.inf.hs-flensburg.de/lang/krypto/protokolle/rsa.htm, Entschlüsselung (15.04.2020 15:35)
33. https://de.wikipedia.org/wiki/Shor-Algorithmus (18.04.2020 14:56)
34. https://de.wikipedia.org/wiki/Imaginäre_Zahl (21.04.2020 15:34)
35. https://de.wikipedia.org/wiki/Konjugation_(Mathematik) (21.04.2020 16:28)
36. https://de.wikipedia.org/wiki/Dirac-Notation (19.04.2020 15:11)

37. https://www.tfp.kit.edu/downloads/lehre_2013_ss/aaa_HSS13_Grundlagen.pdf, Seite 7 (24.04.2020 16:48)
38. https://itp.tugraz.at/LV/arrigoni/projektpraktikum/gasser/ver.1.todel/Bachelorarbeit.pdf, Seite 4 (24.04.2020 16:57)
39. https://itp.tugraz.at/LV/arrigoni/projektpraktikum/gasser/ver.1.todel/Bachelorarbeit.pdf, Seite 5 (24.04.2020 16:57)
40. https://de.wikipedia.org/wiki/Quantenverschränkung (24.04.2020 17:43)
41. https://de.wikipedia.org/wiki/Quantenparallelismus (24.04.2020 18:19)
42. https://de.wikipedia.org/wiki/Shor-Algorithmus#Ablauf (18.04.2020 14:56)
43. http://www.crypto.ruhr-uni-bochum.de/imperia/md/content/may/dima08/16_-neben.pdf, Seite 1 (25.04.2020 18:52)
44. https://www.spektrum.de/lexikon/mathematik/ordnung-modulo-m/7554 (25.04.2020 19:01)
45. https://de.wikipedia.org/wiki/Shor-Algorithmus#Klassischer_Teil (18.04.2020 14:56)
46. https://www.tfp.kit.edu/downloads/lehre_2013_ss/aaa_HSS13_Shor.pdf, Seite 16 (28.04.2020 12:22)

Abbildungen

47. https://commons.wikimedia.org/wiki/File:Gaußsche_Zahlenebene.svg (21.04.2020 15:38)
48. https://www.tfp.kit.edu/downloads/lehre_2013_ss/aaa_HSS13_Grundlagen.pdf, Seite 7 (24.04.2020 16:48)
49. https://www.tfp.kit.edu/downloads/lehre_2013_ss/aaa_HSS13_Shor.pdf, Seite 10 (25.04.2020 19:14)
50. http://page.mi.fu-berlin.de/alt/vorlesungen/sem02/shors_algorithm.pdf, Seite 4 (26.04.2020 14:47)

Verweis

51. http://page.mi.fu-berlin.de/alt/vorlesungen/sem02/shors_algorithm.pdf, Seite 3 (26.04.2020 14:47) [siehe Abschnitt 3.7.3]