



Wie sicher ist das RSA-Verfahren?

Facharbeit im Fach Mathematik
Betreuer: Herr Akeston

Len-Marvin Adler
Hildegard-von-Bingen Gymnasium
Leybergstraße 1
50939 Köln

12. März 2021

Inhaltsverzeichnis

1	Zahlentheorie	3
1.1	Einführung	3
1.2	Teilbarkeitsrelation	3
1.3	Teilmengen	4
1.4	Lineare diophantische Gleichungen	5
1.5	ggT und (Erweiterter) euklidischer Algorithmus	7
1.6	Satz vom kleinsten Teiler	10
2	Primzahlen	10
2.1	Hauptsatz der elementaren Zahlentheorie	10
2.2	Eulersche φ -Funktion	10
3	Restklassen/ Kongruenzen	11
3.1	Satz von Euler	12
4	Big O Notation und Effizienz von Algorithmen	13
5	Verschlüsselung	15
5.1	RSA	15
5.1.1	Verschlüsselung	17
5.1.2	Entschlüsselung	18
5.1.3	Beispielrechnung	18
5.1.4	Implementierung	19
5.2	RSA in der Zukunft	20
	Literaturverzeichnis	22
	Anhang	24

In der heutigen, digitalen Welt ist die Verschlüsselung von Daten essenziell. Aber wie funktioniert diese? Das sogenannte „RSA-Verfahren“ ist eine gängige Verschlüsselungsart.

Im Folgenden wird dieser Algorithmus erklärt und die Frage gestellt, ob diese Verschlüsselung sicher ist, jedoch muss man sich erstmal der Zahlentheorie bewusst werden.

1 Zahlentheorie

1.1 Einführung

Die Mathematik setzt sich aus vielen verschiedenen Teilbereichen zusammen. Beschäftigt sich der Bereich der Analysis mit Funktionen und ihren Eigenschaften, die Stochastik mit Wahrscheinlichkeitsrechnungen, so beschäftigt sich die Zahlenlehre, auch elementare Zahlentheorie genannt, mit den ganzen Zahlen \mathbb{Z} , ihren Eigenschaften und den Zusammenhängen zwischen ihnen.

Die einfachsten Zusammenhänge zwischen den ganzen Zahlen sind Addition, Subtraktion, Multiplikation und Division.

So haben etwa die Zahlen

$$a = 1 \text{ und } b = 3$$

die Eigenschaft, dass

$$a + q = b, \text{ mit } q = 2$$

gilt.

Ferner ergibt das Produkt aus b und q eine weitere Zahl

$$d = 6.$$

Umgekehrt entspricht der Quotient aus d und q der Zahl b .

Verallgemeinert kann man also die erste Definition der Zahlentheorie folgern.

1.2 Teilbarkeitsrelation

Definition. *Gibt es eine beliebige Zahl a , durch die man eine weitere Zahl b teilt, wobei $a, b \in \mathbb{Z}$ sind, dann sagt man, dass*

$$b \mid a \text{ (man spricht } b \text{ teilt } a),$$

ebenso gilt, aufgrund des Kommutativgesetzes, dass

$$q \mid a,$$

wenn eine Zahl $q \in \mathbb{Z}$ existiert, mit $q \cdot b = a$ [12].

Ferner kann man die Regeln der Teilbarkeit erweitern.
Wenn

$$a \mid b \wedge b \mid q,$$

so folgt, dass

$$a \mid q,$$

man spricht auch, die Teilbarkeitsrelation ist transitiv [12, 7].

Ferner, wenn

$$a \mid b \text{ und } a \mid q,$$

dann gilt

$$a \mid (b - q)$$

Beweis. Wenn

$$a \mid b,$$

dann ist $b = k \cdot a$, mit $k \in \mathbb{Z}$.

Desweiteren, wenn

$$a \mid q,$$

dann ist $q = n \cdot a$, mit $n \in \mathbb{Z}$ und

$$b - q = k \cdot a - n \cdot a = (k - n) \cdot a,$$

und damit ist $b - q$ weiterhin ein Vielfaches von a [12]. ■

Genau deswegen gilt auch:

$$a \mid (b + q),$$

wenn

$$a \mid b \text{ und } a \mid q.$$

1.3 Teilermengen

Nun weiß man, dass $a \mid b$, allerdings ist es manchmal hilfreich die konkreten Teiler zu haben.

Die Teilermenge von n , nennen wir $T(n)$, wobei $n \in \mathbb{N}$.

Die Zahlen, durch die n teilbar ist, sind $\{x \in \mathbb{N}, x \mid n\}$.

Als Beispiel können wir $n = 5$ nehmen, so ist die Teilermenge $T(5) = \{1, 5\}$.

Bei $n = 12$ wäre die Teilermenge $T(12) = \{1, 2, 3, 4, 6, 12\}$.

Sind die Teiler, wie in $T(12)$ nach Größe sortiert, ergibt der erste Teiler multipliziert mit dem letzten die Zahl n . Genauso, wenn man den zweiten und den vorletzten Teiler miteinander multipliziert und so weiter.

Man nennt diese dann auch *komplementäre Teiler*.

Mit dem Wissen, dass $a \in \mathbb{N}$ können wir nun einen weiteren Satz äußern.

Satz 1. Jedes $n \in \mathbb{N}$ hat höchstens n positive Teiler.

Beweis. Wenn ein $b \in \mathbb{N}$ existiert, welches n teilt, so gibt es, laut **Definition 1**, ein $q \in \mathbb{N}$, wobei $q \cdot b = n$. Da $q \in \mathbb{N}$ gilt, dass $q \geq 1$. Vergleichen wir nun $q \cdot b = n$ und $1 \cdot b = b$, so fällt auf, da eben genanntes gilt, dass $n = q \cdot b \geq 1 \cdot b = b$. Weil b ebenfalls $\in \mathbb{N}$, ist $b \geq 1$. Daraus folgt unmittelbar, dass $1 \geq b \geq n$ und es höchstens n positive Teiler b von a gibt [12, 24]. ■

Außerdem kann man sagen, dass der größte Teiler $a \in \mathbb{N} \setminus \{n\}$ in der Teilmenge $T(n)$ mit $n \in \mathbb{N} \setminus \{1\} \leq n/2$ ist.

Als Beispiel: $n = 60$:

$$T(60) = \{1, 2, 3, 4, 5, 6, 10, 12, 15, \mathbf{30}, 60\}.$$

Genauso wenn wir $n = 36$:

$$T(36) = \{1, 2, 3, 4, 6, 9, 12, \mathbf{18}, 36\}.$$

Oder auch wenn $n = 49$:

$$T(49) = \{1, \mathbf{7}, 49\} [12].$$

Besteht die Teilermenge einer Zahl $n \in \mathbb{N}$ aus mehr als zwei Zahlen, so nennt man n eine zusammengesetzte Zahl.

Allerdings gibt es auch Zahlen, zum Beispiel $k \in \mathbb{N}$, deren Teilermenge aus nur zwei Zahlen, der eins und sich selber, k , besteht, also

$$T(k) = \{1, k\}.$$

Diese besondere Zahlen, nennen wir nun Primzahlen. Die Menge der Primzahlen bezeichnen wir als \mathbb{P} [12].

1.4 Lineare diophantische Gleichungen

In der Mathematik gibt es verschiedene Arten von Gleichungen, als Beispiel lineare Gleichungen, in denen die Variable x nur als erste Potenz vorkommt. Die einfachste lineare Gleichung sieht dann folgendermaßen aus:

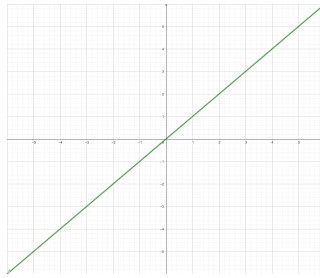


Abbildung 1: $f(x) = x$ im Intervall $I\{-6; 6\}$

Natürlich kann man nun die Potenz der Variable immer weiter erhöhen. Bei zweiter Potenz kann die Funktion in etwa so aussehen:

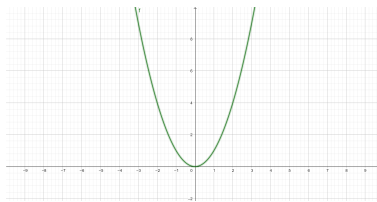


Abbildung 2: $f(x) = x^2$ im Intervall $I\{-10; 10\}$

Nun kann es aber auch sein, dass wir eine Gleichung mit zwei Variablen x, y haben. Wir lassen beide Variablen in der ersten Potenz und stellen eine Gleichung auf:

$$a \cdot x + b \cdot y = c, \text{ mit } a, b, c \in \mathbb{Z}$$

Diese Gleichung, auch lineare diophantische Gleichung, nach dem griechischen Mathematiker Diophantos von Alexandria, genannt, kann mehr als eine Lösung haben. Da x und y variabel wählbar sind, kann man beispielsweise immer x um 1 erhöhen, während man y um 1 verringert. Folglich gibt es unendlich viele Lösungsmengen [22]. Es kann aber auch sein, da $x, y \in \mathbb{Z}$, dass die Gleichung mit keinem x und y erfüllbar ist.

David Hilbert, deutscher Mathematiker, der Anfang des 20. Jahrhunderts lebte und für das „Hilbert'sche Hotel“, eine Veranschaulichung der Unendlichkeit, bekannt wurde, beschrieb als Zehntes seiner 23 Hilbertschen Probleme, die er im Jahr 1900 veröffentlicht hat, dass zu beweisen sei, ob es einen Algorithmus gibt, der bestimmt, ob eine Diophantische Gleichung polynomialer Form, lösbar ist oder nicht. Im Jahr 1970 bewies Yuri Matiyasevich, ein russischer Mathematiker, dass es eben keinen solchen Algorithmus gibt [11].

Dennoch kann man eine lineare diophantische Gleichung vom Typ:

$$a \cdot x + b \cdot y = c,$$

lösen, wenn c einem Vielfachen des $\text{ggT}(a, b)$ entspricht.

Hat man also eine solche Gleichung

$$a \cdot x + b \cdot y = k \cdot \text{ggT}(a, b), \text{ für } k \in \mathbb{N}$$

gegeben, ist sie lösbar [22, 2]. Aber anstatt nun einfach zu erraten, für welche x und y die Gleichung gilt, existieren auch verschiedene Errechnungsmethoden der beiden Variablen.

1.5 ggT und (Erweiterter) euklidischer Algorithmus

Der größte gemeinsame Teiler (oft abgekürzt als ggT) ist eine Zahl $\in \mathbb{N} \setminus \{0\}$. Hat man zwei Zahlen $a, b \in \mathbb{Z}$, so ist der ggT schlicht die Zahl, die sowohl a , als auch b teilt. Nennen wir die Teilermenge von a : $T(a)$ und die von b : $T(b)$, so ist der $\text{ggT}(a, b)$ eben die größte Zahl aus $T(a) \cap T(b)$.

Um genau diesen rechnerisch zu ermitteln gibt es unterschiedliche Methoden und Algorithmen.

Einer davon ist der „euklidische Algorithmus“.

Wir haben zwei Zahlen $a, b \in \mathbb{Z}$ und wollen von diesen den ggT berechnen. Zuerst erstellen wir eine Tabelle mit vier Spalten.

a	b	q	r

Nun will man eine Zeile hinzufügen. Das a und b entsprechen den beiden Zahlen von denen wir den ggT wissen wollen. q gibt an, wie oft b ganzzahlig in a passt. Und r ist der ganzzahlige Rest, welcher davon bis a fehlt.

Mathematisch gesehen, muss q so groß wie möglich gewählt werden, jedoch soll $q \cdot b \leq a$ gelten.

r muss dann $a - q \cdot b$ entsprechen.

Nehmen wir als Beispiel $a = 128$ und $b = 34$, so sieht die erste Zeile der Tabelle folgendermaßen aus:

a	b	q	r
128	34	3	26

Für die nächste Zeile greift der Algorithmus.

Das neue a entspricht dem alten b und das neue b entspricht dem alten r .

Mit „alt“ ist die vorherige Zeile gemeint. q und r arbeiten immer nur mit a und b derselben Zeile. Die zweite Zeile des Beispiels wird dann so aussehen:

a	b	q	r
128	34	3	26
34	26	1	8

Der Algorithmus wird nun immer wieder angewandt, bis $r = 0$.

Sollte $r = 0$, so entspricht der $\text{ggT}(a, b)$ dem b in der selben Zeile.

a	b	q	r
128	34	3	26
34	26	1	8
26	8	3	2

Und weiter:

a	b	q	r
128	34	3	26
34	26	1	8
26	8	3	2
8	2	4	0

r entspricht nun in der letzten Zeile 0 und $b = 2$, weshalb der $ggT(128, 34) = 2$ ist [21, 19, 20].

Haben wir als diophantische Gleichung folgendes:

$$128 \cdot x + 34 \cdot y = 2$$

und wollen nun die Glieder x und y bestimmen, dann wissen wir ersteinmal, dass dies möglich ist, da wir eben berechnet haben, dass $2 = ggT(128, 34)$. Um dies nun für x und y zu berechnen, wenden wir den „erweiterten euklidischen Algorithmus“ an.

Hierfür erweitern wir die Tabelle von eben um zwei Spalten, x und y .

a	b	q	r	x	y
128	34	3	26		
34	26	1	8		
26	8	3	2		
8	2	4	0		

Die neuen Spalten geht man nun von unten nach oben durch. In der ersten Spalte wird das $x = 0$ und das $y = 1$ gesetzt.

a	b	q	r	x	y
128	34	3	26		
34	26	1	8		
26	8	3	2		
8	2	4	0	0	1

Der erweiterte euklidische Algorithmus funktioniert nun folgendermaßen:

Das neue x , also eine Spalte über dem bereits gegebenen x , entspricht dem alten y . Und das neue y entspricht dem q in derselben Zeile wie das neue x , multipliziert mit dem alten y , und dies wird vom alten x abgezogen. Also, y_1 ist in der neuen Zeile und y_0 in der alten Zeile. Genauso für q und x . Und damit ist

$$y_1 = x_0 - q_1 \cdot y_0$$

In unserem Beispiel wird also

$$x_1 = y_0 = 1$$

und

$$y_1 = 0 - 3 \cdot 1 = -3$$

Die Tabelle wird also fortgeführt:

a	b	q	r	x	y
128	34	3	26		
34	26	1	8		
26	8	3	2	1	-3
8	2	4	0	0	1

Die nächste Zeile, also eine darüber wird dann mit dem erweiterten Algorithmus berechnet. y_2 soll nun das y der neuen Zeile repräsentieren. Dasselbe gilt für q und x .

$$x_2 = y_1 = -3$$

und

$$y_2 = x_1 - q_2 \cdot y_1,$$

also:

$$y_2 = 1 - 1 \cdot -3 = 4.$$

Die Tabelle fortgeführt:

a	b	q	r	x	y
128	34	3	26		
34	26	1	8	-3	4
26	8	3	2	1	-3
8	2	4	0	0	1

Und die letzte Zeile mit $x_3 = y_2 = 4$ und $y_3 = x_2 - q_3 \cdot y_2$, also $y_3 = -3 - 3 \cdot 4 = -15$. Die vollständige Tabelle lautet:

a	b	q	r	x	y
128	34	3	26	4	-15
34	26	1	8	-3	4
26	8	3	2	1	-3
8	2	4	0	0	1

Die x - und y -Werte der obersten Zeile, sind nun zwei Werte, mit denen die anfangs aufgestellte diophantische Gleichung

$$128 \cdot x + 34 \cdot y = 2$$

funktioniert.

Wir können dies nochmal nachrechnen und tatsächlich gilt $128 \cdot 4 + 34 \cdot (-15) = 2$ [12, 22, 21, 19, 20].

1.6 Satz vom kleinsten Teiler

Haben wir eine Teilermenge $T(n)$ gegeben, so ist die kleinste Zahl

$$d \in \mathbb{P}.$$

Im Anhang unter [G] kann man den Beweis nachlesen [12].

2 Primzahlen

Eine Primzahl ist eine Zahl $\in \mathbb{N} \setminus \{1\}$, die nur durch sich selbst und die 1 teilbar ist.

Die Griechen, unter anderem Pythagoras und Euklid, kannten und arbeiteten schon mit ihnen.

So wusste Pythagoras, der für seinen Satz $a^2 + b^2 = c^2$ bekannt wurde, von den Unterschieden zwischen Primzahlen und zusammengesetzten Zahlen.

Euklid hingegen, der ungefähr im 3. Jahrhundert v. Chr. lebte, bewies die Unendlichkeit der Menge der Primzahlen, auch als „Satz des Euklid“ bekannt [12].

2.1 Hauptsatz der elementaren Zahlentheorie

Carl Friedrich Gauß (*1777, †1855), einer der bekanntesten deutschen Mathematiker, bewies in seinem Buch „Disquisitiones Arithmeticae“ aus dem Jahr 1801, dass jede Zahl $n \in \mathbb{N} \setminus \{1\}$ eine eindeutige Primfaktorzerlegung besitzt.

Als Beispiel: die Zahl 20 kann man in verschiedene Faktorenpaare aufteilen.

So gilt etwa:

$$20 = 1 \cdot 20 = 2 \cdot 10 = 4 \cdot 5$$

Lässt man allerdings als Faktoren nur Primzahlen zu, so sieht dies anders aus. Die Zahl n besitzt nur noch ein Faktorenpaar, bei der 20 gilt dann:

$$20 = 2 \cdot 2 \cdot 5$$

Die rechte Seite der Gleichung nennen wir nun Primfaktoren von 20. Natürlich kann man die Position der Faktoren unterschiedlich verschieben, aber genau weil die Multiplikation kommutativ ist, bleibt der eigentliche Term gleich [12, 14, 15, 16].

Unter [F] kann man sich den Beweis anschauen.

Der Hauptsatz der elementaren Zahlentheorie ist Grundlage vieler weiterer Forschungen.

Eine davon, wird in 5.1 vorgestellt.

2.2 Eulersche φ -Funktion

Haben wir eine Zahl $n \in \mathbb{N}$ gegeben, wollen wir die Zahlen $\leq n$ haben, die zu n teilerfremd sind.

Leonhard Euler(*1707, †1783), einer der größten Mathematiker aller Zeiten, der weitreichende Entdeckungen gemacht hat und damit die Mathematik, Physik und viele andere Bereiche stark geprägt hat, definierte eine Funktion, die uns dies wiedergibt.

Euler definierte die sogenannte „Eulersche φ Funktion“ als folgende:

$$\varphi(n) = \{a \in \mathbb{N} | 1 \leq a \leq n \wedge \text{ggT}(a, n) = 1\}$$

$\varphi(n)$ gibt uns also die Menge der Zahlen, die kleiner als n und außerdem zu n teilerfremd sind, sprich deren größter gemeinsamer Teiler mit n die 1 ist, aus.

Die Funktion ist also relativ simpel. Will man etwa $\varphi(9)$ bestimmen, kann man erstmal alle Zahlen bis 9 auflisten.

$$1, 2, 3, 4, 5, 6, 7, 8, 9$$

Wir können nun immer den ggT zwischen der jeweiligen Zahl und der 9 nehmen, sollte 1 herauskommen, so ist die jeweilige Zahl teilerfremd zur 9 und somit in $\varphi(9)$ enthalten.

Also gehen wir die eben genannten Zahlen durch und zählen wie oft der $\text{ggT} = 1$ ist.

In unserem Beispiel bekommen wir also $\varphi(9) = 6$.

Sollte n nun allerdings eine Primzahl sein, so ist $\varphi(n) = n - 1$, da n zu jeder kleineren Zahl außer sich selbst teilerfremd ist [12].

Außerdem ist die φ Funktion multiplikativ, dass heißt sollten zwei Zahlen a, b teilerfremd sein, so gilt:

$$\varphi(a \cdot b) = \varphi(a) \cdot \varphi(b).$$

Als Beispiel $a = 3$ und $b = 8$, dann ist

$$\varphi(24) = \varphi(3 \cdot 8) = \varphi(3) \cdot \varphi(8) = 2 \cdot 4 = 8.$$

Die Funktion findet vorallem Anwendung im „Satz von Euler“ (Kapitel 3.1).

3 Restklassen/ Kongruenzen

Kongruenzen sind wichtig um verschiedene Problemstellung schnell und einfach zu lösen.

Haben wir etwa einen Tag im Jahr gegeben, allerdings als z.B. den 170. Tag seit Beginn des Jahres.

Unsere Fragestellung ist nun, welcher Wochentag dies ist, so rechnen wir

$$170 \mod 7$$

Dies ist kongruent 2, auch Rest genannt. Die Rechnung funktioniert folgendermaßen:

Man nimmt die 170 und zieht $a \cdot 7$ ab, mit gewähltem a so, dass $a \cdot 7 < 170$, aber $(a + 1) \cdot 7 > 170$.

In unserem Beispiel wäre $a = 24$.

$$24 \cdot 7 = 168$$

und

$$170 - 168 = 2.$$

Man sagt

$$2 \equiv 170 \pmod{7} \text{ (sprich: „2 ist kongruent 170 modulo 7“) [12].}$$

Fängt das Jahr an einem Montag an (der 01.01. ist ein Montag), so ist der 170. Tag ein Mittwoch, denn ein Montag wird in der Restklasse 7 kongruent zur 0, der Dienstag zur 1 und schließlich der Mittwoch kongruent zur 2 sein.

Auch bei Kongruenzen gibt es einige Definitionen, so ist

Definition. eine Zahl a kongruent zu b modulo m , wenn

$$m \mid a - b$$

gilt.

Ist dies nicht der Fall, sagt man a ist inkongruent zu b modulo m [12].

Auch kann man die Uhrzeit bestimmen, sollte man beispielsweise sagen „um 35 Uhr“, so rechnet man

$$35 \pmod{24}, \text{ da ein Tag 24 Stunden hat, und somit die Restklasse 24 beträgt.}$$

Dies ergibt 11, dass heißt 35 Uhr entspricht 11 Uhr.

Wichtig bei Kongruenzen ist, dass es unendlich viele Lösungen für b gibt, da $b + a$, sollte die Kongruenz gegeben sein, nur ein Vielfaches von m ist und man somit b beliebig groß wählen kann.

Übrigens: Die englischsprachige Uhrzeit ist $\pmod{12}$ und je nachdem ob es morgens/vormittags oder nachmittags/abends ist, wird ein am/pm hinten drangehangen.

3.1 Satz von Euler

Der Satz von Euler besagt, dass für zwei Zahlen $a, m \in \mathbb{N}$ die folgende Beziehung gilt:

$$a^{\varphi(m)} \equiv 1 \pmod{m}$$

Hiermit verallgemeinert Euler den „kleinen fermatschen Satz“, welcher besagt, dass

$$a^{p-1} \equiv 1 \pmod{p}, \text{ wobei } p \in \mathbb{P}.$$

Denn $\varphi(m)$, wenn $m \in \mathbb{P}$, ist laut 2.2 $m - 1$ und entspricht dann wieder dem kleinen fermatschen Satz [12].

Sollten wir also zwei Zahlen a, m haben, mit denen der kleine fermatsche Satz gilt, so existiert das multiplikate Inverse a^{-1} . Multiplikatives Inverse heißt, dass laut dem „Satz von Euler“

$$a^{-1} \cdot a \equiv 1 \pmod{m}$$

gilt.

Dies kann man auch noch in einer linearen diophantischen Gleichung schreiben, zur Veranschaulichung nennen wir das multiplikativ Inverse a^{-1} einfach x

$$a \cdot x + b \cdot y = \text{ggT}(a, b).$$

Und wir haben eine Verbindung zu 1.4. Laut 1.5 kann man nun für x und y mit dem erweiterten euklidischen Algorithmus lösen.

Der Algorithmus liefert uns also das multiplikativ Inverse x [18, 17].

4 Big O Notation und Effizienz von Algorithmen

Ein Algorithmus ist eine bestimmte Abfolge von Schritten zur Lösung eines Problems. Der erweiterte euklidische Algorithmus in 1.5 hat zum Beispiel als Ziel, das x und y herauszubekommen.

Allerdings kann auch ein Computer einen solchen Algorithmus ausführen, meist sogar schneller als ein Mensch.

Jedoch kommt auch ein solcher irgendwann an seine Grenzen und er braucht einige Zeit um etwas zu errechnen. Mit dem heutigen Standard der Computer muss der Algorithmus hierfür sehr viele Befehle/Ausführungen haben.

Um die Laufzeit, also wie lange der Computer braucht um den Algorithmus auszuführen, zu veranschaulichen und zu mathematisieren wird nun die sogenannte „O-Notation“ (im Englischen „Big O notation“) eingeführt.

Diese O Notation wird geschrieben:

$$\mathcal{O}(n)$$

$\mathcal{O}(n)$ ist eine Klasse von Funktionen die keine richtige Einheit besitzt. Denn oftmals hängt es auch von der jeweiligen Hardware ab, wie schnell ein Algorithmus ausgeführt wird. Ein Handy wird beispielsweise länger als ein leistungsstarker Computer brauchen, um einen Algorithmus auszuführen. Daher spricht man allgemein von „Takten“, also wie viele Takte der Algorithmus ausführen muss.

n ist im Kontext der Algorithmen die Funktion, die das Wachstum der Laufzeit des Algorithmus darstellt.

Sind die Startwerte also sehr groß gewählt wird der Algorithmus viel mehr Schritte benötigen und folglich eine längere Laufzeit besitzen. n beschreibt damit als Asymptote die „schlechteste“, also längste, Laufzeit. Die O Notation gibt uns also das Wachstum des Algorithmus mit immer größer werdenden Zahlen. Natürlich kann n praktisch jede mathematische Funktion darstellen. Würde die Laufzeit des Algorithmus also quadratisch verlaufen, würde diese als

$$\mathcal{O}(n^2)$$

dargestellt werden [9].

Außerdem sei gesagt, dass ein Algorithmus „schlecht“ ist, sollte die Laufzeit mit

größer werdenden Zahlen sehr viel schneller ansteigen. n^2 wäre ein solches Beispiel. Natürlich gibt es noch schlimmere Laufzeiten beispielsweise e^n , also ein exponentielles Wachstum.

Die Big O Notation gibt uns also die generell schlechteste Laufzeit als Asymptote.

Entscheidend ist nun, dass Vorfaktoren und oder Konstanten nicht berücksichtigt werden, da es tatsächlich nur um das Wachstum $\rightarrow \infty$ geht [1].

Kommen wir nun zur Komplexität eines realen Algorithmus.

Im Anhang unter Bild [A] sehen wir eine Implementation einer Addition der Zahlen 10 und 22 in Java ¹. Die Laufzeit dieser Methode ist $\mathcal{O}(1)$, da jede Zeile nur einmal ausgeführt wird. Vor allem da es sich hier lediglich um eine Zuweisung von Variablen (eineZahl, eineWeitereZahl, additionBeiderZahlen) handelt, ist die tatsächliche Länge der Methode, sollte die Variablenzuweisung so weitergehen, nicht relevant. Man sagt, da es sich hier 4-mal um eine Zuweisung von Variablen handelt:

$$4 \cdot \mathcal{O}(1).$$

Konstanten und Vorfaktoren werden in der Big O Notation vernachlässigt.

Wenn etwa ein Wachstum eines Algorithmus linear ist, dann wird es, trotz einer Addition oder Multiplikation mit einer Konstante, ein lineares Wachstum bleiben (dies gilt natürlich nicht nur für lineares Wachstum).

Desweiteren gibt es in sehr vielen Programmiersprachen, ebenso in Java, sogenannte „Schleifen“, hierbei wird ein bestimmter Aufruf, der in der Schleife steht, wiederholt aufgerufen. Außerdem kann man in der Schleife festlegen wie oft dies wiederholt werden soll.

Unter [B] im Anhang sieht man eine solche Beispielimplementation.

Ausgegeben werden die Zahlen von 1 bis einschließlich 10.

Kommen wir nun zu einer komplexen Methode, welche die Primfaktoren einer Zahl wiedergibt. Wie wir aus 2.1 wissen, besitzt jede Zahl $n > 2$ eine solche.

Unter [C] kann man sich die Implementation anschauen. Gibt man beispielsweise die Zahl 24 ein, so gibt uns die Methode [2, 2, 2, 3] aus. Dies sind also die Primfaktoren von 24. Will man nun die Primfaktoren von größeren Zahlen bestimmen, so merkt man deutlich, dass es immer länger dauert, bis das Programm uns antwortet. Die Laufzeit ist als Diagramm unter [D] zu finden.

Auch enthalten ist die *Big O Notation* der Methode.

¹Java ist eine Objektorientierte Programmiersprache (OOP), die 1995 von James Gosling entworfen wurde „*The Java Language Specification*“, James Gosling, vol.3, 2005, Addison-Wesley.

$$\mathcal{O}(n^{1,63} \cdot \log n)$$

Mit dieser Funktion können wir nun arbeiten und setzen beliebige Zahlen für n ein, um zu bestimmen, wie viele Schritte das Programm ausführen muss. Nehmen wir etwa die Zahl 2^{2048} , welche deutlich größer als die eben genannte 24 ist. Dann können wir unter anderem dank [25] berechnen:

$$\mathcal{O}(2^{2048}) \approx 13,2151653636 \cdot 10^{579}$$

Der Prozessor eines Computers besitzt heutzutage eine Taktfrequenz von etwa 4 GHz („Giga Hertz“). Das bedeutet in einer 1 Nanosekunde kann ein solcher Prozessor 4 Takte durchführen [8].

Hiermit können wir unseren eben ermittelten Wert der Takte zur Primfaktorenfindung von 2^{2048} in die Zeiteinheiten: Sekunden, Stunden, Tage und sogar Jahre umrechnen. Wir rechnen es einfach mal in Jahre um und wir erhalten:

$$\begin{aligned} &13,2151653636 \cdot 10^{579} \text{ Takte} \cdot 0,25 \cdot 10^{-9} \frac{\text{Sekunden}}{\text{Takt}} \div 60 \text{ Sekunden} \\ &\div 60 \text{ Minuten} \div 24 \text{ Stunden} \div 365 \text{ Tage} \\ &\approx 10,476253617791 \cdot 10^{573} \text{ Jahre.} \end{aligned}$$

Diese Zeit, die der Computer also brauchen würde, um die Primfaktoren von 2^{2048} zu ermitteln, ist deutlich länger als das Universum (ca. $13,8 \cdot 10^9$ Jahre [3]) alt ist.

Man sieht also, das Finden von Primfaktoren ist auf Dauer sehr ineffizient.

5 Verschlüsselung

Sendet man über das Internet einer anderen Person eine Nachricht, so ist diese verschlüsselt. Logisch, die Nachricht ist nur an diese eine andere Person gerichtet und nicht jeder im Internet soll vom Inhalt dieser Nachricht erfahren. Genau hier kommt die Verschlüsselung ins Spiel.

Besonders oft verwendet wird das „RSA-Verfahren“, welches im Jahr 1977 entworfen wurde.

5.1 RSA

RSA steht für die Nachnamen der drei Erfinder Ron Rivest, Adi Shamir und Leonard Adleman.

Beim RSA-Verfahren handelt sich um eine asymmetrische Verschlüsselung, dass heißt der Verschlüsselungsprozess unterscheidet sich von dem der Entschlüsselung.

Es funktioniert folgendermaßen:

Man generiert zwei verschiedene öffentliche Schlüssel. Diese beiden Schlüssel heißen „öffentlich“, da sie an jede Person verschickt werden. Außerdem wird ein

privater Schlüssel erstellt, den man nur für sich behält.

Wenn man also einer Person eine Nachricht schicken will, so verschlüsselt man seine Nachricht mit den öffentlichen Schlüsseln der anderen Person, schickt allen Leuten diese verschlüsselte Nachricht. Und nur die andere Person, mit deren Schlüsseln man verschlüsselt hat, kann mit ihrem privaten Schlüssel entschlüsseln.

Konkret bedeutet dies:

Man erstellt zwei Primzahlen p_1, p_2 , welche im heutigen Standard jeweils ca. 600 bis 1000 Ziffern haben.

Nun multiplizieren wir diese und erhalten die Zahl n und es gilt

$$n = p_1 \cdot p_2$$

Dies gelingt dem Computer schnell, denn für die Grundrechenarten gibt es effiziente Algorithmen. Man spricht auch von einer „one-way function“, denn die Multiplikation ist einfach und schnell; aus dem Produkt die Faktoren herauszufinden jedoch schwer.

$$p_1 \cdot p_2 \xrightarrow{\text{einfach}} n$$

$$p_1 \cdot p_2 \xleftarrow{\text{schwer}} n$$

Dieses n ist nun einer der beiden Öffentlichen Schlüssel [10].

Nun errechnet man mithilfe von 2.2

$$\varphi(n)$$

Jede Person würde Schwierigkeiten haben dies zu berechnen, man kann schließlich nur abzählen zu welchen Zahlen n teilerfremd ist.

Man selbst kann allerdings $\varphi(n)$ bestimmen, denn man kennt ja die beiden Primzahlen p_1 und p_2 .

Übrigens behält man logischerweise alle Zahlen, außer die Öffentlichen Schlüssel, für sich.

$\varphi(n)$ kann nun dank der multiplikativen Eigenschaft der φ -Funktion in

$$\varphi(n) = \varphi(p_1) \cdot \varphi(p_2)$$

geschrieben werden, da p_1 und p_2 Primzahlen sind und

daher zueinander teilerfremd sind; desweiteren, weil n das Produkt aus p_1 und p_2 ist.

Und

$$\varphi(p_1) \cdot \varphi(p_2) = (p_1 - 1) \cdot (p_2 - 1)$$

Ferner wird nun der zweite Öffentliche Schlüssel generiert. Hierfür nehmen wir eine zufällig ausgewählte Zahl e , die zu $\varphi(n)$ teilerfremd ist.

Dabei sollte

$$1 < e < \varphi(n)$$

und nur um es nochmal zu verdeutlichen,
weil sie teilerfremd zueinander sind

$$ggT(e, \varphi(n)) = 1$$

gelten.

Unsere öffentlichen Schlüssel lauten also:

$$(e, n)$$

Uns fehlt nur noch der private Schlüssel d und wir können die Verschlüsselung starten.

Wir wollen hierfür das Inverse eines der öffentlichen Schlüssel nehmen. Dank 3.1 haben wir die folgende lineare diophantische Gleichung

$$e \cdot d + \varphi(n) \cdot y = ggT(e, \varphi(n))$$

also

$$e \cdot d + \varphi(n) \cdot y = 1$$

e und $\varphi(n)$ sind uns schon bekannt und mit 1.5 bestimmen wir nun d und y .
Uns interessiert allerdings nur das d .

Wir haben nun also alles zusammen, was wir für eine Verschlüsselung brauchen

Öffentliche Schlüssel: (e, n)

Privater Schlüssel: (d)

$Person_1$ teilt nun die Öffentlichen Schlüssel.

Nun will eine $Person_2$ der $Person_1$ eine Nachricht schicken. Diese Nachricht wandeln wir in die Zahl m um.

Wir wollen nun m verschlüsseln, sie $Person_1$ zukommen lassen, und diese wird dann von jener wieder entschlüsselt.

5.1.1 Verschlüsselung

Unsere Geheimzahl G ist

$$G \equiv m^e \pmod{n},$$

wobei e und n die öffentlichen Schlüssel der anderen Person sind.

Die verschlüsselte Nachricht wird an $Person_1$ weitergeleitet und es folgt die Entschlüsselung.

5.1.2 Entschlüsselung

Nun benutzt $Person_1$ ihren privaten Schlüssel d und den öffentlichen Schlüssel n für die Entschlüsselung.

Und die Nachricht m von $Person_2$ ist

$$m \equiv G^d \pmod{n},$$

wobei G der verschlüsselte Text ist.

Denn wenn wir die Ver- und Entschlüsselung kombinieren, erhalten wir

$$(m^e)^d \equiv m \pmod{n}$$

Was wiederum dank den Potenzregeln

$$m^{e \cdot d} \equiv m \pmod{n}$$

entspricht.

Und folglich wissen wir durch 3.1, dass

$$e \cdot d \equiv 1 \pmod{n}$$

ist, weil d das multiplikativ Inverse von e ist.

Und genau deshalb ist auch

$$(m^e)^d \equiv m \pmod{n},$$

weil logischerweise

$$m^1 \equiv m \pmod{n}$$

gilt [10].

Man spricht von einer sogenannten „trapdoor“. Im RSA-Verfahren ist der private Schlüssel d diese „trapdoor“, denn nur mit diesem kann man entschlüsseln [10]. Und man sieht, dass die Entschlüsselung funktioniert!

5.1.3 Beispielrechnung

Als Beispiel verschlüsseln wir die Zahl 6.

Wir generieren also zwei Primzahlen:

$$\text{Primzahl } p_1 = 5$$

$$\text{Primzahl } p_2 = 11$$

Nun multiplizieren wir beide Zahlen und erhalten

$$n = 5 \cdot 11 = 55$$

Desweiteren ist

$$\varphi(55) = (5 - 1) \cdot (11 - 1) = 4 \cdot 10 = 40$$

Nun bestimmen wir zufällig den zweiten Öffentlichen Schlüssel e , wobei

$$ggT(17, 40) = 1$$

gilt,

$$e = 17$$

Unsere Öffentlichen Schlüssel lauten

$$(17, 55)$$

Und der erweiterte euklidische Algorithmus liefert uns:

$$d = 33$$

Nun verschlüsseln wir als Beispiel die 21:

$$G \equiv 6^{17} \mod 55 = 41$$

Und entschlüsseln wieder:

$$m \equiv 41^{33} \mod 55 = 6$$

In der realen Welt, in der das Verfahren zum Einsatz kommt, wird allerdings nicht mit so kleinen Zahlen verschlüsselt, dies wäre zu unsicher. Deshalb rechnet man nach heutigem Standard mit Primzahlen, welche 2048 oder 4096 Bits groß sind. Dies entspricht ungefähr einer Zahl mit 600 bis 1300 Ziffern [5]. Bisher gibt es noch kein effizientes Verfahren zur Bestimmung der Primzahlen, bis dahin ist das RSA-Verfahren eines der sichersten.

5.1.4 Implementierung

Unter [E] kann man sich eine solche Implementation anschauen. Damit auch normale Wörter und Satzzeichen verschlüsselt werden können, werden die eingegebenen Wörter mithilfe der ASCII² Tafel in Zahlen umgewandelt. Mit den eingegebenen Wörter oder Satzzeichen, welche dann jeweils in ihre identischen ASCII Charaktere umgewandelt werden, wird nun einzeln mit Primzahlen der Größe 2^{4096} verschlüsselt und wieder entschlüsselt.

²ASCII steht für „American Standard Code for Information Interchange“. Es handelt sich hierbei um jeweils 8-Bit Charaktere, wobei jeder Bit einen anderen Buchstaben/Charakter darstellt. Als Beispiel ist das a : 01100001, was dann als Zahl der 97 entspricht. „ASCII Code - The extended ASCII table“, Injosoft AB, 2005, <https://www.ascii-code.com/> Lastaccessed : 8Mar2021

5.2 RSA in der Zukunft

Wie wir aus 4 wissen, gibt es für herkömmliche Computer keinen geeigneten Algorithmus zur Bestimmung der Primfaktoren.

Im Jahr 2010 lag die Primzahlgröße bei ungefähr 2^{1024} Bits [5].

Jedoch befürchten vor allem Banken, dass man die Faktorisierung dieser Zahlen in einigen Jahren berechnen könnte.

Daher wurde der internationale Standard zum Jahr 2014 auf 2^{2048} erhöht [5]. Man hat zwar nur Länge der Primzahlen verdoppelt, jedoch wird nun, da die Algorithmen nicht sehr effizient sind, die Zeit, die der Algorithmus braucht um die Faktoren zu bestimmen, exponentiell steigen.

Dadurch ist also gesichert, dass die Verschlüsselung der, zum Beispiel, Bankdaten nicht in naher Zukunft entschlüsselt werden kann.

Da die Unendlichkeit der Primzahlen bewiesen ist, könnte man praktisch die Länge der Primzahlen zur RSA-Verschlüsselung nach einigen Jahren wieder erhöhen. Dies würde sehr wahrscheinlich aber die Dauer, die der Computer braucht um zu verschlüsseln, erhöhen. Mit immer leistungstärkeren Rechnern sollte dies allerdings kein allzu großes Problem werden.

Was allerdings zum Problem werden könnte, ist die Entwicklung von sogenannten „Quantencomputern“, die auf dem Prinzip der physikalischen Quantenmechanik aufgebaut sind und sich somit signifikant von herkömmlichen Rechnern unterscheiden.

Stark vereinfacht, gibt es sogenannte „Qubits“, die das äquivalent zu den gängigen Bits darstellen sollen. Diese Qubits können allerdings, nicht so wie Bits, mehr als 2 Zustände annehmen [13].

Ein Bit kann höchstens die Zahlen 0 und 1 annehmen, wohingegen QuBits weitaus mehr Zustände, durch das „Superpositionsprinzip“ der Quantenmechanik, annehmen [13].

Hierdurch kann ein Quantencomputer herkömmliche Rechnungen deutlich schneller ausführen.

Nun hat im Jahr 1994 Peter Shor, heutiger Professor am MIT ³ einen Algorithmus für Quantencomputer entwickelt, der in polynomialer Laufzeit, das heißt, die Big O Notation beinhaltet ein Polynom, die Primfaktoren einer Zahl n berechnet [6]. Die Big O Notation des Algorithmus lautet:

$$\mathcal{O}((\log n)^2 \cdot (\log \log n) \cdot (\log \log \log n))$$

Man kann nun im Vergleich zu 4 errechnen, wie lange dieser Algorithmus für die Berechnung der Primfaktoren der Zahl $n = 2^{2048}$ braucht.

$$\mathcal{O}((\log n)^2 \cdot (\log \log n) \cdot (\log \log \log n)) \approx 472513$$

Ein Quantencomputer bräuchte also nur 472513 Takte für die Berechnung.

³MIT: Massachusetts Institute of Technology, Universität in Boston, Massachusetts

Quantencomputer sind zurzeit noch in der Produktions- und Testphase, dennoch sind kleinere Erfolge erzielt worden, so beispielsweise wurden die Primfaktoren der *15* ermittelt ⁴.

Trotz alledem werden irgendwann funktionierende Quantencomputer zu finden sein und, die mit dem RSA-Verfahren verschlüsselten Daten von heute, werden in der Zukunft in Gefahr sein.

Bis dahin, jedenfalls, funktioniert das RSA-Verfahren.

⁴„Demonstration of a Compiled Version of Shor's Quantum Factoring Algorithm Using Photonic Qubits“, u.a. Lu, Chao-Yang and Browne, Daniel E., 19 Dec 2007

Literaturverzeichnis

- [1] BACH, E., AND SHALLIT, J. *Algorithmic number theory: Efficient algorithms*, vol. 1. MIT press, 1996.
- [2] BASHMAKOVA, I. G., AND SILVERMAN, J. H. *Diophantus and Diophantine equations*. No. 20. Cambridge University Press, 1997.
- [3] CLARK, S. Wie alt ist das universum? In *Die großen Fragen Universum*. Springer, 2012, pp. 27–35.
- [4] DICKSON, L. E. *History of the Theory of Numbers*, vol. 2. Carnegie Institution of Washington, 1920.
- [5] DUNCAN, R. Minimum rsa public key lengths: guidelines or rules? Security, Around the Net, 2012. <https://news.netcraft.com/archives/2012/09/10/minimum-rsa-public-key-lengths-guidelines-or-rules.html> *Lastaccessed* : 7Mar2021.
- [6] GIDNEY, C. Shor’s quantum factoring algorithm. Algorithmic Assertions, 2017. <https://algassert.com/post/1718> *Lastaccessed* : 9Mar2021.
- [7] HAZEWINDEL, M. Encyclopaedia of mathematics. <http://encyclopediaofmath.org/index.php?>, 2002.
- [8] INTEL. What is clock speed? Tech. rep., 2009. <https://www.intel.com/content/www/us/en/gaming/resources/cpu-clock-speed.html> *Lastaccessed* : 6Mar2021.
- [9] LOVISCACH, J. 12a.3 algorithmen, suchen und sortieren, bubble sort, quicksort, laufzeit. Loviscach, Jörn, 2012. <https://doi.org/10.5446/9600> *Lastaccessed* : 26Feb2021.
- [10] MOLLIN, R. A. *RSA and public-key cryptography*. Chapman and Hall/CRC, 2003.
- [11] NEWSON, M. W. Mathematical problems: Lecture delivered before the international congress of mathematicians at paris in 1900 by professor david hilbert. *Bulletin of the American Mathematical Society* 8 (1902).
- [12] PADBERG, F. *Elementare Zahlentheorie*, vol. 639. BI Wissenschaftsverlag, 1989.
- [13] SILVERMAN, M. P. *Quantum Superposition*. Springer, 2008.
- [14] SPANNAGEL, C. Der hauptsatz der elementaren zahlentheorie *teil1*. Pädagogische Hochschule Heidelberg *PHH*, 2012. <https://doi.org/10.5446/19874> *Lastaccessed* : 14Feb2021.
- [15] SPANNAGEL, C. Der hauptsatz der elementaren zahlentheorie *teil2*. Pädagogische Hochschule Heidelberg *PHH*, 2012. <https://doi.org/10.5446/19875> *Lastaccessed* : 18Feb2021.

- [16] SPANNAGEL, C. Der hauptsatz der elementaren zahlentheorie *teil3*. Pädagogische Hochschule Heidelberg *PHH*, 2012. <https://doi.org/10.5446/19876> *Lastaccessed* : 18Feb2021.
- [17] SPANNAGEL, C. Der kleine satz von fermat. Pädagogische Hochschule Heidelberg *PHH*, 2012. <https://doi.org/10.5446/19892> *Lastaccessed* : 25Feb2021.
- [18] SPANNAGEL, C. Der satz von euler. Pädagogische Hochschule Heidelberg *PHH*, 2012. <https://doi.org/10.5446/19893> *Lastaccessed* : 25Feb2021.
- [19] SPANNAGEL, C. Erweiterter euklidischer algorithmus teil 2. Pädagogische Hochschule Heidelberg *PHH*, 2012. <https://doi.org/10.5446/19886> *Lastaccessed* : 15Feb2021.
- [20] SPANNAGEL, C. Erweiterter euklidischer algorithmus teil 3. Pädagogische Hochschule Heidelberg *PHH*, 2012. <https://doi.org/10.5446/19887> *Lastaccessed* : 15Feb2021.
- [21] SPANNAGEL, C. Erweiterter euklidischer algorithmus teil1. Pädagogische Hochschule Heidelberg *PHH*, 2012. <https://doi.org/10.5446/19885> *Lastaccessed* : 11Feb2021.
- [22] SPANNAGEL, C. Lösbarkeit linearer diophantischer gleichungen. Pädagogische Hochschule Heidelberg *PHH*, 2012. <https://doi.org/10.5446/19888> *Lastaccessed* : 14Feb2021.
- [23] SPANNAGEL, C. Satz des euklid. Pädagogische Hochschule Heidelberg *PHH*, 2012. <https://doi.org/10.5446/19866> *Lastaccessed* : 13Feb2021.
- [24] SPANNAGEL, C. Teilmengen und primzahlen. Pädagogische Hochschule Heidelberg *PHH*, 2012. <https://doi.org/10.5446/19879> *Lastaccessed* : 05Feb2021.
- [25] TURNER, A. Logarithm of a bigdecimal. <https://stackoverflow.com, 2010. https://stackoverflow.com/questions/739532/logarithm-of-a-bigdecimal> *Lastaccessed* : 6Mar2021.

Anhang

[A]:

```
public static int addition()
{
    int eineZahl = 10;
    int eineWeitereZahl = 22;

    int additionBeiderZahlen = eineZahl + eineWeitereZahl;
    return additionBeiderZahlen;
}
```

Abbildung 3: Addition der Zahlen 10 und 22, implementiert in Java

[B]:

```
public static void schleife()
{
    for(int ausgabe=1; ausgabe<=10; ausgabe++)
    {
        System.out.println(ausgabe);
    }
}
```

Abbildung 4: Schleife von 1 bis einschließlich 10

[C]:

Programmierung zu finden unter
<https://github.com/goosz/Facharbeit2021/tree/main/PFZ>

[D]:

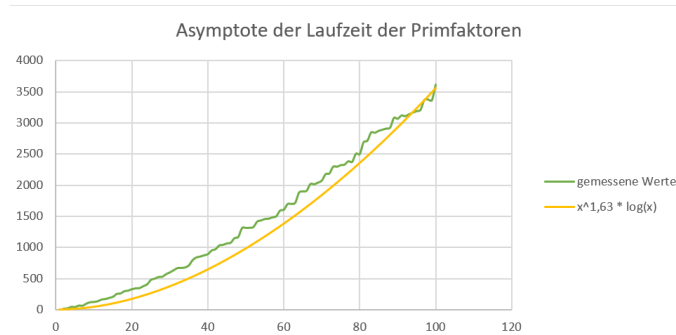


Abbildung 5: Die gemessenen Werte der Laufzeit der Primfaktorenfindung aufgetragen mit der Funktion $f(x) = x^{1,63} \cdot \log x$

Auch zu finden unter

<https://github.com/goosz/Facharbeit2021/tree/main/PFZ>

[E]:

Programmierung zu finden unter

<https://github.com/goosz/Facharbeit2021/tree/main/RSA>

[F]:

Zu beweisen. Jede natürliche Zahl $n \geq 2$ besitzt eine eindeutige Primfaktorzerlegung.

Sei $n \in \mathbb{N} \setminus \{1\}$, dann gibt es zwei verschiedene Fälle.

Fall 1:

$$n \in \mathbb{P}$$

dann ist alles bewiesen und n ist der Primfaktor von sich selbst.

Fall 2:

$$n \notin \mathbb{P}$$

dann existieren zwei Zahlen $d_1, k_1 \in \mathbb{N} \setminus \{1\}$, mit

$$d_1 \cdot k_1 = n,$$

wobei d_1 , laut 1.6 das kleinste Element in $T(n) \setminus \{1\}$ ist, und dadurch $d_1 \in \mathbb{P}$ gilt.

Nun betrachten wir k_1 :

Und es gibt erneut zwei Fälle:

Fall 1:

$$k_1 \in \mathbb{P}$$

dann haben wir unsere Annahme erneut bestätigt und die Primfaktorzerlegung lautet: $d_1 \cdot k_1$.

Ansonsten gibt es noch Fall 2:

$$k_1 \notin \mathbb{P}$$

dann verfahren wir wie oben und $k_1 = d_2 \cdot k_2$.

Sollte einer der beiden Faktoren keine Primzahl sein, so werden die einzelnen Faktoren dessen immer kleiner, bis irgendwann das kleinste Element die 2 rauskommt.

Es gibt also immer eine Primfaktorzerlegung, nun wollen wir aber beweisen, dass jede Zahl genau eine hat.

Annahme. Sei n die kleinste natürliche Zahl mit mehr als einer Primfaktorzerlegung. Also haben wir nun

$$n = p_1 \cdot p_2 \cdot p_3 \cdot \dots \cdot p_k, \text{ mit } p_k \in \mathbb{P}$$

und

$$n = q_1 \cdot q_2 \cdot q_3 \cdot \dots \cdot q_l, \text{ mit } q_l \in \mathbb{P}$$

Wichtig ist nun, dass

p_k und q_l paarweise verschieden sind.

Dies muss gegeben sein, weil wenn nur ein Faktor $p_1 = q_1$, könnte man $n \div p_1$ rechnen, und $n \div p_1 < n$, was aber einen Widerspruch zu unserer Wahl von n , als kleinstes Element mit mehr als einer Primfaktorzerlegung, darstellt. Deshalb gilt:

$$p_k \neq q_l.$$

o.B.d.A. (ohne Beschränkung der Allgemeinheit) nehmen wir nun einfach an, dass $p_1 < q_1$.

Wir definieren jetzt drei neue Zahlen a, b, c :

$$1) \ a = n \div p_1 = p_2 \cdot p_3 \cdot \dots \cdot p_k, \text{ und damit } a < n,$$

$$2) \ b = n \div q_1 = q_2 \cdot q_3 \cdot \dots \cdot q_l, \text{ und damit } b < n$$

und zuletzt

$$3) \ c = n - p_1 \cdot b, \text{ und damit } 0 < c < n.$$

a, b, c haben also eine eindeutige Primfaktorzerlegung, denn sie sind kleiner als n , das kleinste Element ohne eindeutige Primfaktorzerlegung.

Wenn wir nun 1) umformen, und für n in 3) einsetzen, erhalten wir folgendes:

$$A) \ c = n - p_1 \cdot b = p_1 \cdot a - p_1 \cdot b = p_1 \cdot (a - b).$$

Wir können aber auch die 2) umformen und ebenfalls für n in 3) einsetzen und erhalten eine zweite Gleichung:

$$\mathcal{B}) \quad c = n - p_1 \cdot b = q_1 \cdot b - p_1 \cdot b = (q_1 - p_1) \cdot b.$$

Nun setzen wir $\mathcal{A})$ und $\mathcal{B})$ gleich, da beide der selben Zahl c entsprechen.

$$p_1 \cdot (a - b) = (q_1 - p_1) \cdot b$$

Und man sieht, dass $p_1 \mid (q_1 - p_1) \cdot b$.
 p_1 teilt allerdings nicht b , da p_1 und q_1 teilerfremd sind.
 Weshalb

$$p_1 \mid q_1 - p_1$$

gelten muss.

Außerdem wissen wir, dass

$$p_1 \mid p_1.$$

Dank 1.2 gilt auch folgendes:

$$p_1 \mid (q_1 - p_1) + p_1$$

und damit

$$p_1 \mid q_1.$$

Dies kann aber nicht sein, da zum einen p_1 und q_1 Primzahlen, und desweiteren teilerfremd sind.

Daher hat n genau eine eindeutige Primfaktorzerlegung [12, 14, 15, 16]. ■

[G]:

Sei $n \in \mathbb{N} \setminus \{1\}$. Der kleinste Teiler $d \in \mathbb{N}$ mit $d > 1$ ist eine Primzahl. Nach dem Wohlordnungsprinzip ist die Teilmengenfolge nicht leer, da es eine kleinste Zahl gibt, diese nennen wir d .

Die Beweistechnik, welche hier angewandt wird, ist erneut ein indirekter Beweis.

Annahme. Um zu beweisen, dass $d \in \mathbb{P}$, nehmen wir an $d \notin \mathbb{P}$.

Dann existieren zwei Zahlen,

$$a, b \in \mathbb{N} \setminus \{1, d\}, \text{ mit } a \cdot b = d \text{ (und außerdem } a, b < d).$$

Daraus folgt, dass $a \mid d$ und auch noch $d \mid n$. Hier gilt laut 1.2 eine Transitivität und

$$a \mid n, \text{ mit } a < d, \quad d < n \text{ und folglich } a < n.$$

Dies führt aber zu einem Widerspruch, dass d das kleinste Element in $T(n) \setminus \{1\}$ ist.

Schlussfolgernd muss $d \in \mathbb{P}$ [14]. ■

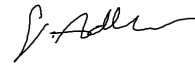
Ich erkläre hiermit, dass ich die vorliegende Facharbeit

Wie sicher ist das RSA-Verfahren?

ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis aufgeführt und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, habe ich in jedem Falle unter Angabe der Quelle kenntlich gemacht.

Köln, den 14.03.2021

Ort, Datum



Unterschrift