

# Angewandte Steganografie und die Grundlagen der Kryptografie

## STUDIENARBEIT

für die Prüfung zum

Bachelor of Science

des Studiengangs Informatik / Angewandte Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

**Jens Döllmann**

Abgabedatum 17. Mai 2021

Bearbeitungszeitraum	2 Semester
Matrikelnummer	8876462
Kurs	TINF18B4
Gutachter der Studienakademie	Ralf Brune

## **Erklärung**

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: „Angewandte Steganografie und die Grundlagen der Kryptografie“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

---

Ort      Datum

---

Unterschrift

## Zusammenfassung

Diese Arbeit beschäftigt sich mit dem Entwurf eines steganografischen Algorithmus, um Nachrichten in einer Bilddatei zu verstecken. Die Prozedur verwendet ein *least significant bit* Einbettungsverfahren und es kann experimentell gezeigt werden, dass durch pseudozufällige Pixelauswahl bis zu einer oberen Nachrichtenlänge ein steganografisch gutes Ergebnis erreicht wird. Das Verfahren wird als Teil einer Anwendung verwendet, um einen sicheren Nachrichtenaustausch über in Bildern versteckten Informationen zu ermöglichen. Am Beispiel von symmetrischen und asymmetrischen Chiffren werden Begriffe und Ideen aus dem Bereich der Kryptografie eingeführt. Wir gehen auf Strom- und Blockchiffren ein und beweisen warum die RSA-Verschlüsselung funktioniert und sicher ist. Die Sicherheitsdienste der Anwendung können anschließend ermittelt und bewertet werden.

## Abstract

We show how to design an algorithm for data hiding in colour images, using a least significant bit embedding technique. This type of secret sharing is called steganography. Considering some upper bound message length, experiments show that high-quality stego-images can be achieved by selecting a pseudorandom distribution of pixel positions. Using the algorithm as part of an application we can describe a system for securely sharing secret information through messages embedded in images. We explain some of the main techniques in cryptography, with chapters addressing symmetric as well as asymmetric ciphers. In particular, we introduce stream and block ciphers and proof why the RSA scheme works and is secure. The security services of the application can then be defined and afterwards evaluated.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Symmetrische Verschlüsselung . . . . .	3
1.2 Modulare Arithmetik . . . . .	5
1.3 Die Cäsar-Chiffre . . . . .	8
<b>2 Stromchiffren und Blockchiffren</b>	<b>10</b>
2.1 Ein Vergleich der Verfahren . . . . .	10
2.2 Die Ver- und Entschlüsselung mit Stromchiffren . . . . .	12
2.3 Zufallszahlengeneratoren . . . . .	15
2.4 Das One-Time-Pad . . . . .	16
2.5 Die praktischen Stromchiffren . . . . .	17
<b>3 Das RSA-Kryptosystem</b>	<b>19</b>
3.1 Asymmetrische Verschlüsselung . . . . .	19
3.2 Zahlentheoretische Grundlagen . . . . .	23
3.2.1 Der Euklidische Algorithmus . . . . .	23
3.2.2 Der erweiterte Euklidische Algorithmus . . . . .	25
3.2.3 Die Eulersche Phi-Funktion . . . . .	30
3.2.4 Satz von Euler und Fermat . . . . .	31
3.3 Das RSA-Verfahren . . . . .	33
<b>4 Ein Steganografischer Algorihtmus</b>	<b>37</b>
4.1 Modifikation von Bilddateien . . . . .	37
4.2 Wie stark kann ein Bild angepasst werden? . . . . .	38
4.3 Beschreibung eines Algorihtmus . . . . .	40

4.4	Die Architektur der Anwendung . . . . .	44
4.5	Sicherheitsanalyse . . . . .	46
4.5.1	Risiken der Token basierten Autorisierung . . . . .	46
4.5.2	Kryptografische Qualität . . . . .	47
4.5.3	Steganografische Qualität . . . . .	48
<b>5</b>	<b>Reflektion und Ausblick</b>	<b>55</b>
	<b>Literaturverzeichnis</b>	<b>A</b>
	<b>Glossar</b>	<b>B</b>

# Tabellenverzeichnis

1.1	Enkodierung des lateinischen Alphabets . . . . .	8
2.1	Wahrheitstabelle der Addition Modulo 2 . . . . .	13
2.2	Wahrheitstabelle der Exklusiv-Oder-Verknüpfung . . . . .	14
3.1	Erweiterter Euklidischer Algorithmus . . . . .	28
4.1	Die Zuordnung von $n$ zu der Nachrichtenlänge $l$ in Bezug auf $B_{max}$ . . . . .	48

# Abbildungsverzeichnis

1.1	Die Kryptologie und ihre Untergebiete (Paar und Pelzl 2010, S. 3) . . . . .	2
1.2	Kommunikation über einen unsicheren Kanal . . . . .	4
1.3	Kommunikation mit symmetrischer Verschlüsselung . . . . .	4
2.1	Unterteilung der Symmetrischen Chiffren (Paar und Pelzl 2010, S. 29) . . . . .	10
2.2	Das Prinzip der Verschlüsselung von $n$ Bits mittels Strom- (a) und Blockchiffre (b) (Paar und Pelzl 2010, S. 30) . . . . .	11
2.3	Der Ver- und Entschlüsselungsprozess bei Stromchiffren . . . . .	14
2.4	Praktische Stromchiffre mit Schlüsselstromerzeugung (Paar und Pelzl 2010, S. 38) . . . . .	17
3.1	Das Prinzip der symmetrischen Verschlüsselung . . . . .	19
3.2	Das Prinzip der asymmetrischen Verschlüsselung . . . . .	21
3.3	Schlüsselaustausch mit asymmetrischer Verschlüsselung . . . . .	22
4.1	Farbbild Paprika verändert durch maximalen Fehler für $n \in [0, 8]$ . . . . .	39
4.2	Koordinatenverteilung auf einem $500 \times 500$ Pixel Farbbild mit schwarzen Hintergrund für Nachrichtenlängen von 1200, 3600, 10800 und 32400 Byte.	43
4.3	Sequenzdiagramm JWT Autorisierung . . . . .	45
4.4	Paprika mit einer Größe von $509 \times 509$ Pixel. Gute Bildqualität zwischen 373 und 466 kB versteckter Nachricht ( $B_{max} \approx 777$ kB). . . . .	49
4.5	Schildkröte mit einer von Größe $1368 \times 1824$ Pixel. Gute Bildqualität bis zu 3.6 MB versteckter Nachricht ( $B_{max} \approx 7.5$ MB). . . . .	50
4.6	Blumen mit einer Größe von $2848 \times 4272$ Pixel. Gute Bildqualität bis zu 22 MB versteckter Nachricht ( $B_{max} \approx 36.5$ MB). . . . .	51
4.7	Benutzeroberfläche Nachrichten schreiben . . . . .	53

4.8 Benutzeroberfläche Nachrichten lesen . . . . .	54
----------------------------------------------------	----

# Liste der Algorithmen

3.1	Euklidischer Algorithmus . . . . .	25
3.2	Erweiterter Euklidischer Algorithmus . . . . .	29
4.1	LSB-Verfahren Schreiben . . . . .	40
4.2	LSB-Verfahren Lesen . . . . .	41
4.3	Fisher-Yates-Verfahren . . . . .	41
4.4	<i>Data Protection API</i> Verschlüsselung . . . . .	47
4.5	<i>Data Protection API</i> Entschlüsselung . . . . .	47

# Kapitel 1

## Einleitung

Redet man heutzutage über das Thema Kryptografie, sind im Gespräch wahrscheinlich Themen wie E-Mail-Verschlüsselung, Internetprotokolle oder Anwendungen im Bankenwesen. Auch bekannt sind die Angriffe auf kryptografische Systeme wie die Entzifferung der durch die Enigma-Chiffriermaschine verschlüsselten deutschen Funksprüche während des Zweiten Weltkrieges. Es scheint, als wäre Kryptografie stark mit den modernen elektronischen Kommunikationstechniken verbunden. Dies ist allerdings nicht so: Frühe Formen der Kryptografie gehen zurück bis etwa 2000 v. Chr., als bereits im antiken Ägypten neben den Standard-Hieroglyphen zusätzlich auch „geheime“ Zeichen verwendet wurden (Paar und Pelzl 2010, S. 2). Es werden prinzipiell zwei Varianten kryptografischer Verfahren unterschieden, diese sind symmetrische und asymmetrische Algorithmen. Die symmetrische Verschlüsselung ist seit langer Zeit ein fester Bestandteil der Kryptografie mit bekannten historischen Verfahren wie die Cäsar-Chiffre, welche bereits im antiken Rom für das Verschlüsseln von Nachrichten verwendet wurde. Asymmetrische Verschlüsselung hingegen ist eine gänzlich neue Form der Kryptografie, Whitfield Diffie, Martin Hellman und Ralph Merkle haben die Idee im Jahr 1976 erstmalig öffentlich eingeführt ([ebd.](#), S. 3). Eine Übersicht über das Gebiet der Kryptografie ist in [Abbildung 1.1](#) zu sehen. Es ist zu bemerken, dass an oberster Stelle nicht die Kryptografie, sondern der Oberbegriff Kryptologie zu finden ist, welche sich in die drei großen Bereiche unterteilt:

**Kryptografie** Die Wissenschaft, eine Nachricht so zu verändern, dass ihr Sinn nur von dem Empfänger verstanden werden kann, für den sie bestimmt ist.

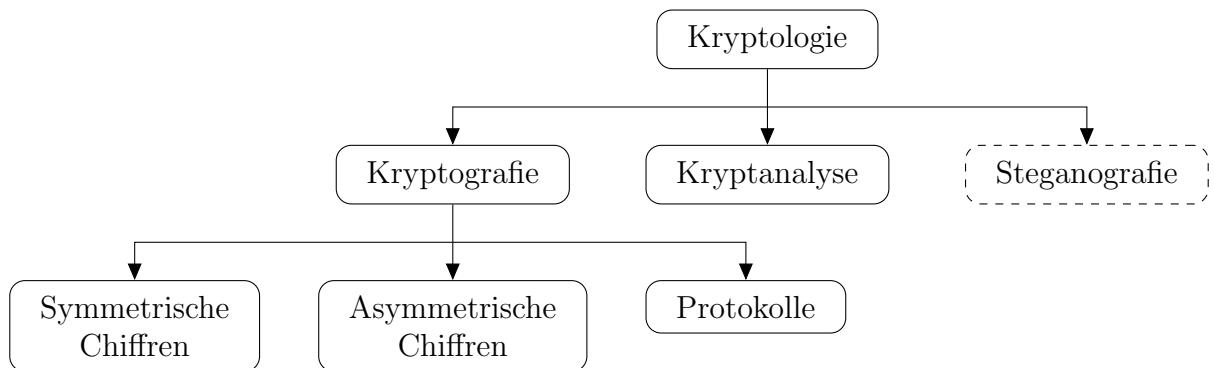


Abbildung 1.1: Die Kryptologie und ihre Untergebiete (Paar und Pelzl 2010, S. 3)

**Kryptanalyse** Die Wissenschaft, ein kryptografisches System zu analysieren mit dem Ziel, mögliche Schwachstellen aufzudecken. Die Kryptanalyse ist ein äußerst wichtiger Teil der Kryptologie, denn ohne Personen, welche versuchen ein kryptografisches System zu brechen, wird man nie herausfinden können, ob dieses wirklich sicher ist.

**Steganografie** Die Kunst, Informationen in einer für den Computer zugänglichen Trägerdatei (engl. *cover media*) zu verstecken, sodass eine weitere Person die Existenz einer geheimen Botschaft gar nicht erst vermuten würde. Die Steganografie wird in der Literatur häufig nicht als ein Bereich der Kryptologie erwähnt, soll aber als wesentlicher Teil dieser Studienarbeit auch hier schon betrachtet werden. Es ist wichtig abzugrenzen, dass es sich im Gegensatz zu den Chiffren der Kryptografie nicht um ein Verschlüsselungsverfahren handelt und ein Außenstehender, welcher von der Existenz einer geheimen Nachricht Bescheid weiß, diese unverschlüsselt lesen könnte.

Ein starkes Kryptoverfahren sollte dem *Kerckhoffs's principle* unterliegen, welches im Jahr 1883 von Auguste Kerckhoffs postuliert wurde und von Paar und Pelzl durch folgende Definition beschrieben ist (2010, S. 11):

**Definition 1.0.1 (Kerckhoffs's principle).** „A cryptosystem should be secure even if the attacker knows all details about the system, with the exception of the secret key. In particular, the system should be secure when the attacker knows the encryption and decryption algorithms.“

Die Sicherheit eines Kryptosystems darf also nur auf der Geheimhaltung des Schlüssels beruhen. Auf den ersten Blick scheint das *Kerckhoffs's principle* nicht sonderlich intuitiv.

tiv. Es sei einfach zu glauben, dass ein System sicherer sein muss, wenn die Details der Implementierung geheim gehalten werden. In der Regel ist dies aber nicht so. Ein Kryptoverfahren bleibt nicht für immer geheim und die Vergangenheit hat gezeigt, dass ein System, dessen geheimes Design an die Öffentlichkeit gelangt, fast immer unsicher ist. Ein hierfür gutes Beispiel ist das Content Scrambling System (CSS) für das Verschlüsseln von DVD-Videoinhalten. Trotz großer Bemühungen der Industrie, die Funktionsweise von CSS geheim zu halten, gelang das Design durch Reverse Code Engineering schnell an die Öffentlichkeit. Es zeigten sich Mängel in der Implementierung, welche das Brechen der Verschlüsselung mit sehr geringem Aufwand ermöglichten (Barry 2004). Am Beispiel von symmetrischen und asymmetrischen Verschlüsselungsverfahren werden in den folgenden Kapiteln Begriffe und Ideen aus dem Bereich der Kryptografie eingeführt. Zusätzlich wird in [Kapitel 4](#) gezielt darauf eingegangen, wie ein steganografischer Algorithmus entworfen werden kann. Der Algorithmus soll eine beliebig lange Nachricht in einer Bilddatei verstecken, ohne dabei die visuelle Qualität sichtbar zu beeinflussen. Es werden Fragen untersucht, wie stark ein Bild angepasst werden kann und ab welcher Nachrichtenlängen eine Veränderung sichtbar wird.

## 1.1 Symmetrische Verschlüsselung

Denkt man an die Teilbereiche der Kryptografie, ist die symmetrische Verschlüsselung das wohl klassischste Beispiel. Zwei Parteien kommunizieren mit einem Algorithmus zum Ver- und Entschlüsseln von Nachrichten und haben sich auf einen gemeinsamen geheimen Schlüssel geeinigt. Wie es in der Literatur sehr beliebt ist, wird die Idee der symmetrischen Verschlüsselung mit einem einfachen Beispiel eingeführt (Paar und Pelzl 2010, S. 4–6): Zwei Parteien Alice und Bob möchten über einen unsicheren Kanal Nachrichten untereinander austauschen. Ein unsicherer Kanal ist hierbei lediglich die Kommunikationsstrecke, z. B. das Internet, die Luftschnittstelle bei WLAN und Mobilfunk oder jedes andere Medium, über das sich digitale Daten übertragen lassen.

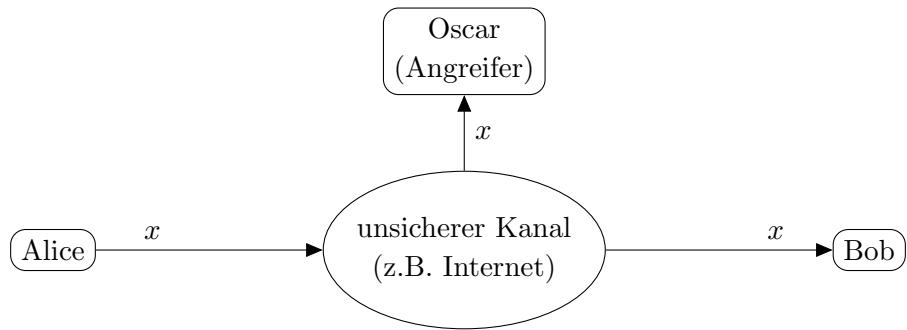


Abbildung 1.2: Kommunikation über einen unsicheren Kanal

Es ist klar, warum Alice und Bob gerne geheime Nachrichten austauschen würden: Alice möchte sich an ihrem Bankkonto anmelden und sendet ihr Passwort zu Bob. Ein potenzieller Angreifer Oscar soll die Passwörter von Alice nicht in Klartext mitlesen können. In einer solchen Situation bietet die symmetrische Verschlüsselung eine gute Lösung: Bevor Alice ihr Passwort sendet, verschlüsselt sie es mit einem symmetrischen Algorithmus. Bob invertiert die Verschlüsselung und erhält die unverschlüsselte Nachricht. Wurde für die Verschlüsselung ein sicherer Algorithmus gewählt, erscheint die Nachricht für Oscar nur wie eine zufällige Folge von Bit.

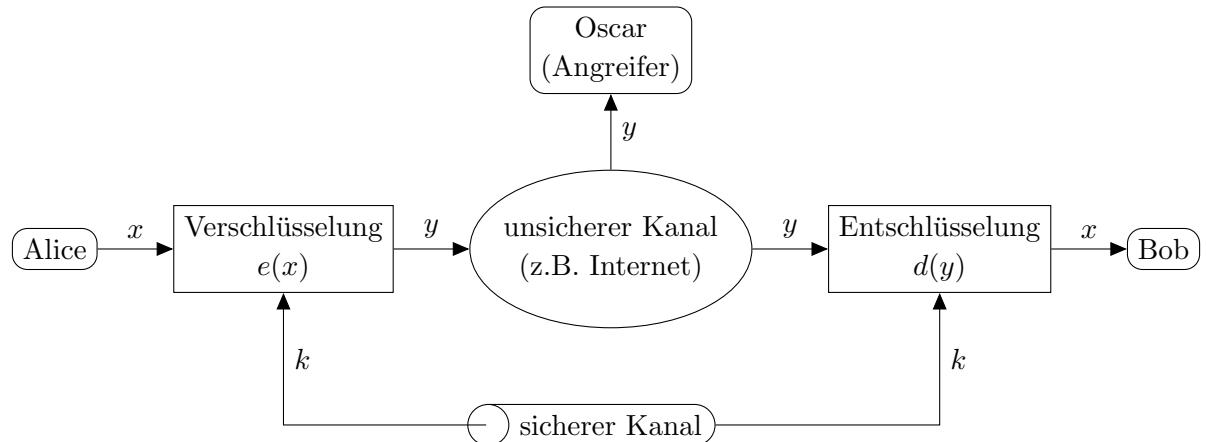


Abbildung 1.3: Kommunikation mit symmetrischer Verschlüsselung

Die Variablen  $x, y$  und  $k$  aus Abbildung 1.3 haben in der Kryptografie eine besondere Bedeutung:

- $x$  ist der Klartext (engl. *plaintext*).

- $y$  ist das Chiffraut oder der Geheimtext (engl. *ciphertext*).
- $k$  ist der Schlüssel (engl. *key*).
- $e$  bezeichnet die Verschlüsselung (engl. *encryption*).
- $d$  bezeichnet die Entschlüsselung (engl. *decryption*).

Für die symmetrische Verschlüsselung wird der geheime Schlüssel  $k$  benötigt. Dieser muss vor der Kommunikation auf einem sicheren Weg zwischen Alice und Bob verteilt werden.

## 1.2 Modulare Arithmetik

Fast alle kryptografischen Algorithmen, sowohl symmetrische als auch asymmetrische Chiffren basieren auf Arithmetik in einer endlichen Menge von ganzen Zahlen (Paar und Pelzl 2010, S. 13). Dies steht im Gegensatz zu der Mathematik (und dem Alltagsleben), von der wir es gewohnt sind, in unendlichen Mengen zu rechnen, z. B. den natürlichen oder reellen Zahlen. Die modulare Arithmetik, also die Division mit Rest, bietet eine gute Möglichkeit, um mit diesen begrenzten Mengen zu arbeiten.

**Definition 1.2.1** (Teilbarkeitsbegriff (Remmert und Ullrich 2008, S. 22)). Eine Zahl  $d \in \mathbb{Z}$  heißt *Teiler* der Zahl  $a \in \mathbb{Z}$ , in Zeichen  $d | a$ , wenn es eine Zahl  $v \in \mathbb{Z}$  gibt, sodass gilt:  $a = dv$ . Ist  $d$  kein *Teiler* von  $a$ , so schreibt man:  $d \nmid a$ .

Beispiel 1.2.1.  $3 | 12, -5 | 20, 3 \nmid 7, 19 | 0, 0 | 0$

△

**Lemma 1.2.1** (ebd., S. 179–180). *Folgende Aussagen über drei ganze Zahlen  $a, b, m$ , wobei  $m > 0$ , sind äquivalent:*

- a und b lassen bei Division mit Rest durch m denselben Rest.*
- Die Differenz  $a - b$  ist durch m teilbar.*

*Beweis.* Es seien  $a = q_1m + r_1$  und  $b = q_2m + r_2$  mit  $0 \leq r_1, r_2 < m$  und  $q_1, q_2, r_1, r_2 \in \mathbb{Z}$  die Gleichungen, die bei Division mit Rest entstehen.

*i)  $\Rightarrow$  ii):* Es gilt  $a - b = q_1m + r_1 - q_2m - r_2 = (q_1 - q_2)m + (r_1 - r_2)$ . Da  $r_1 = r_2$  vorausgesetzt wird, folgt:  $m | (a - b)$ .

ii)  $\Rightarrow$  i): Es gilt  $m \mid (a - b) \Leftrightarrow m \mid (q_1 - q_2)m + (r_1 - r_2)$ . Teilt  $m$  zwei Zahlen  $a$  und  $b$  muss  $m$  auch eine Linearkombination dieser Zahlen teilen.

$$\begin{aligned} & m \mid (q_1 - q_2)m + (r_1 - r_2) \wedge m \mid (q_1 - q_2)m \\ \Rightarrow & m \mid (q_1 - q_2)m + (r_1 - r_2) - (q_1 - q_2)m \\ \Leftrightarrow & m \mid (r_1 - r_2) \end{aligned}$$

Aus der letzten Zeile folgt wegen  $0 \leq r_1, r_2 < m$  sofort:  $r_1 - r_2 = 0$ . Es folgt:  $r_1 = r_2$ .  $\square$

Nach Gauß nennt man zwei Zahlen  $a, b \in \mathbb{Z}$ , die bei der Division durch  $m$  denselben Rest ergeben, *kongruent Modulo  $m$* . Anstelle der schwerfälligen Teilbarkeitsschreibweise  $m \mid (a - b)$  führte Gauß folgende Schreibweise ein (Remmert und Ullrich 2008, S. 180):

$$a \equiv b \pmod{m} \quad \text{oder kürzer: } a \equiv b \pmod{m} \quad (1.1)$$

Beispiel 1.2.2. Es sind  $a = 29$  und  $m = 8$ . Man schreibt:

$$29 \equiv 5 \pmod{8} \quad \text{oder als Gleichung: } 29 = 3 \cdot 8 + 5 \quad \triangle$$

**Gleichung 1.1** hat unendlich viele Lösungen. Elemente welche bei Division durch  $m$  denselben Rest ergeben, können in eine Klasse zusammengefasst werden, diese nennt man dann eine Restklasse bezüglich  $m$ .

Beispiel 1.2.3. Die acht Restklassen bezüglich 8 sind:

$$\begin{aligned} |0|_8 &= \{\dots, -24, -16, -8, 0, 8, 16, 24, \dots\} \\ |1|_8 &= \{\dots, -25, -17, -9, 1, 9, 17, 25, \dots\} \\ &\vdots \\ |7|_8 &= \{\dots, -17, -9, -1, 7, 15, 23, 31, \dots\} \end{aligned}$$

Man schreibt  $|a|_m$  und bezeichnet  $a$  als einen Repräsentanten der Restklasse mit dem Modul  $m$ .  $\triangle$

Alle Elemente einer Restklasse verhalten sich gleich. Es spielt keine Rolle, welches Element für eine Berechnung ausgewählt wird. Diese Eigenschaft ist von großem Nutzen vor allem

bei Berechnungen mit großen Zahlen, wie es in der Kryptografie oft der Fall ist.

Beispiel 1.2.4 (Paar und Pelzl 2010, S. 15–16). Die Hauptoperation in vielen asymmetrischen Chiffren ist die Exponentiation der Form  $x^e \bmod m$ , wobei  $x, e, m$  sehr große natürliche Zahlen sind. Anhand eines Beispiels können zwei Formen der modularen Exponentiation gezeigt werden. Es soll das Ergebnis der Berechnung  $3^8 \bmod 7$  ermittelt werden. Im ersten Beispiel wird das Ergebnis einfach ausgerechnet, und im zweiten Beispiel wird zwischen den Restklassen gewechselt:

1. Es gilt  $3^8 = 6561 \equiv 2 \pmod{7}$ , wir erhalten das relative große Zwischenergebnis 6561 obwohl wir wissen, dass das Ergebnis im Bereich  $[0, 6]$  liegen muss.
2. Es gilt  $3^3 = 27 \equiv -1 \pmod{7}$ , man schreibt:

$$3^8 = (3^3)^2 \cdot 3^2 \equiv (-1)^2 \cdot 9 \equiv 2 \pmod{7}$$

Indem das Zwischenergebnis  $3^3 = 27$  mit einem kleineren Element aus der selben Restklasse ersetzt wird, kann das Ergebnis effizient ermittelt werden. Zwischenergebnisse werden nie größer als 27 und man könnte die Berechnung mit wenig Aufwand auch ohne Taschenrechner durchführen.  $\triangle$

Wir einigen uns für den folgenden Text darauf, ein  $b$  in (1.1) für gewöhnlich so zu wählen, dass:  $0 \leq b < m$ . Man schreibt somit  $27 \equiv 6 \pmod{7}$  und nicht  $27 \equiv -1 \pmod{7}$  oder  $27 \equiv 13 \pmod{7}$ . Mathematisch macht es jedoch keinen Unterschied. Wir vereinbaren für den folgenden Text außerdem, dass auch die Null eine natürliche Zahl ist, also  $\mathbb{N} := \{0, 1, 2, \dots, 100, 101, \dots\}$ . Da häufig die Menge  $\{1, 2, 3, \dots\}$  aller von 0 verschiedenen natürlichen Zahlen betrachtet wird, ist es sinnvoll, auch für diese Menge ein Symbol einzuführen. Wir vereinbaren folgende Bezeichnung:  $\mathbb{N}^\times := \{1, 2, 3, \dots\}$ . Es gilt also:  $\mathbb{N} := \{0\} \cup \mathbb{N}^\times$ . Um das bekannte Verschlüsselungsverfahren, die Cäsar-Chiffre, im nächsten Abschnitt besser beschreiben zu können, wird an dieser Stelle der Ring  $\mathbb{Z}_m$  definiert:

**Definition 1.2.2** (Der Ring  $\mathbb{Z}_m$  der Reste Modulo  $m$ ).

1. Der Ring  $\mathbb{Z}_m$  mit  $m \in \mathbb{N}$  und  $m > 1$  ist definiert als die Menge  $\mathbb{Z}_m = \{0, 1, \dots, m-1\}$
2. Zu je zwei Elementen  $a, b \in \mathbb{Z}_m$  existiert eindeutig in  $\mathbb{Z}_m$ 
  - (a) Eine Summe  $a + b \bmod m$
  - (b) Ein Produkt  $a \cdot b \bmod m$

## 1.3 Die Cäsar-Chiffre

Die Cäsar- oder Verschiebe-Chiffre ist das vielleicht bekannteste historische Verschlüsselungsverfahren, es wird von Paar und Pelzl auf Seiten 18-19 eingeführt (2010). Bei der Cäsar-Chiffre handelt es sich um eine spezielle Form der Buchstabensubstitution. Jedes Zeichen im Klartext wird zur Verschlüsselung um einen bestimmten Wert im Alphabet verschoben. Um die Cäsar-Chiffre mathematisch zu beschreiben, muss das zu Grunde liegende Alphabet enkodiert werden. Eine Möglichkeit ist die fortlaufende Nummerierung der Buchstaben. Das so kodierte lateinische Alphabet ist in [Tabelle 1.1](#) zu sehen.

Tabelle 1.1: Enkodierung des lateinischen Alphabets

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Würde ein Buchstabe zu weit verschoben werden, wird von vorne, also beim ersten Buchstaben weitergemacht. Sowohl Klartext- als auch Geheimtextbuchstaben sind somit Teil des Rings  $\mathbb{Z}_{26}$ . Das Verfahren kann nun folgendermaßen beschrieben werden:

**Definition 1.3.1** (Cäsar-Chiffre). Es seien  $x, y, k \in \mathbb{Z}_{26}$ . Dann gilt:

**Verschlüsselung:**  $y = e_k(x) \equiv x + k \pmod{26}$  (26)

**Entschlüsselung:**  $x = d_k(y) \equiv y - k \pmod{26}$  (26)

*Beweis.* Das Verschlüsselungsverfahren funktioniert. Die Entschlüsselung des Geheimtextes muss erneut den Klartext ergeben. Zu zeigen:  $d_k(e_k(x)) \equiv x \pmod{26}$ .

$$d_k(e_k(x)) \equiv x + k - k \equiv x \pmod{26} \quad (26)$$

□

Beispiel 1.3.1. Es sei  $k = 9$  und der Klartext:

$$\text{ANGRIFF} = x_1, x_2, \dots, x_7 = 0, 13, 6, 17, 8, 5, 5$$

Der Geheimtext wird wie folgt berechnet:

$$\begin{aligned} e_9(0) &\equiv 9 \quad (26) \\ e_9(13) &\equiv 22 \quad (26) \\ e_9(6) &\equiv 15 \quad (26) \\ e_9(17) &\equiv 26 \equiv 0 \quad (26) \\ e_9(8) &\equiv 17 \quad (26) \\ e_9(5) &\equiv 14 \quad (26) \end{aligned}$$

$$y_1, y_2, \dots, y_7 = 9, 22, 15, 0, 17, 14, 14 = \text{JWPAROO}$$

△

Wie zu erwarten, bietet die Cäsar-Chiffre natürlich keine besonders sicher Verschlüsselung. Es gibt nur 26 verschiedene Schlüssel (wobei  $k = 0$  den Klartext nicht verändert), welche schnell alle ausprobiert werden können. Zusätzlich haben Klartext und Geheimtext die selben statistischen Eigenschaften. Klartextbuchstaben werden immer auf die selben Geheimtextbuchstaben abgebildet, dies erlaubt es eine Häufigkeitsanalyse der Buchstaben durchzuführen.

# Kapitel 2

## Stromchiffren und Blockchiffren

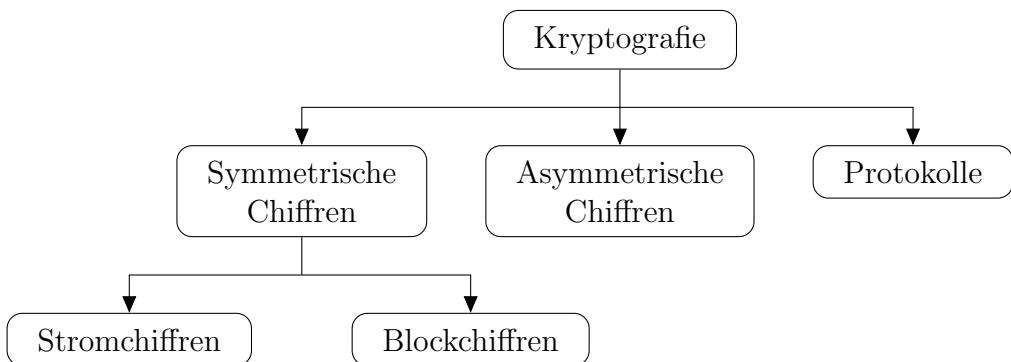


Abbildung 2.1: Unterteilung der Symmetrischen Chiffren (Paar und Pelzl 2010, S. 29)

Werfen wir in Abbildung 2.1 einen genaueren Blick auf die Algorithmen der Kryptografie, stellen wir fest: Das Gebiet der symmetrischen Verschlüsselungsverfahren kann unterteilt werden in Stromchiffren und Blockchiffren. In diesem Kapitel sollen anhand von Stromchiffren einige Definitionen eingeführt und der Unterschied zu den Blockchiffren erläutert werden. Außerdem wird gezeigt, welche Rolle hierbei die Zufallszahlengeneratoren spielen.

### 2.1 Ein Vergleich der Verfahren

Die symmetrischen Verschlüsselungsverfahren sind unterteilt in Strom- und Blockchiffren. Während beide Verfahren das gleiche Ziel verfolgen, Informationen zu verschlüsseln, ist die jeweilige Methode eine unterschiedliche. Stromchiffren verschlüsseln jedes Bit im Klartext

einzelnen, während Blockchiffren pro Durchlauf mehrere Bits verschlüsseln können. Abbildung 2.2 zeigt diesen prinzipiellen Unterschied, für den Fall, dass  $n$  Bits verschlüsselt werden sollen.

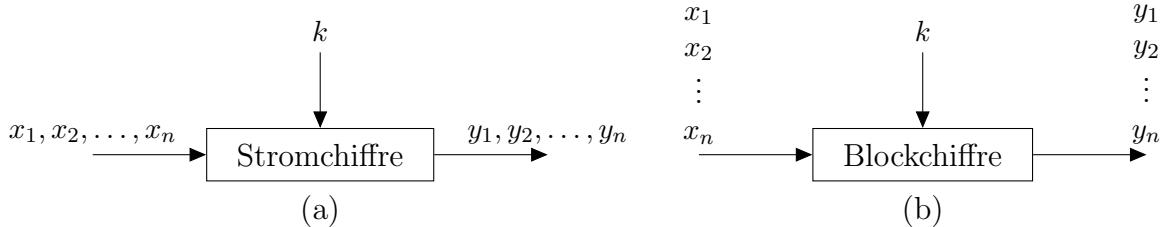


Abbildung 2.2: Das Prinzip der Verschlüsselung von  $n$  Bits mittels Strom- (a) und Blockchiffre (b) (Paar und Pelzl 2010, S. 30)

Blockchiffren arbeiten mit Datenblöcken fester Länge, wobei ein Großteil der relevanten Algorithmen eine Eingangsweite von 64 oder 128 Bit besitzen. Prominente Beispiele sind Verfahren wie der Data Encryption Standard (DES, Blocklänge 64 Bit, Paar und Pelzl 2010, S. 55–58) oder der Advanced Encryption Standard (AES, Blocklänge 128 Bit, ebd., S. 87–90). Eine grundlegende Idee für das Entwerfen starker Blockchiffren, ist das Prinzip der Konfusion (engl. *confusion*) und Diffusion (engl. *diffusion*) (ebd., S. 57):

1. Die **Konfusion** hat die Aufgabe, den Zusammenhang von Schlüssel und Geheimtext unklar zu machen. Ein beliebtes Vorgehen ist die Bitsubstitution, welche sowohl in DES als auch AES verwendet wird.
2. Die **Diffusion** hat die Aufgabe, den Einfluss an einer Stelle des Klartextes, über den gesamten Geheimtext zu verteilen. Es wird das Ziel verfolgt, statistische Eigenschaften zu verstecken. Das Ändern von einer Stelle im Klartext soll im Durchschnitt eine Änderung des halben Geheimtextes bewirken. Eine einfache Operation für Diffusion ist die Bitpermutation, welche häufig in DES eingesetzt wird. AES verwendet eine komplexere MixColumn-Transformation.

Chiffren, welche nur eine der beiden Operationen anwenden, wie die alleinige Konfusion bei der Cäsar-Chiffre, sind nicht sicher. Die heutigen Blockchiffren arbeiten iterativ in aufeinanderfolgenden Runden, die gleich aufgebaut sind. In jeder Runde wird eine Rundenfunktion angewandt, welche sowohl Konfusion als auch Diffusion durchführt. Generell lassen sich die folgenden drei Punkte zusammenfassen (ebd., S. 31):

1. Blockchiffren werden in der Praxis vor allem für die Verschlüsselung im Internet häufiger eingesetzt als Stromchiffren.
2. Stromchiffren sind oft klein und schnell und deshalb attraktiv für Anwendungen, bei denen vergleichsweise wenig Rechenleistung zur Verfügung steht, beispielsweise bei Mobilgeräten oder anderen eingebetteten Systemen. Ein bekanntes Verfahren ist der A5/1 Algorithmus, welcher Teil des GSM-Mobilfunkstandards ist und die Gesprächsdaten an der Luftschnittstelle verschlüsselt. A5/1 gilt aufgrund der kurzen Schlüssellänge von 64 Bit nicht mehr als sicher. Obwohl die vollständige Schlüsselsuche immer noch aufwändig ist, wurden viele Angriffe gezeigt, welche die Verschlüsselung in nahezu Echtzeit brechen können.
3. In der Vergangenheit galt der generelle Gedanke, dass Stromchiffren effizienter seien als Blockchiffren. Diese Annahme gilt im Allgemeinen heutzutage jedoch nicht mehr. Moderne Verfahren wie AES können sowohl in Hardware als auch Software sehr effizient implementiert werden.

## 2.2 Die Ver- und Entschlüsselung mit Stromchiffren

Stromchiffren verschlüsseln jedes Bit im Klartext einzeln. Hierfür wird mit einem Schlüssel ein geheimer Bitstrom errechnet, welcher paarweise mit den Bit des Klartextes kombiniert wird. Die Ver- und Entschlüsselung ist verblüffend einfach, es handelt sich in beide Richtungen um eine einfache Addition im Ring  $\mathbb{Z}_2$ .

**Definition 2.2.1** (Ver- und Entschlüsselung mit Stromchiffren, Paar und Pelzl 2010, S. 31). Es seien  $x_i, y_i, k_i \in \{0, 1\}$  die einzelnen Bit aus Klartext, Geheimtext und Schlüsselstrom. Es gilt:

**Verschlüsselung:**  $y_i = e_{k_i}(x_i) \equiv x_i + k_i \quad (2)$

**Entschlüsselung:**  $x_i = d_{k_i}(y_i) \equiv y_i + k_i \quad (2)$

Betrachteten wir die Ver- und Entschlüsselungsfunktion, fallen drei Aspekte auf, welche besprochen werden müssen ([ebd.](#), S. 31–34):

1. Warum sind Verschlüsselung und Entschlüsselung dieselbe Funktion?
2. Warum ist Addition im Ring  $\mathbb{Z}_2$  eine gute Verschlüsselung?
3. Was sind die Eigenschaften der Schlüsselstrombit?

## Warum sind Verschlüsselung und Entschlüsselung dieselbe Funktion?

Bis auf spezielle Kürzungsregeln erlauben Kongruenzen (ganz analog wie mit Gleichungen) das Ausführen der elementaren Rechenoperationen (Remmert und Ullrich 2008, S. 181–183). Mit diesem Wissen kann die Verschlüsselungsfunktion durch einfaches Umformen in die Entschlüsselungsfunktion überführt werden:

*Beweis.*

$$\begin{aligned} y_i &\equiv x_i + k_i \quad (2) \\ -x_i &\equiv -y_i + k_i \quad (2) \\ x_i &\equiv y_i + k_i \quad (2) \end{aligned}$$

□

Im letzten Schritt wird erneut vom Wechsel innerhalb der Restklasse Gebrauch gemacht. Es gilt:  $-1 \equiv 1 \pmod{2}$ .

## Warum ist Addition im Ring $\mathbb{Z}_2$ eine gute Verschlüsselung?

Addition in  $\mathbb{Z}_2$  liefert aufgrund der Division mit Rest nur Ergebnisse in der Menge  $\{0, 1\}$ . Dieses Verhalten ist sehr hilfreich, denn es ermöglicht das Ausdrücken der Rechenregeln durch einfache boolesche Algebra. Betrachten wir die Wahrheitstabelle (2.1) der Addition in  $\mathbb{Z}_2$ , kann sofort eine weitere Beobachtung gemacht werden: Die Addition Modulo 2 ist äquivalent zu der Exklusiv-Oder-Verknüpfung durch ein XOR-Gatter.

Tabelle 2.1: Wahrheitstabelle der Addition Modulo 2

$x_i$	$k_i$	$y_i \equiv x_i + k_i \pmod{2}$
0	0	0
0	1	1
1	0	1
1	1	0

Das XOR-Gatter spielt eine wesentliche Rolle in vielen kryptografischen Verfahren. Es besitzt besondere Eigenschaften, welche es von anderen Logikgattern unterscheidet und diese sollen jetzt untersucht werden. Angenommen es soll das Klartextbit  $x_i = 0$  verschlüsselt werden. In der Wahrheitstabelle (2.2) befindet man sich demnach in der ersten oder zweiten Zeile. Verhalten sich die Schlüsselbit unvorhersehbar, d. h. sie sind mit genau 50-prozentiger Wahrscheinlichkeit null oder eins, ist es nur durch den Klartext nicht

Tabelle 2.2: Wahrheitstabelle der Exklusiv-Oder-Verknüpfung

$x_i$	$k_i$	$y_i = x_i \oplus k_i$
0	0	0
0	1	1
1	0	1
1	1	0

möglich, auf den Geheimtext zu schließen. Zu jedem Zeitpunkt hat ein Angreifer nur eine 50-prozentige Chance, die richtigen Zeichen zu erraten. Diese Symmetrie unterscheidet das XOR-Gatter von anderen Logikgattern, zusätzlich ist es die einzige Verknüpfung, welche durch doppeltes Anwenden invertierbar ist. [Abbildung 2.3](#) zeigt den Ver- und Entschlüsselungsprozess bei Stromchiffren.

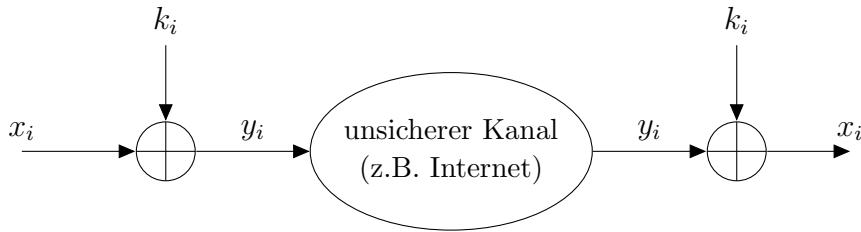


Abbildung 2.3: Der Ver- und Entschlüsselungsprozess bei Stromchiffren

In einem Beispiel soll jetzt ein einfacher Informationsaustausch demonstriert werden:

Beispiel 2.2.1. Alice möchte Bob eine Nachricht senden. Sie verschickt den Buchstaben P, wobei dieser zur Verschlüsselung in ASCII-Zeichenkodierung angegeben ist  $P = 80_{10} = 01010000_2$ . Der Schlüsselstrom wird außerdem angegeben, es seien  $(k_0, k_1, \dots, k_7) = 00111010_2$ .

$$\begin{array}{r}
 x_0, \dots, x_7 = 01010000_2 = 80_{10} = P \\
 \oplus \\
 k_0, \dots, k_7 = 00111010_2 \\
 y_0, \dots, y_7 = 01101010_2 = 106_{10} = j
 \end{array}
 \quad \text{Alice}$$
  

$$\underline{j = 01101010_2} \quad \text{Oscar}$$

$$\begin{array}{ccc}
 j = 01101010_2 & & \text{Oscar} \\
 \xrightarrow{\quad\quad\quad} & & \\
 y_0, \dots, y_7 = 01101010_2 = 106_{10} = j & & \\
 \oplus & & \text{Bob} \\
 k_0, \dots, k_7 = 00111010_2 & & \\
 x_0, \dots, x_7 = 01010000_2 = 80_{10} = P & &
 \end{array}$$

Vor der Übertragung wird der Großbuchstabe  $P$  durch die XOR-Verknüpfung in den Kleinbuchstaben  $j$  umgewandelt. Oscar kann mit dieser Information aufgrund der oben beschriebenen Eigenschaften nur wenig anfangen.  $\triangle$

Stromchiffren erscheinen fast zu gut, um wahr zu sein. Es gibt jedoch auch hier Einschränkungen, welche in der Praxis entstehen und später untersucht werden. Es bleibt die Letzte der drei Fragestellungen zu beantworten.

### Was sind die Eigenschaften der Schlüsselstrombit?

Die Sicherheit von Stromchiffren hängt vollständig von der Qualität des Schlüsselstroms ab. Das Generieren dieser Folge  $(k_1, k_2, \dots, k_n)$  bildet daher die zentrale Fragestellung der Verschlüsselungsmethode. Es wurde gezeigt, dass der Schlüsselstrom wie eine zufällige Folge von Bit aussehen muss. Das Theme der Zufallszahlen soll deshalb im nächsten Abschnitt näher untersucht werden.

## 2.3 Zufallszahlengeneratoren

Die Unvorhersehbarkeit des Schlüsselstroms ist die wesentliche Eigenschaft von Stromchiffren. Im Folgenden werden einige Formen von Zufallszahlengeneratoren (engl. *random number generators (RNGs)*) vorgestellt und deren Unterschiede beschrieben (Paar und Pelzl 2010, S. 35–36) (Haahr 2021).

**Echte Zufallszahlengeneratoren** Echte Zufallszahlengeneratoren (engl. *true random number generators (TRNGs)*) erzeugen Zahlen, welche in keiner Weise vorhersehbar und reproduzierbar sind. Wirft man eine Münze 100-mal und erzeugt eine Folge von 100 Bit, ist es quasi unmöglich, dieselbe Sequenz ein zweites Mal zu erzeugen. Die Wahrscheinlichkeit, dass dies passieren würde, beträgt 1 in  $2^{100}$ , was verschwindend gering ist. Echte

Zufallszahlen basieren auf physikalischen Prozessen wie das Werfen einer Münze, Würfeln, radioaktiver Zerfall oder atmosphärisches Rauschen. **TRNGs** werden in der Kryptografie und Schlüsselerzeugung häufig eingesetzt.

**Pseudozufallszahlengeneratoren** Pseudozufallszahlengeneratoren (engl. *pseudorandom number generators (PRNGs)*) generieren Zahlenfolgen basierend auf einem Startwert, welcher im Englischen oft als *seed* bezeichnet wird. Die Zahlen eines **PRNG** sind nicht in der Art zufällig, wie man es erwarten könnte. Obwohl sie aussehen wie echte Zufallszahlen, werden sie durch mathematische Formeln errechnet und sind deterministisch, d. h. dieselbe Sequenz kann zu einem späteren Zeitpunkt reproduziert werden. In der Kryptografie können **PRNGs** aufgrund dieser Eigenschaft nicht ohne Weiteres eingesetzt werden. Die große Relevanz der Generatoren sollte aber auf keinen Fall unterschätzt werden. **PRNGs** haben außerhalb der Kryptografie weitreichende Anwendungsgebiete, beispielsweise in der Steganografie, Simulation oder während des Testens von Software und Hardware. Determinismus ist in diesen Bereichen oftmals eine gewünschte Eigenschaft.

**Kryptografisch sichere Pseudozufallszahlengeneratoren** Kryptografisch sichere Pseudozufallszahlengeneratoren (engl. *cryptographically secure pseudorandom number generators (CSPRNGs)*) sind **PRNGs** mit einer zusätzlichen Eigenschaft: **CSPRNGs** sind unvorhersehbar. Grob gesprochen bedeutet dies, dass es rechentechnisch nicht möglich ist, aus  $n$  gegebenen Schlüsselstrombit  $(k_1, k_2, \dots, k_n)$ , die folgenden Bit  $(k_{n+1}, k_{n+2}, \dots, k_{n+m})$  zu berechnen. Außerdem soll es zeitlich unmöglich sein, eines der vorherigen Bit  $(k_0, k_{-1}, \dots, k_{-m})$  zu berechnen.

## 2.4 Das One-Time-Pad

Es sind nun mit den bisher eingeführten Ideen alle Teile vorhanden, um ein beweisbar sicheres Verschlüsselungsverfahren zu entwerfen. Ein System heißt beweisbar sicher, wenn es trotz unendlich vorhandener Rechenleistung nachweislich nicht gebrochen werden kann. Es kann folgende Definition gegeben werden (Paar und Pelzl 2010, S. 36):

**Definition 2.4.1 (Unconditional Security).** „*A cryptosystem is unconditionally or information-theoretically secure if it cannot be broken even with infinite computational resources.*“

Das **One-Time-Pad (OTP)** ist ein solches Verschlüsselungsverfahren, welches dieses Kriterium erfüllt. Es ist folgendermaßen definiert (Paar und Pelzl 2010, S. 37):

**Definition 2.4.2** (One-Time-Pad). Eine Stromchiffre heißt One-Time-Pad, wenn die folgenden Kriterien eingehalten werden:

1. Der Schlüsselstrom  $(k_1, k_2, \dots, k_n)$  wird durch einen **TRNG** generiert.
2. Nur vertrauenswürdige Gesprächspartner kennen den Schlüsselstrom.
3. Jedes Schlüsselstrombit wird nur einmal verwendet.

Das One-Time-Pad ist beweisbar sicher.

## 2.5 Die praktischen Stromchiffren

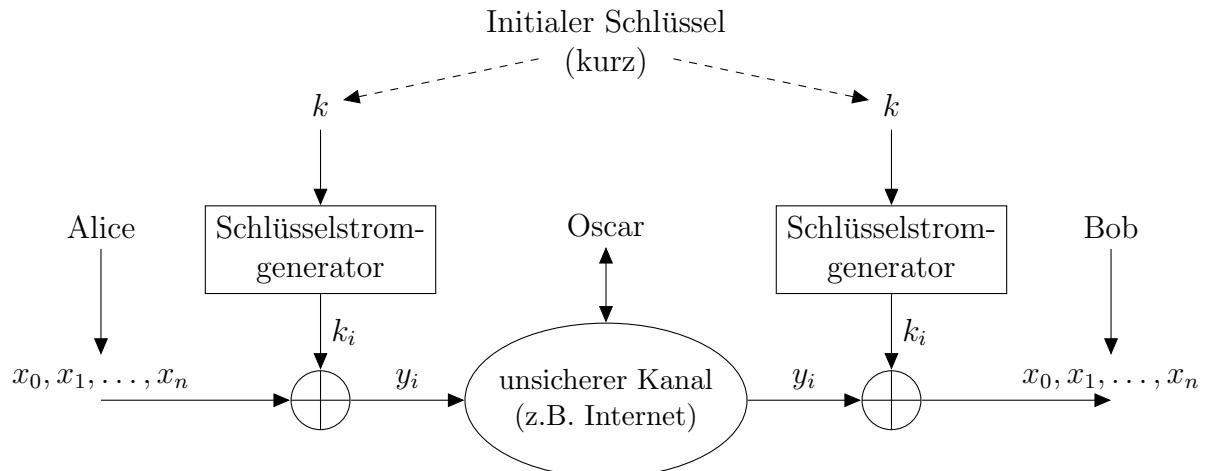


Abbildung 2.4: Praktische Stromchiffre mit Schlüsselstromerzeugung (Paar und Pelzl 2010, S. 38)

In der Praxis hat das Verschlüsseln mit **OTP** natürlich einen Hacken, denn warum sonst sollten heutzutage andere Verfahren eingesetzt werden? Der erste Nachteil ist die Anforderung eines **TRNG**, dieser benötigt spezielle Hardware und ist in der Regel nicht Teil eines Standardcomputers. Der zweite und wahrscheinlich größere Nachteil ist die Handhabung des Schlüsselstroms: Jedes Schlüsselstrombit wird nur einmal verwendet. Das One-Time-Pad benötigt somit einen Schlüssel, welcher genauso lang ist wie die Nachricht selbst.

Es ist klar, warum diese Anforderung selbst bei mittelgroßen Nachrichten problematisch wird, zusätzlich muss nach jeder Übertragung ein neuer Schlüssel ausgetauscht werden. Die Idee von **OTP** ist gut, jedoch müssen für den praktischen Gebrauch einige Modifikationen vorgenommen werden. Es wird versucht, den Schlüsselstrom, welcher aus einer echt zufälligen Quelle stammt, durch die Ausgabe eines **CSPRNG** zu ersetzen, wobei der Schlüssel  $k$  den Startwert des **PRNG** bildet. Das Prinzip dieser Idee ist in [Abbildung 2.4](#) zu sehen. Zunächst ist wichtig festzuhalten, dass Stromchiffren nicht beweisbar sicher sind. Es ist allerdings so, dass alle in der Praxis verwendeten Algorithmen (Stromchiffren, Blockchiffren, asymmetrische Verfahren), keine informationstheoretische Sicherheit aufweisen (Paar und Pelzl [2010](#), S. 38). Um den Sicherheitsbegriff realistischer zu gestalten, wird der Angreifer in seiner Rechenleistung beschränkt, man hofft also, dass ein Verfahren berechenbarkeitstheoretisch sicher ist. Ein System heißt berechenbarkeitstheoretisch sicher ist, wenn es keinen bekannten Algorithmus gibt, welcher das System effizient brechen kann. In anderen Worten ausgedrückt: Es ist nicht effizient möglich, den Schlüssel anhand des Geheimtextes zu berechnen. Allgemein kann keine Aussage darüber getroffen werden, ob ein Problem mit bekannten Verfahren optimal gelöst wird, weshalb die Sicherheit eines Systems nie garantiert, sondern nur angenommen werden kann. Ein bekanntes Beispiel ist dass des RSA-Kryptosystems (asymmetrisches Verschlüsselungsverfahren), wessen Schlüssel berechnet werden kann, durch das Bestimmen der Primfaktorzerlegung einer großen natürlichen Zahl. Obwohl viele Faktorisierungsverfahren bekannt sind, ist es nicht klar, ob es andere Verfahren gibt, die das Problem effizienter lösen können. Im Fall von symmetrischen Verfahren nimmt man an, dass es keinen besseren Algorithmus gibt als die vollständige Schlüsselsuche.

# Kapitel 3

## Das RSA-Kryptosystem

Das RSA-Kryptosystem ist eines der bekanntesten und meist verbreiteten asymmetrischen Verschlüsselungsverfahren. Es ist benannt nach seinen drei Erfindern Ronald Rivest, Adi Shamir und Leonard Adleman, welche das Verfahren im Jahr 1977 veröffentlichten (Paar und Pelzl 2010, S. 173). Am Beispiel von RSA sollen in diesem Kapitel die Ideen der asymmetrischen Verschlüsselung (engl. *public-key-cryptography* oder *asymmetric cryptography*) vorgestellt werden. Es werden die nötigen zahlentheoretischen Begriffe eingeführt, um nachzuvollziehen warum die RSA-Verschlüsselung funktioniert und sicher ist.

### 3.1 Asymmetrische Verschlüsselung

Um die Idee der asymmetrischen Verschlüsselung besser zu verstehen, ist es hilfreich auf das Prinzip der symmetrischen Verfahren zurückzukommen ([Abbildung 3.1](#)).

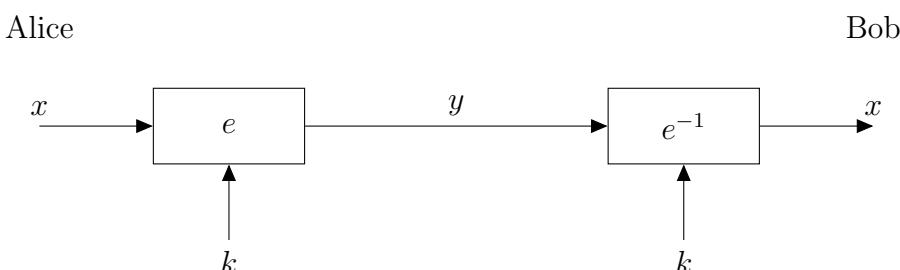


Abbildung 3.1: Das Prinzip der symmetrischen Verschlüsselung

Ein solches System hat zwei symmetrische Eigenschaften:

1. Derselbe geheime Schlüssel wird sowohl für die Verschlüsselung als auch Entschlüsselung verwendet.
2. Die Ver- und Entschlüsselungsfunktion sind sich sehr ähnlich (im Fall von **OTP** oder **DES** sind sie sogar gleich).

Verfahren wie **AES** sind sehr sicher und schnell. Ihre symmetrischen Eigenschaften führen jedoch auch zu Problemen und Einschränkungen, welche im Folgenden beschrieben werden:

**Schlüsselverteilungsproblem** Da beide Parteien denselben Schlüssel benötigen, muss dieser in irgendeiner Form untereinander verteilt werden. Der direkte Austausch über die Schnittstelle ist nicht möglich, denn die öffentliche Kommunikationsstrecke (unsicherer Kanal) ist nicht geschützt gegen Abhören.

**Anzahl der Schlüssel** Auch wenn das Schlüsselverteilungsproblem gelöst werden kann, entstehen schnell Probleme, da die Anzahl der Schlüssel in einem Netz mit zunehmender Teilnehmerzahl stark wächst. In einem Netz mit  $n$  Teilnehmern, wobei jeder Teilnehmer mit jedem verschlüsselt kommunizieren soll, gibt es

$$\binom{n}{2} = \frac{n!}{2! \cdot (n-2)!} = \frac{n \cdot (n-1)}{2} \approx \frac{n^2}{2}$$

verschiedene Schlüsselpaare und jeder Teilnehmer muss  $n - 1$  Schlüssel speichern.

*Beispiel 3.1.1.* In einem Netz mit 500 Teilnehmern gibt es bereits  $500 \cdot 499/2 = 124.750$  Schlüsselpaare und  $124.750 \cdot 2 = 249.500$  Schlüssel müssen verteilt werden.  $\triangle$

Das Problem ist auch bekannt als das  $n^2$ -Schlüsselverteilungsproblem (Paar und Pelzl 2010, S. 334–335).

**Verbindlichkeit** Es ist nur mit symmetrischer Verschlüsselung nicht möglich, einer dritten Person zu beweisen, welcher Gesprächsteilnehmer eine Nachricht erstellt hat. Es gibt jedoch viele Bereiche, in denen dieser Beweis wichtig ist, beispielsweise im Onlinehandel:

*Beispiel 3.1.2.* Alice betreibt einen Onlinehandel, sie muss beweisen können, dass ein Käufer Bob eine Bestellung getätigt hat, denn anderenfalls könnte er jederzeit behaupten, Alice hätte diese fälschlicherweise erstellt.  $\triangle$

Das Problem ist auch bekannt als Non-Repudiation (McCullagh und Cealli 2000) und kann gelöst werden durch digitale Signaturen.

Asymmetrische Verschlüsselungsverfahren bieten mögliche Lösungen zu den eben beschriebenen Problemen, um eine Nachricht zu verschlüsseln, ist es nicht mehr nötig, dass der Absender in Besitz von geheimer Information ist. Solch ein System wird realisiert, indem Bob einen Schlüssel veröffentlicht, welcher jedem Netzteilnehmer zur Verfügung steht und frei verwendet werden kann. Bob hat außerdem einen privaten Schlüssel, welchen nur er kennt. Bobs Schlüssel  $k$  besteht also aus einem öffentlichen Teil,  $k_{pub}$  (*key public*), und einem privaten Teil,  $k_{pr}$  (*key private*). Wichtig ist, dass eine Nachricht, welche mit Bobs öffentlichen Schlüssel verschlüsselt wurde, nur auch mit Bobs privaten Schlüssel wieder entschlüsselt werden kann. Ein einfaches Protokoll, welches nach diesem Prinzip arbeitet ist in [Abbildung 3.2](#) zu sehen.

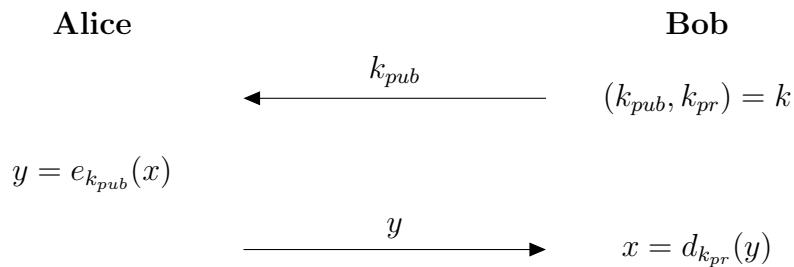


Abbildung 3.2: Das Prinzip der asymmetrischen Verschlüsselung

Ein Nachrichtenaustausch ist somit ohne sicheren Kanal möglich. Der oben beschriebene Ablauf kann nun so modifiziert, um einen symmetrischen Schlüssel auszutauschen, beispielsweise für [AES](#). Alice generiert einen symmetrischen Schlüssel und verschlüsselt ihn mit einem asymmetrischen Verfahren. Bob kann die Nachricht entschlüsseln und ist somit ebenfalls im Besitz des Schlüssels. Wie in [Abbildung 3.3](#) zu sehen ist, kann die restliche Kommunikation jetzt mit einem symmetrischen Verfahren gesichert werden. Es ist durchaus wünschenswert, nicht dauerhaft asymmetrisch zu verschlüsseln, da dies im Gegensatz zu dem symmetrischen Gegenstück sehr viel rechenintensiver ist.

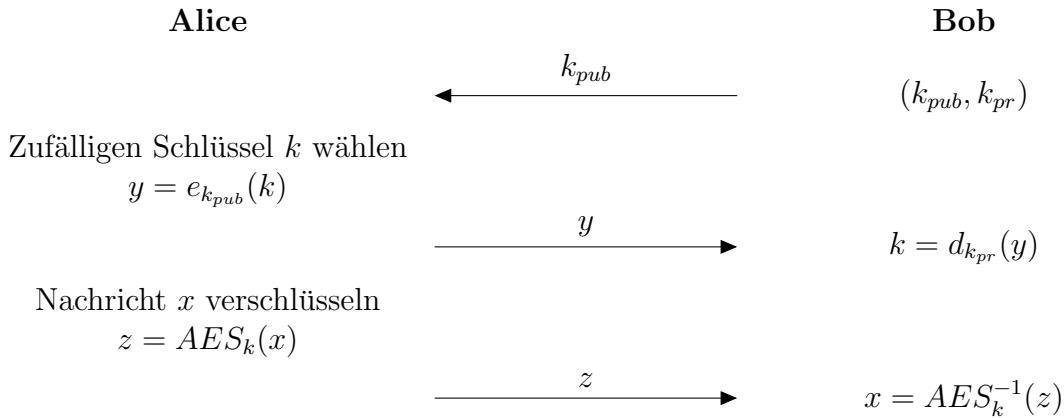


Abbildung 3.3: Schlüsselaustausch mit asymmetrischer Verschlüsselung

Asymmetrische Verfahren basieren alle auf einem zugrunde liegenden Prinzip: Der Einwegfunktion. Es kann folgende Definition gegeben werden (Paar und Pelzl 2010, S. 153):

**Definition 3.1.1** (Einwegfunktion). Eine Funktion  $f$  heißt Einwegfunktion, wenn gilt:

- Der Funktionswert  $y = f(x)$  ist komplexitätstheoretisch einfach berechenbar, d. h. die Laufzeit des Algorithmus wächst nicht stärker als eine Polynomfunktion (Polynomialzeit).
- Die Umkehrfunktion  $x = f^{-1}(y)$  ist komplexitätstheoretisch schwierig berechenbar, d. h. es gibt keinen bekannten Algorithmus, der das Problem in angemessener Zeit lösen kann, z. B. in 1000 Jahren.

Es gibt zwei Einwegfunktionen, welche in der Praxis häufig eingesetzt werden: Das Faktorisierungsproblem und die Umkehrung der Exponentiation mit ganzen Zahlen (Lösen des diskreten Logarithmus). RSA basiert auf dem Faktorisierungsproblem. Es ist einfach ein Produkt zu berechnen, jedoch ist es schwierig, eine Zahl zu faktorisieren. Der diskrete Logarithmus bildet die Einwegfunktion des Diffie-Hellman-Schlüsselaustausches. Es ist einfach eine Zahl zu potenzieren, jedoch ist es schwierig, den diskreten Logarithmus zu bestimmen.

## 3.2 Zahlentheoretische Grundlagen

Es werden in diesem Abschnitt die zahlentheoretischen Grundlagen beschrieben, um nachzuvollziehen warum die RSA-Verschlüsselung funktioniert.

### 3.2.1 Der Euklidische Algorithmus

Es wird begonnen mit dem Begriff des größten gemeinsamen Teilers. Es seien  $a, b \in \mathbb{N}$ , es bezeichnet  $T(a)$  die Menge aller Teiler von  $a$ , dann heißt jedes  $t \in T(a) \cap T(b)$  gemeinsamer Teiler von  $a$  und  $b$ .

Beispiel 3.2.1.

$$\begin{aligned}T(6) &= \{1, 2, 3, 6\} \\T(4) &= \{1, 2, 4\} \\T(6) \cap T(4) &= \{1, 2\}\end{aligned}\quad \triangle$$

**Definition 3.2.1** (Größter gemeinsamer Teiler). Es seien  $a, b \in \mathbb{N}$ . Es sei  $T = T(a) \cap T(b)$ . Es sei  $g \in T$  und es gelte für alle  $t \in T$ :  $t \leq g$ . Dann heißt  $g$  größter gemeinsamer Teiler von  $a$  und  $b$ . Man schreibt auch  $g = \text{ggT}(a, b)$ .

**Rechenregeln für ggT:** Für  $a, b \in \mathbb{N}$  gilt (Spannagle 2011):

1.  $\text{ggT}(a, a) = a$
2.  $\text{ggT}(a, 1) = 1$
3.  $\text{ggT}(a, 0) = a$
4.  $\text{ggT}(a, b) = \text{ggT}(b, a)$
5.  $\text{ggT}(a, b) = \text{ggT}(a - b, b)$

*Beweis.*

1. Wegen  $T(a) \cap T(a) = T(a)$ .
2. Wegen  $T(1) = \{1\}$  und  $T(a) \cap \{1\} = \{1\}$ .
3. Wegen  $T(0) = \mathbb{N}$  und  $T(a) \cap \mathbb{N} = T(a)$ .

4. Wegen der Kommutativität der Schnittmenge  $T(a) \cap T(b) = T(b) \cap T(a)$ .
5. Zu zeigen:  $ggT(a, b) = ggT(a - b, b)$  geschrieben als  $g = g'$ .

$$g \mid a \wedge g \mid b \Rightarrow g \mid a - b \Rightarrow g \in T(a - b) \cap T(b).$$

$g$  ist ein Teiler von  $a - b$  und  $b$ .

Ist  $g$  der größte gemeinsame Teiler? Es sei  $h \in T(a - b) \cap T(b)$  und  $h > g$ :

$$h \mid a - b \wedge h \mid b \Rightarrow h \mid a.$$

$h \mid a$  und  $h \mid b$  und  $h > g$  führt zum Widerspruch.

□

Beispiel 3.2.2. Es seien  $a = 132$  und  $b = 51$ , dann gilt:

$$\begin{aligned} ggT(132, 51) &\stackrel{(5)}{=} ggT(132 - 51, 51) = ggT(81, 51) \stackrel{(5)}{=} ggT(30, 51) \stackrel{(4)}{=} ggT(51, 30) \stackrel{(5)}{=} \\ ggT(21, 30) &\stackrel{(4)}{=} ggT(30, 21) \stackrel{(5)}{=} ggT(9, 21) \stackrel{(4)}{=} ggT(21, 9) \stackrel{(5)}{=} ggT(12, 9) \stackrel{(5)}{=} \\ ggT(3, 9) &\stackrel{(4)}{=} ggT(9, 3) \stackrel{(5)}{=} ggT(6, 3) \stackrel{(5)}{=} ggT(3, 3) \stackrel{(1)}{=} 3 \end{aligned} \quad \triangle$$

Es ist zu sehen, dass sich Rechenregel (5) iterativ anwenden lässt:

$$ggT(a, b) = ggT(a - b, b) = ggT(a - 2b, b) = \dots = ggT(a - mb, b)$$

Wird ein maximales  $m$  gewählt, mit der Bedingung  $(a - mb) > 0$ , kann der Algorithmus in minimalen Schritten durchgeführt werden. Dies ist der Fall für die Division mit Rest:

$$ggT(a, b) = ggT(a \bmod b, b)$$

Beispiel 3.2.3. Es seien  $a = 132$  und  $b = 51$ , dann gilt:

$$\begin{array}{ll} 132 = 2 \cdot 51 + 30 & ggT(132, 51) = ggT(51, 30) \\ 51 = 1 \cdot 30 + 21 & ggT(51, 30) = ggT(30, 21) \\ 30 = 1 \cdot 21 + 9 & ggT(30, 21) = ggT(21, 9) \\ 21 = 2 \cdot 9 + 3 & ggT(21, 9) = ggT(9, 3) \\ 9 = 3 \cdot 3 + 0 & ggT(9, 3) = ggT(3, 0) = 3 \end{array} \quad \triangle$$

In allgemeiner Form können die eben gezeigten Schritte folgendermaßen beschrieben werden:

$$\begin{aligned}
 a_i &= q_i \cdot b_i + r_i \\
 a_{i+1} &= q_{i+1} \cdot b_{i+1} + r_{i+1} \\
 \text{mit } a_{i+1} &= b_i \\
 b_{i+1} &= r_i = a_i - q_i \cdot b_i
 \end{aligned} \tag{3.1}$$

Eine rekursive Implementierung des Euklidischen Algorithmus ist in [Algorithmus 3.1](#) zu sehen (engl. *greatest common divisor* (gcd)). Das Verfahren terminiert nachdem das erste Mal ein Rest von null berechnet wurde.

#### **Algorithmus 3.1 : Euklidischer Algorithmus**

```

Input : zwei ganze Zahlen  $a$  und  $b$ 
Output :  $ggT(a, b)$ 
def  $ggT(a, b)$ 
begin
     $r \leftarrow a \bmod b$ 
    if  $r = 0$  then
         $\mid$  return  $b$ 
    end
    return  $ggT(a, b)$ 
end

```

### 3.2.2 Der erweiterte Euklidische Algorithmus

Es wurde gezeigt, dass der größte gemeinsame Teiler zweier Zahlen durch das Reduzieren der Operanden ermittelt werden kann. In der Kryptografie ist das Finden dieser Zahl allerdings nicht das Hauptanwendungsgebiet des Algorithmus. Es stellt sich heraus, dass eine Erweiterung des Euklidischen Algorithmus verwendet werden kann, um multiplikative Inverse Modulo  $m$  zu ermitteln. Betrachtet man den Ring  $\mathbb{Z}_m$ , dann ist das Inverse  $a^{-1}$  einer Zahl  $a \in \mathbb{Z}_m$  gegeben durch die folgende Beziehung:

$$a \cdot a^{-1} \equiv 1 \pmod{m} \tag{3.2}$$

Das multiplikative Inverse existiert nicht für alle Elemente. Es kann aber eine Aussage darüber getroffen werden, wann es existiert. Ein Element  $a \in \mathbb{Z}_m$  besitzt genau dann ein Inverses, wenn gilt  $ggT(a, m) = 1$ . Zwei Zahlen  $a$  und  $b$  für die gilt  $ggT(a, b) = 1$  nennt man teilerfremd oder relativ Prim (engl. *relatively prime* oder *coprime*). Es wird hierfür das Symbol  $a \perp b$  verwendet. Der Erweiterte Euklidische Algorithmus berechnet eine Linearkombination der folgenden Form, welche auch als diophantische Gleichung bezeichnet wird (Paar und Pelzl 2010, S. 160) (Spannagle 2012a):

$$x \cdot a + y \cdot b = ggT(a, b) \quad (3.3)$$

Beispiel 3.2.4. Es seien erneut  $a = 132$  und  $b = 51$ , es wird zuerst der  $ggT(132, 51)$  mit dem Euklidischen Algorithmus bestimmt:

$$\begin{aligned} 132 &= 2 \cdot 51 + 30 \\ 51 &= 1 \cdot 30 + 21 \\ 30 &= 1 \cdot 21 + 9 \\ 21 &= 2 \cdot 9 + 3 \\ 9 &= 3 \cdot 3 + 0 \end{aligned}$$

Der  $ggT(132, 51)$  ist mit 3 bestimmt. Es soll nun versucht werden, ein  $x$  und  $y$  zu finden, sodass  $3 = x \cdot 132 + y \cdot 51$ . Aus der vorletzten Zeile weiß man:

$$3 = 21 - 2 \cdot 9$$

Wie kam die 9 zustande? Aus der Zeile darüber mit  $9 = 30 - 21$ . Eingesetzt und zusammengefasst:

$$3 = 21 - 2 \cdot (30 - 21) = 21 - 2 \cdot 30 + 2 \cdot 21 = -2 \cdot 30 + 3 \cdot 21$$

Wie kam die 21 zustande? Erneut aus der Zeile darüber mit  $21 = 51 - 30$ . Eingesetzt,

zusammengefasst und für die letzten beiden Zeilen fortgeführt:

$$3 = -2 \cdot 30 + 3 \cdot (51 - 30) = -2 \cdot 30 + 3 \cdot 51 - 3 \cdot 30 = 3 \cdot 51 - 5 \cdot 30$$

$$3 = 3 \cdot 51 - 5 \cdot (132 - 2 \cdot 51) = 3 \cdot 51 - 5 \cdot 132 + 10 \cdot 51 = -5 \cdot 132 + 13 \cdot 51$$

Man erhält die gewünschte Gleichung.  $\triangle$

Das Verfahren kann erneut allgemein betrachtet werden. Durch Rückwärtsarbeiten erhält man verschiedene Gleichungen der folgenden Form:

$$ggT(a, b) = x_i \cdot a_i + y_i \cdot b_i$$

$$ggT(a, b) = x_{i+1} \cdot \boxed{a_{i+1}} + y_{i+1} \cdot \boxed{b_{i+1}}$$

Außerdem kennen wir die Gleichungen aus (3.1):

$$a_i = q_i \cdot b_i + r_i$$

$$a_{i+1} = q_{i+1} \cdot b_{i+1} + r_{i+1}$$

$$\text{mit } a_{i+1} = b_i$$

$$b_{i+1} = r_i = a_i - q_i \cdot b_i$$

Einsetzen und umformen:

$$\begin{aligned} ggT(a, b) &= x_i \cdot a_i + y_i \cdot b_i \\ &= x_i \cdot (q_i \cdot b_i + r_i) + y_i \cdot a_{i+1} \\ &= x_i \cdot (q_i \cdot a_{i+1} + b_{i+1}) + y_i \cdot a_{i+1} \\ &= x_i \cdot q_i \cdot a_{i+1} + x_i \cdot b_{i+1} + y_i \cdot a_{i+1} \\ &= (x_i \cdot q_i + y_i) \cdot \boxed{a_{i+1}} + x_i \cdot \boxed{b_{i+1}} \end{aligned}$$

Es ergeben sich die folgenden Regeln:

$$x_i = y_{i+1} \quad (3.4)$$

$$\begin{aligned} x_i \cdot q_i + y_i &= x_{i+1} \\ y_i &= x_{i+1} - x_i \cdot q_i \\ y_i &= x_{i+1} - y_{i+1} \cdot q_i \end{aligned} \quad (3.5)$$

Dies resultiert im erweiterten Euklidischen Algorithmus, welcher in Tabellenform einfach und schnell durchführbar ist:

Tabelle 3.1: Erweiterter Euklidischer Algorithmus

$i$	$a$	$b$	$q$	$r$	$x$	$y$	Kontrolle
1	132	51	2	30	-5	$3 + 5 \cdot 2 = 13$	$3 = -5 \cdot 132 + 13 \cdot 51$
2	51	30	1	21	3	$-2 - 3 \cdot 1 = -5$	$3 = 3 \cdot 51 - 5 \cdot 30$
3	30	21	1	9	-2	$1 + 2 \cdot 1 = 3$	$3 = -2 \cdot 30 + 3 \cdot 21$
4	21	9	2	3	1	$0 - 1 \cdot 2 = -2$	$3 = 1 \cdot 21 - 2 \cdot 9$
5	9	3	3	0	0	1	$3 = 0 \cdot 9 + 1 \cdot 3$

Es soll nun gezeigt werden, wie der erweiterte Euklidischen Algorithmus verwendet werden kann, um multiplikative Inverse zu berechnen. Es soll das Inverse  $a \bmod m$  bestimmt werden, wobei  $m > a$ . Das Inverse existiert genau dann, wenn  $\text{ggT}(m, a) = 1$ , dies bedeutet es gibt eine Gleichung der Form  $x \cdot m + y \cdot a = 1$ . Stellt man diese Gleichung als Kongruenzrelation Modulo  $m$  dar, erhalten wir:

$$\begin{aligned} x \cdot m + y \cdot a &= 1 \\ x \cdot m + y \cdot a &\equiv 1 \pmod{m} \\ x \cdot 0 + y \cdot a &\equiv 1 \pmod{m} \\ a \cdot y &\equiv 1 \pmod{m} \end{aligned}$$

Die letzte Zeile ist genau die Definition des Inversen (3.2), welches mit  $y$  bestimmt wurde.

Beispiel 3.2.5. Es soll das Inverse  $21 \cdot 21^{-1} \equiv 1 \pmod{89}$  bestimmt werden. Es gilt  $21 \perp 89$  und es gibt somit eine Gleichung  $x \cdot 89 + y \cdot 21 = 1$ . Der Euklidische Algorithmus kann

tabellarisch durchgeführt werden:

$a$	$b$	$q$	$r$	$x$	$y$
89	21	4	5	-4	$1 + 4 \cdot 4 = 17$
21	5	4	1	1	$0 - 1 \cdot 4 = -4$
5	1	5	0	0	1

Wir erhalten den größten gemeinsamen Teiler als Linearkombination:

$$-4 \cdot 89 + 17 \cdot 21 = 1$$

Es folgt hieraus: Das Inverse von 21 in  $\mathbb{Z}_{89}$  beträgt 17. Dieses Ergebnis kann durch nachrechnen verifiziert werden:

$$17 \cdot 21 = 357 \equiv 1 \pmod{89} \quad \triangle$$

Eine rekursive Implementierung des erweiterten Euklidischen Algorithmus ist in [Algorithmus 3.2](#) zu sehen. Das Symbol  $\div$  bezeichnet die Ganzzahldivision zweier Zahlen  $a$  und  $b$ . Wie zuvor terminiert das Verfahren nachdem das erste Mal ein Rest von null berechnet wurde.

<b>Algorithmus 3.2 :</b> Erweiterter Euklidischer Algorithmus	
<b>Input :</b>	zwei ganze Zahlen $a$ und $b$
<b>Output :</b>	$ggT(a, b)$ als auch $x$ und $y$ , sodass $ggT(a, b) = x \cdot a + y \cdot b$
<b>def</b>	$ggT(a, b)$
<b>begin</b>	
	$r \leftarrow a \bmod b$
	$q \leftarrow a \div b$
	<b>if</b> $r = 0$ <b>then</b>
	<b>return</b> $(0, 1, b)$
	<b>end</b>
	$(x, y, ggt) \leftarrow ggT(b, r)$
	<b>return</b> $(y, x - y \cdot q, ggt)$
<b>end</b>	

### 3.2.3 Die Eulersche Phi-Funktion

Die Eulersche Phi-Funktion befasst sich mit dem auf den ersten Blick seltsamen Problem, die Anzahl an teilerfremden Zahlen in einer Menge zu finden. Die Regeln und Sätze welche von der zahlentheoretischen Funktion abgeleitet werden können, sind jedoch sehr hilfreich in der asymmetrischen Verschlüsselung und insbesondere für das RSA-Verfahren. Die Eulersche Phi-Funktion ist folgendermaßen definiert (Paar und Pelzl 2010, S. 165) (Spannagle 2012b):

**Definition 3.2.2** (Eulersche Phi-Funktion). Die Anzahl der zu  $m \in \mathbb{N}^\times$  teilerfremden Zahlen aus  $\{1, 2, \dots, m\}$ , ist gekennzeichnet als  $\varphi(m)$  und heißt Eulersche Phi-Funktion.  $\varphi(m)$  kann formal beschrieben werden als die Mächtigkeit der Menge:

$$\varphi(m) = |\{x \in \mathbb{N} \mid 1 \leq x \leq m \wedge ggT(m, x) = 1\}|$$

Beispiel 3.2.6.

$$\varphi(5) = |\{1, 2, 3, 4\}| = 4$$

$$\varphi(6) = |\{1, 5\}| = 2$$

$$\varphi(20) = |\{1, 3, 7, 9, 11, 13, 17, 19\}| = 8 \quad \triangle$$

Die naive Berechnung der Eulerschen Phi-Funktion, alle Zahlen zu durchlaufen und den größten gemeinsamen Teiler zu bestimmen, ist sehr langsam und für große Zahlen, wie sie in der asymmetrischen Verschlüsselung verwendet werden, nicht möglich. Es existiert jedoch eine Beziehung, mit welcher  $\varphi(m)$  bestimmt werden kann, unter der Bedingung, dass die Primfaktorzerlegung von  $m$  bekannt ist. Diese lässt sich aus den folgenden Sätzen ableiten, welche hier im einzelnen nicht noch einmal bewiesen werden sollen (ebd.):

1. Es sei  $p \in \mathbb{P}$ , dann gilt:  $\varphi(p) = p - 1$ .
2. Es sei  $p \in \mathbb{P}$ , dann gilt:  $\varphi(p^n) = p^n - p^{n-1}$ .
3. Es sei  $ggT(n, m) = 1$ , dann gilt:  $\varphi(n \cdot m) = \varphi(n) \cdot \varphi(m)$ .

**Satz 3.2.1** (Formel zur Eulerschen Phi-Funktion). *Es sei*

$$n = p_1^{m_1} \cdot p_2^{m_2} \cdot \dots \cdot p_r^{m_r} = \prod_{i=1}^r p_i^{m_i}$$

die Primfaktorzerlegung einer natürlichen Zahl mit  $p_i$  unterschiedlichen Primzahlen und Exponenten  $m_1 \geq 1, \dots, m_r \geq 1$ . Dann ist:

$$\varphi(n) = \prod_{i=1}^r p_i^{m_i} - p_i^{m_i-1} = n \cdot \prod_{i=1}^r 1 - \frac{1}{p_i}$$

*Beweis.* Da die Primfaktoren einer Zahl teilerfremd sind, kann nach Regel (3) und (2) geschrieben werden:

$$\begin{aligned} \varphi(n) &\stackrel{(3)}{=} \prod_{i=1}^r \varphi(p_i^{m_i}) \stackrel{(2)}{=} \prod_{i=1}^r p_i^{m_i} - p_i^{m_i-1} = \prod_{i=1}^r p_i^{m_i} \cdot \left(1 - \frac{1}{p_i}\right) \\ &= \prod_{i=1}^r p_i^{m_i} \cdot \prod_{i=1}^r 1 - \frac{1}{p_i} \\ &= n \cdot \prod_{i=1}^r 1 - \frac{1}{p_i} \end{aligned} \quad \square$$

Beispiel 3.2.7.

$$\begin{aligned} \varphi(6 = 2 \cdot 3) &= 6 \cdot \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{3}\right) = 6 \cdot \frac{2}{6} = 2 \\ \varphi(20 = 2^2 \cdot 5) &= 20 \cdot \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{5}\right) = 20 \cdot \frac{4}{10} = 8 \\ \varphi(1000 = 2^3 \cdot 5^3) &= 1000 \cdot \frac{4}{10} = 400 \end{aligned} \quad \triangle$$

### 3.2.4 Satz von Euler und Fermat

Es werden im Folgenden zwei Sätze bewiesen von Euler und Fermat, die in fast allen asymmetrischen Verschlüsselungsverfahren eine sehr wesentliche Bedeutung haben. Es muss zuvor jedoch noch eine Eigenschaft der Restklassen festgehalten werden:

**Satz 3.2.2** (Eindeutigkeit des ggT von Repräsentanten einer Restklasse und dem Modul  $m$ ). Für einen Modul  $m$  und zwei Repräsentanten  $a, b \in \mathbb{N}$  derselben Restklasse  $|a|_m = |b|_m$  gilt:

$$ggT(a, m) = ggT(b, m)$$

*Beweis.* Es ist  $b = a + tm$  mit  $t \in \mathbb{Z}$  nach Definition gegeben. Es sind

$$\begin{aligned} g_1 &= ggT(a, m) = x \cdot \boxed{a} + y \cdot \boxed{m} \\ g_2 &= ggT(b, m) = p \cdot \boxed{b} + q \cdot \boxed{m} \end{aligned}$$

mit  $x, y, p, q \in \mathbb{Z}$ . Es gilt außerdem allgemeiner: Sei  $c = x \cdot a + y \cdot m$ , dann ist  $c$  ein Vielfaches von  $ggT(a, m)$ . Man schreibt:

$$\begin{aligned} g_1 &= xa + ym & g_2 &= pb + qm \\ &= x(b - tm) + ym & &= p(a + tm) + qm \\ &= xb - xtm + ym & &= pa + ptm + qm \\ &= x \boxed{b} - (xt + y) \boxed{m} & &= p \boxed{a} + (pt + q) \boxed{m} \\ \Rightarrow g_1 &\mid g_2 & \Rightarrow g_2 &\mid g_1 \end{aligned}$$

Aus den beiden letzten Zeile folgen:  $g_1 = g_2$ . □

**Satz 3.2.3** (Der Satz von Euler). *Es seien  $a, m \in \mathbb{N}^\times$  mit  $a \perp m$ , dann gilt:*

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

*Beweis.* Mit gleichem Grundgedanken in (Spannagle 2012b) und (Remmert und Ullrich 2008, S. 187–188).

Es ist  $n := \varphi(m)$ . Es seien  $x_1, x_2, \dots, x_n$  die  $n$  verschiedenen zu  $m$  teilerfremden Zahlen aus der Menge  $\{1, 2, \dots, m\}$ . Mit einer vorgegeben Zahl  $a \in \mathbb{N}^\times \wedge a \perp m$  bilden wir die Produkte  $ax_1, ax_2, \dots, ax_n$ . Wir betrachten ein Beispiel um eine Vermutung zu bestätigen:

Beispiel 3.2.8. Es ist  $\varphi(8) = |\{1, 3, 5, 7\}| = 4$ . Wir wählen  $a = 3$  und bilden in  $\mathbb{Z}_8$  die Produkte:  $3 \cdot \{1, 3, 5, 7\} = \{3, 9, 15, 21\} \equiv \{3, 1, 7, 5\} \pmod{8}$ . Die Multiplikation von  $a$  erzeugt eine Permutation der Menge  $\{1, 3, 5, 7\}$ . △

Es sind  $|x_1|_m, \dots, |x_n|_m$  genau alle zu  $m$  teilerfremden Restklassen.<sup>1</sup> Es gilt  $x_i \perp m$  und  $a \perp m$ , wonach auch das Produkt  $ax_i$  teilerfremd zu  $m$  sein muss.

---

<sup>1</sup>Nach [Satz 3.2.2](#) wissen wir, ist eine Zahl einer Restklasse teilerfremd zu  $m$ , gilt dies auch für alle anderen Zahlen der Restklasse.

Es folgt  $ax_i \bmod m = x_j$  und wir halten fest:

$$\forall ax_i \exists x_j : ax_i \equiv x_j \pmod{m} \quad (*)$$

Es bleibt zu zeigen, dass in  $(*)$  keine Doppelungen entstehen, also alle  $ax_i$  Teil unterschiedlicher Restklassen sind. Es muss gelten:

$$ax_i \not\equiv ax_j \pmod{m} \quad i \neq j$$

Angenommen  $ax_i \equiv ax_j \pmod{m}$  für  $i \neq j$ , dann gilt nach der Kürzungsregel<sup>2</sup>  $x_i \equiv x_j \pmod{m}$ , was nicht sein kann. Wegen  $x_1, x_2, \dots, x_n \perp m$  liefert die Kürzungsregel, wenn man noch  $n = \varphi(m)$  beachtet, die ursprüngliche Behauptung:

$$\begin{aligned} ax_1 \cdot ax_2 \cdot \dots \cdot ax_n &\equiv x_1 \cdot x_2 \cdot \dots \cdot x_n \pmod{m} \\ a^{\varphi(m)} &\equiv 1 \pmod{m} \end{aligned}$$

□

Der Kleine Satz von Fermat kann jetzt als Spezialform des Satzes von Euler (mit  $m = p$ ) direkt aufgeschrieben werden:

**Satz 3.2.4** (Kleiner Satz von Fermat). *Es sei  $a \in \mathbb{N}^\times$  und  $p \in \mathbb{P}$  mit  $a \perp p$ , dann gilt:*

$$a^{p-1} \equiv 1 \pmod{p}$$

Der Kleine Satz von Fermat wird in der Kryptografie unter anderem dafür verwendet, Primzahltests durchzuführen.

### 3.3 Das RSA-Verfahren

Das RSA-Verfahren basiert auf der Annahme, dass es nicht effizient möglich ist, die Primfaktorzerlegung einer natürlichen Zahl zu bestimmen. Die Generierung des privaten und öffentlichen Schlüssels kann in die folgenden fünf Schritte unterteilt werden (Paar und Pelzl 2010, S. 176):

---

<sup>2</sup>Da das Inverse wegen  $a \perp m$  existiert, kann mit diesem multipliziert werden und die  $a$ 's fallen weg.

**Definition 3.3.1** (RSA Schlüsselgenerierung).

1. Wähle zwei sehr große Primzahlen  $p$  und  $q$ , beispielsweise mit einer Länge von 512 oder 1024 Bit.
2. Berechne  $N = p \cdot q$ .
3. Berechne  $\varphi(N) = \varphi(p \cdot q) = \varphi(p) \cdot \varphi(q) = (p - 1) \cdot (q - 1)$ .
4. Wähle eine Zahl  $e$  mit  $1 \leq e < \varphi(N)$  und  $e \perp \varphi(N)$ .
5. Berechne  $d$  mit  $e \cdot d \equiv 1 \pmod{\varphi(N)}$  und dem erweiterten Euklidischen Algorithmus.
6. Vernichte  $p$  und  $q$ .
7. Jetzt bilden  $(N, e)$  den öffentlichen und  $(N, d)$  den privaten Schlüssel.

Sind  $p$  und  $q$  entsprechend groß, ist es nicht in annehmbarer Zeit möglich, die Faktorisierung von  $N$  zu bestimmen und damit auch  $\varphi(N)$ . Die Ver- und Entschlüsselung mit RSA gestaltet sich sehr einfach:

**Definition 3.3.2** (Das RSA-Verfahren). Es sei  $k_{pub} = (N, e)$  und  $k_{pr} = (N, d)$  der öffentliche und private Schlüssel. Dann gilt für  $x, y \in \mathbb{Z}_N$ :

**Verschlüsselung:**  $y = e_{k_{pub}}(x) \equiv x^e \pmod{N}$

**Entschlüsselung:**  $x = d_{k_{pr}}(y) \equiv y^d \pmod{N}$

Bevor die Korrektheit von RSA bewiesen wird, schreiben wir noch einen Satz auf, der im Beweis verwendet werden kann:

**Satz 3.3.1.**  *$m$  ist eine zusammengesetzte Zahl der Form  $m = k_1 \cdot k_2 \cdot \dots \cdot k_n$  und es gilt  $a \equiv b \pmod{m}$ , dann gilt auch:*

$$a \equiv b \pmod{k_1} \quad \dots \quad a \equiv b \pmod{k_n}$$

*Beweis.* Die eben gezeigten Kongruenzen lassen sich als Gleichungen schreiben:

$$a - b = t_0 m \quad a - b = t_1 k_1 \quad \dots \quad a - b = t_n k_n$$

Wobei  $t_i$  irgendwelche ganzen Zahlen sind. Es folgt:  $t_0 m = t_1 k_1 = \dots = t_n k_n$ . Man wählt jetzt  $t_1 = \frac{t_0 m}{k_1} \quad \dots \quad t_n = \frac{t_0 m}{k_n}$  und findet immer eine Lösung.  $\square$

Es soll nun Bewiesen werden, warum das RSA-Verfahren funktioniert:

*Beweis.* Die Entschlüsselung des Geheimtextes muss erneut den Klartext ergeben. Zu zeigen:

$$x \equiv x^{ed} \pmod{N} \quad (*)$$

Wir wissen wegen  $ed \equiv 1 \pmod{\varphi(N)}$ :

$$ed = t\varphi(N) + 1 \quad t \in \mathbb{Z} \quad (**)$$

Um  $x \equiv x^{ed} \pmod{pq}$  zu zeigen, reicht es nach (3.3.1) die Faktoren einzeln zu betrachten.

1. Um  $x \equiv x^{ed} \pmod{p}$  zu beweisen, betrachten wir zwei Fälle:

(a) Trivial für  $p \mid x$ .

(b) Es gilt  $p \nmid x$  und daher auch  $p \perp x$ , einsetzen von (\*\*) in (\*) und umformen:

$$x^{ed} \equiv x^{t\varphi(N)+1} \equiv x^{t\varphi(N)} \cdot x \equiv x^{t(p-1)(q-1)} \cdot x \equiv (x^{p-1})^{t(q-1)} \cdot x \pmod{p}$$

Nach dem Kleinen Satz von Fermat (3.2.4) ist jetzt zu sehen:

$$(x^{p-1})^{t(q-1)} \cdot x \equiv 1^{t(q-1)} \cdot x \equiv x \pmod{p}$$

2. Die Überlegung für  $q$  kann analog durchgeführt werden. Es müssen hierfür alle  $p$  und  $q$  vertauscht werden.  $\square$

Beispiel 3.3.1.

1. Wähle  $p = 3$  und  $q = 11$ .
2. Berechne  $N = 33$ .
3. Berechne  $\varphi(33) = (3 - 1) \cdot (11 - 1) = 20$ .
4. Wähle  $e = 3$ .
5. Berechne  $d$  mit dem erweiterten Euklidischen Algorithmus:

$a$	$b$	$q$	$r$	$x$	$y$
20	3	6	2	-1	$1 + 1 \cdot 6 = 7 = d$
3	2	1	1	1	$0 - 1 \cdot 1 = -1$
2	1	2	0	0	1

6. Alice kennt den öffentlichen Schlüssel  $(N, e) = (33, 3)$  und kann eine Nachricht  $x = 4$  verschlüsseln mit  $x^e = 4^3 \equiv 31 \pmod{33}$ .
7. Bob kennt den privaten Schlüssel  $(N, d) = (33, 7)$  und kann die Nachricht entschlüsseln mit  $y^d = 31^7 \equiv (-2)^7 \equiv (-2)^5 \cdot 2^2 \equiv 1 \cdot 4 \equiv 4 \pmod{33}$ .  $\triangle$

# Kapitel 4

## Ein Steganografischer Algorithmus

Die Steganografie bezeichnet eine weitere Methode die Vertraulichkeit eines Informationsaustausches zu gewährleisten. Es wird das Ziel verfolgt, eine Nachricht in einer für den Computer zugänglichen Trägerdatei (engl. *cover media*) zu verstecken, sodass eine weitere Person die Existenz einer geheimen Botschaft gar nicht erst vermuten würde. Trägerdateien, unempfindlich gegenüber kleinen Änderungen in den Daten eignen sich besonders gut für die Anwendung steganografischer Verfahren. Digitale Bilddateien sowie Audio- und Videodateien sind sehr gute Trägermedien, da ihre Daten ein ganz natürliches Rauschen aufweisen. In diesem Kapitel soll ein Algorithmus vorgestellt werden, welcher eine beliebig lange Nachricht in einem Bild versteckt und dabei versucht, die visuelle Qualität des Urbilds zu bewahren. Zusätzlich wird auf eine Anwendung eingegangen, welche die beschriebenen Ideen umsetzt und einen Nachrichtenaustausch über in Bildern versteckten Informationen ermöglicht.

### 4.1 Modifikation von Bilddateien

Eine digitale Bilddatei besteht aus einer zweidimensionalen Anordnung von Pixel, wobei jeder eine bestimmte Farbe annehmen kann. Farben können unterschiedlich dargestellt werden, dass in der Bildwiedergabe am häufigsten verwendete Modell ist der RGB-Farbraum. Die Farbwahrnehmung des menschlichen Auges kann durch das additive Mischen der drei Grundfarben Rot, Grün und Blau (RGB) nachgebildet werden (Cimato und Yang 2017, S. 32–40). In computerorientierten Anwendungen werden hierfür pro Farbka-

nal Zahlenwerte zwischen 0 und 255 gespeichert, es gilt je größer der Wert desto heller die Farbe. Die Kombinationen (255, 0, 0), (0, 255, 0) und (0, 0, 255) beschreiben jeweils die Grundfarben Rot, Grün und Blau. Das Mischen aller Farben (255, 255, 255) ergibt die Farbe Weiß und das Hinzufügen gar keines Lichts (0, 0, 0) resultiert in Schwarz. Kombinationen mit gleicher Intensität (100, 100, 100) werden als Grauton wahrgenommen. Pro Pixel müssen in einem Bild also drei Byte an Information gespeichert werden, dies verspricht ein großes Potenzial, wenn es darum geht, unentdeckt Information zu verbergen. Ein einfaches und effektives Verfahren ist das Überschreiben der niederwertigsten Bit (engl. *least significant bit (LSB)*) im Farbkanal durch das zu versteckende Signal. Das Anpassen der **LSB** verändert den Farbwert nur minimal und die kleinen Änderungen in den Zahlen werden nur durch das Betrachten des veränderten Bilds nicht zu erkennen sein.

## 4.2 Wie stark kann ein Bild angepasst werden?

Es soll nun abgeschätzt werden, wie stark ein Bild verändert werden kann, ohne dass die Qualität des Ergebnisses sichtbar beeinflusst wird. Es seien  $b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$  die acht Bit eines Farbkanals, es soll für jeden Kanal der maximale Fehler betrachtet werden, welcher entstehen kann, wenn die  $n$  niederwertigsten Bit durch eine Nachricht ersetzt werden. Der neue Farbwert einschließlich Fehler wird nach (4.1) in Bezug auf  $n$  bestimmt:

$$a(n) = \sum_{i=0}^{n-1} b_i \cdot 2^i \quad b_{i,n} = \begin{cases} b_i & \text{wenn } i \geq n \\ 1 & \text{wenn } a(n) \leq \lfloor \frac{1}{2} \cdot \sum_{i=0}^{n-1} 2^i \rfloor \\ 0 & \text{sonst} \end{cases} \quad (4.1)$$

**Abbildung 4.1** zeigt die Auswirkung der Veränderung auf die Bildqualität eines Farbbilds für Fehlerparameter  $0 \leq n \leq 8$ . Es kann die durchaus vielversprechende Beobachtung gemacht werden, dass Änderungen bis hin zur vierten Stelle im Farbkanal nur schwer und ohne Vergleich mit der Originaldatei wahrscheinlich nicht erkannt werden würden.

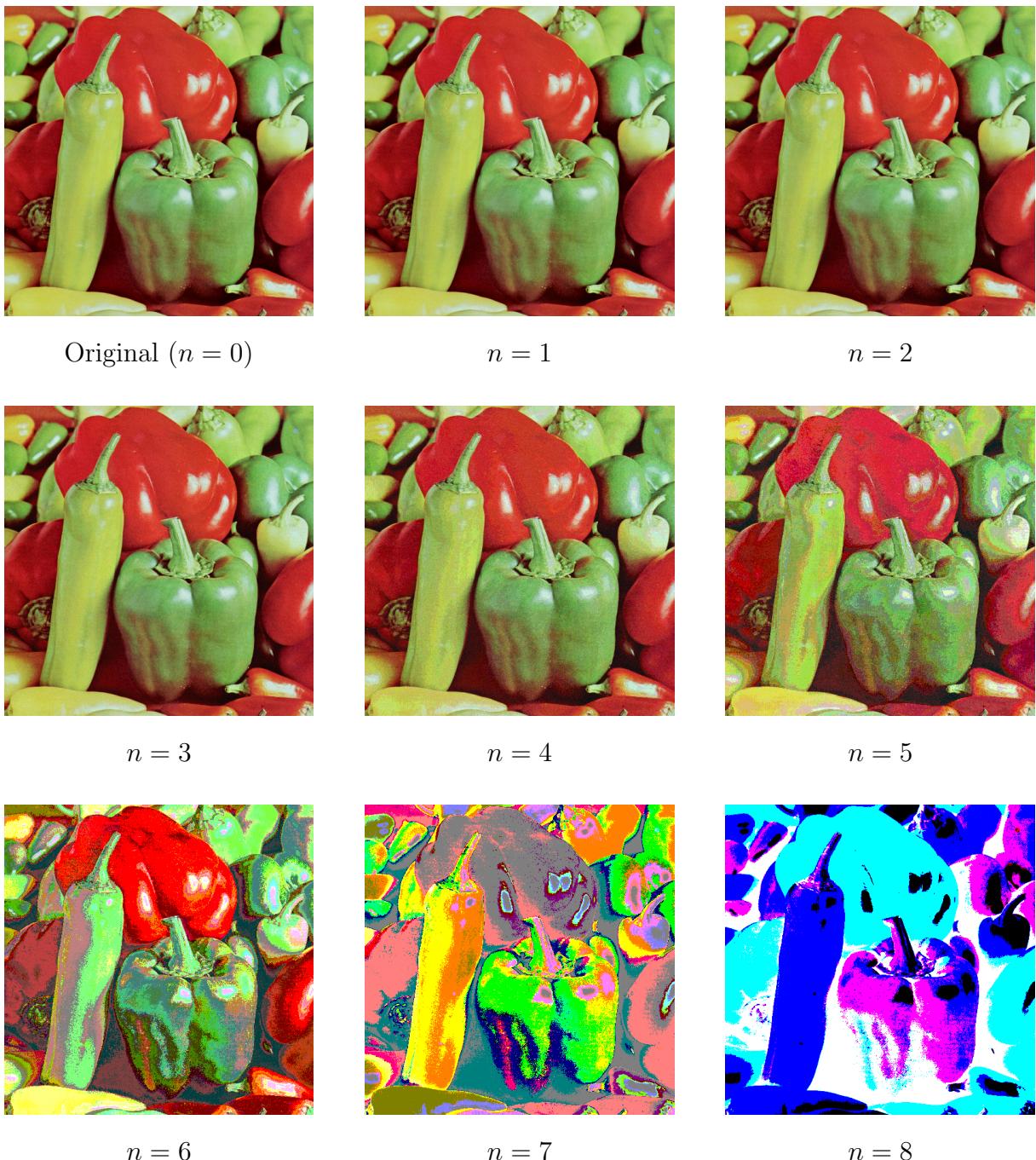


Abbildung 4.1: Farbbild Paprika verändert durch maximalen Fehler für  $n \in [0, 8]$

Zusätzlich wird in diesem Beispiel der schlimmste Fall betrachtet. Das Verstecken einer echten Nachricht wird fast immer ein besseres Ergebnis liefern, da Nachrichtenbit zufällig

mit den des Bilds übereinstimmen oder vorherige Fehler durch weitere Teile der Nachricht wieder ausgeglichen werden.

### 4.3 Beschreibung eines Algorithmus

Im vorherigen Abschnitt wurden die Möglichkeiten von **LSB**-Verfahren untersucht, es kann jetzt eine mehr formale Beschreibung gegeben werden, wie ein solcher Algorithmus umgesetzt werden kann. Da ein steganografisches Verfahren die Vertraulichkeit einer Nachricht nur indirekt sichert, ist es sinnvoll, diese vor dem Verwenden mit einem kryptografischen Algorithmus zu verschlüsseln.

**Definition 4.3.1** (**LSB**-Verfahren). Es sei  $p_{x,y} = (r, g, b)$  ein Pixel und  $\mathbf{B} = p^{m \times n}$  ein Bild mit  $y \in [1, m]$  und  $x \in [1, n]$ . Es sei  $P$  die Menge aller Pixel von  $\mathbf{B}$  und  $v$  eine Funktion mit  $v : P \setminus \{p_{1,1}, p_{m,n}\} \leftarrow v(\mathbf{B})$ . [Algorithmus 4.1](#) und [4.2](#) zeigen ein Verfahren für das Schreiben und Lesen einer Nachricht in  $\mathbf{B}$ .

<b>Algorithmus 4.1 : LSB-Verfahren Schreiben</b>
--------------------------------------------------

<p><b>Input :</b> <math>\mathbf{B}</math> und Nachricht <math>x</math>  <b>Output :</b> <math>\mathbf{B}</math> mit versteckter Nachricht <math>y</math></p> <pre> <b>begin</b>     <math>y \leftarrow e_k(x)</math>     <math>n \leftarrow 1</math>     schreibe Nachrichtenlänge von <math>y</math> nach <math>p_{1,1}</math> und <math>p_{m,n}</math>     <b>while</b> true <b>do</b>         <b>if</b> <math>n = 9</math> <b>then</b> <math>y</math> ist zu lang         <b>for</b> <math>p \in v(\mathbf{B})</math> <b>do</b>             <b>for</b> <math>c \in p</math> <b>do</b>                                // Farbwerte (r,g,b)                 <math>b \leftarrow</math> lese nächstes Bit von <math>y</math>                 schreibe <math>b</math> nach Position <math>n</math> von <math>c</math>                 <b>if</b> <math>y</math> bearbeitet <b>then return</b>             <b>end</b>         <b>end</b>         <math>n \leftarrow n + 1</math>     <b>end</b> <b>end</b> </pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Algorithmus 4.2 : LSB-Verfahren Lesen**

```

Input : B mit versteckter Nachricht  $y$ 
Output : Nachricht  $x$ 
begin
     $n \leftarrow 1$ 
     $y \leftarrow \emptyset$ 
     $l \leftarrow$  lese Nachrichtenlänge bei  $p_{1,1}$  und  $p_{m,n}$ 
    while  $l \neq 0$  do
        for  $p \in v(\mathbf{B})$  do
            for  $c \in p$  do                                // Farbwerte (r,g,b)
                 $b \leftarrow$  lese Bit bei Position  $n$  von  $c$ 
                 $y \leftarrow y \cup \{b\}$ 
                if  $l = 0$  then return  $d_k(y)$ 
                else  $l \leftarrow l - 1$ 
            end
        end
         $n \leftarrow n + 1$ 
    end
end

```

Damit eine Nachricht im Bild möglichst wenig auffällt, macht es Sinn, Pixel so auszuwählen, dass diese gleichmäßig verteilt sind. Die Verteilfunktion  $v$  hat genau diese Aufgabe. Das Bild **B** wird als Folge der natürlichen Zahlen  $a_i = 0, 1, \dots, mn - 1$  betrachtet. Durch ein PRNG wird jetzt eine pseudozufällige Permutation von  $a_i$  bestimmt. Der Algorithmus ist bekannt als das Fisher-Yates-Verfahren:

**Algorithmus 4.3 : Fisher-Yates-Verfahren**

```

Input : Ein Array  $A$  mit allen Elementen aus  $a_i$  und ein seed  $s$ 
Output : Permutation  $A'$ 
begin
    for  $i \leftarrow mn - 1$  to 0 do
         $j \leftarrow$  pseudozufällige Zahl im Bereich  $[0, i]$ 
        tausche  $A[i]$  und  $A[j]$ 
    end
    return  $A$ 
end

```

Der Determinismus des PRNG ist wichtig, da die gleiche Permutation für das Lesen der

Nachricht ein zweites Mal erzeugt werden muss. Durch das Lösen von  $a = n \cdot y + x$  für alle  $a \in A'$  mit  $0 \leq x < n$  kann die Permutation in null indizierte Koordinaten  $(x, y)$  des Bilds umgewandelt werden.

Beispiel 4.3.1. Ein wird ein  $2 \times 3$  großes Bild betrachtet und es gilt  $A' = (2, 5, 3, 1, 0, 4)$ .

$$2 = 3 \cdot 0 + 2 \Rightarrow (2, 0)$$

$$5 = 3 \cdot 1 + 2 \Rightarrow (2, 1)$$

$$3 = 3 \cdot 1 + 0 \Rightarrow (0, 1)$$

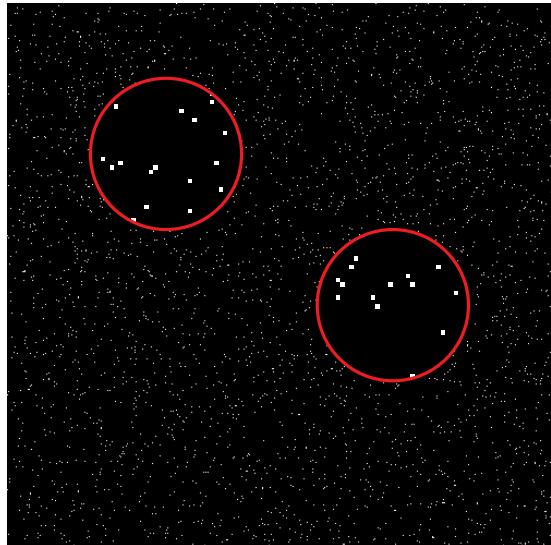
$$1 = 3 \cdot 0 + 1 \Rightarrow (1, 0)$$

$$0 = 3 \cdot 0 + 0 \Rightarrow (0, 0)$$

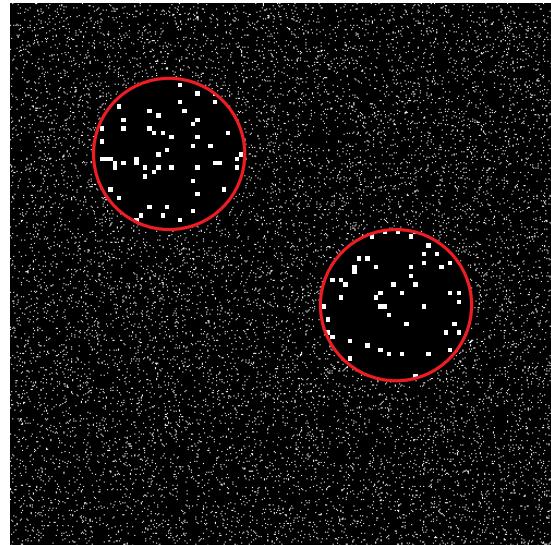
$$4 = 3 \cdot 1 + 1 \Rightarrow (1, 1)$$

△

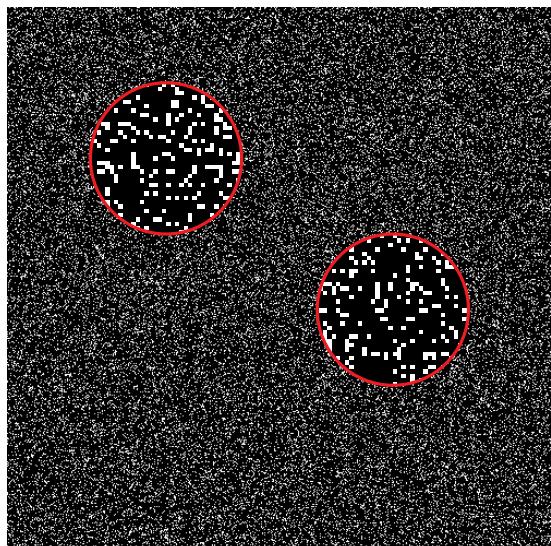
**Abbildung 4.2** zeigt die Koordinatenverteilung für Nachrichten unterschiedlicher Längen auf einem  $500 \times 500$  Pixel Farbbild mit schwarzen Hintergrund. Die durch den Algorithmus errechneten Koordinaten sind weiß eingefärbt.



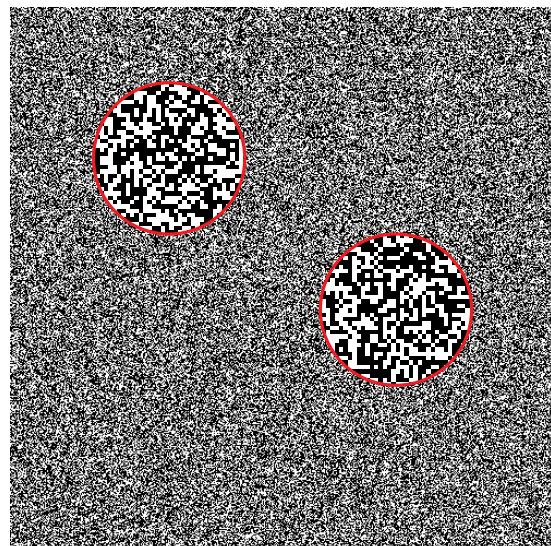
1200 Byte



3600 Byte



10800 Byte



32400 Byte

Abbildung 4.2: Koordinatenverteilung auf einem  $500 \times 500$  Pixel Farbbild mit schwarzen Hintergrund für Nachrichtenlängen von 1200, 3600, 10800 und 32400 Byte.

## 4.4 Die Architektur der Anwendung

Es soll jetzt auf die zu Beginn des Kapitels beschrieben Anwendung eingegangen werden, welche im Rahmen dieser Studienarbeit entwickelt wurde. Die Anwendung muss verschiedene Benutzer verwalten können und einen sicheren Nachrichtenaustausch über in Bildern versteckten Informationen gewährleisten. Die wichtigsten Sicherheitsaspekte eines solchen Systems sind die Folgenden:

1. **Vertraulichkeit** einer Nachricht (Geheimhaltung, Verschlüsselung).
2. **Integrität** einer Nachricht (Hashfunktion, *Message Authentication Codes (MACs)*).
3. **Authentizität** des Empfängers. Es darf nicht passieren, der falschen Person unwissentlich eine geheime Nachricht zu senden. Im Rahmen dieser Betrachtung sollte es reichen, nicht zwei Personen mit demselben Benutzernamen zuzulassen.
4. **Authentifizierung und Autorisierung**. Um eine Nachricht (z. B. für das Schreiben im Bild) benutzergebunden zu verschlüsseln, müssen Anfragen immer klar mit einem Benutzer in Verbindung gebracht werden.

Die verschiedenen Teile der Anwendung können unterteilt werden in die drei Bereiche: Präsentation, Schnittstelle und Persistenz. Benutzer müssen angemeldet sein, um auf geschützte Bereiche der Schnittstelle zuzugreifen. Es wurde eine Token basierte Autorisierung gewählt, welche mithilfe von **JSON Web Tokens (JWTs)** implementiert ist. **JWT** ist ein Internet Standard (RFC 7519, Jones, Bradley und Sakimura 2015) und ein weit verbreitetes Verfahren für die Autorisierung im Web und *Single Sign-On* Anwendungen. **Abbildung 4.3** zeigt einen typischen Anfrageablauf zwischen Anwender und einer Anwendung mit **JWT** Autorisierung.

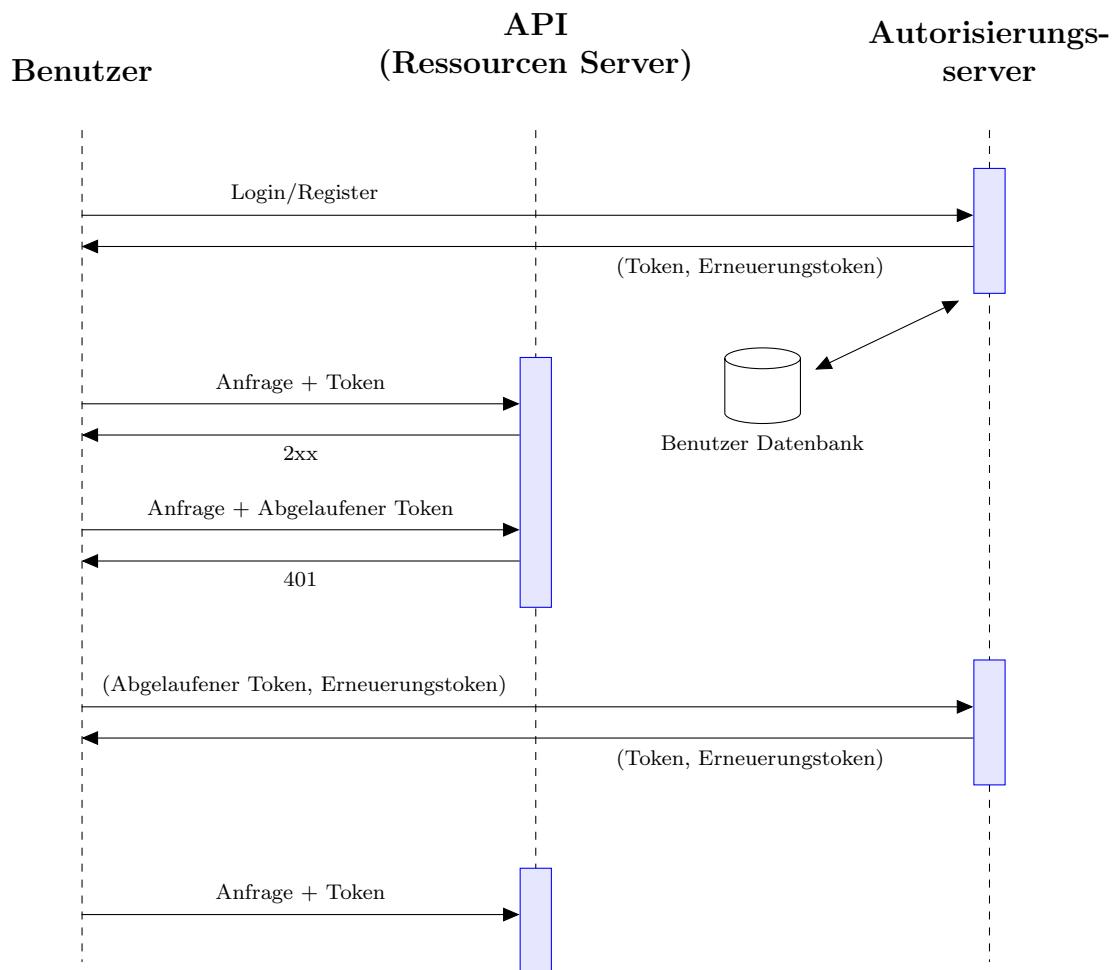


Abbildung 4.3: Sequenzdiagramm **JWT** Autorisierung

Ein Benutzer erhält nach dem Anmelden eine Kombination aus Zugriffstoken (engl. *access token*) und Erneuerungstoken (engl. *refresh token*), welche clientseitig gespeichert werden. Wird versucht, auf einen geschützten Bereich der API zuzugreifen, muss die Anfrage einen gültigen Token enthalten, welcher als Teil des HTTP-Header im Autorisierungsfeld versandt wird. Ist im Header der Anfrage kein oder ein abgelaufener Token vorhanden, wird diese mit einem HTTP-Statuscode 401 „Unautorisiert“ abgelehnt. Ist ein Token abgelaufen, kann dieser beim Autorisierungsserver zusammen mit dem Erneuerungstoken aktualisiert werden.

## 4.5 Sicherheitsanalyse

Es soll nun weiter auf die beschriebene Architektur der Anwendung eingegangen werden, mit dem Ziel, Sicherheitsbedenken aufzudecken und mögliche Lösungen zu diskutieren. Anschließend wird die steganografische Qualität des Algorithmus untersucht und einige Beispiele gezeigt.

### 4.5.1 Risiken der Token basierten Autorisierung

Besondere Vorsicht muss geboten werden, wenn es um die Handhabung der Zugriffstoken geht. Fällt ein Token in die Hände eines Angreifers, kann dieser im Namen des Tokeninhabers beliebig Anfragen durchführen, d. h. sowohl geheime Nachrichten lesen als auch Nachrichten schreiben. Zugriffstoken werden clientseitig gespeichert. Über eine Clientumgebung wie dem Webbrower, besteht in den meisten Fällen keine 100-prozentige Kontrolle, weshalb die volle Sicherheit hier nicht garantiert sein kann. Dennoch sollten Sicherheitsmaßnahmen getroffen werden, um das Risiko eines Missbrauchs zu verringern. Um den Zeitraum von Angriffsmöglichkeiten kurz zu halten, sollte ein Token nur so lange gültig sein wie nötig (z. B. nicht länger als fünf Minuten). Es existieren zwei wesentliche Angriffstypen, welche das Prinzip der Token Autorisierung versuchen auszunutzen. Die folgende Überlegung ist beschränkt auf die Domäne der Webanwendungen:

**Cross-Site-Scripting (XSS)** Schafft es ein Angreifer auf einer Webseite an der richtigen Stelle ein Stück JavaScript auszuführen, kann er mit den richtigen Befehlen den Zugriffstoken ganz einfach auslesen. Webseiten sollten daher an Stellen wie Benutzereingaben vorsichtig sein und diese beispielsweise nie direkt als Teil des HTML anzeigen lassen.

**Cross-Site-Request-Forgery (CSRF)** CSRF-Angriffe zielen darauf ab, HTTP-Anfragen im Namen anderer Benutzer durchzuführen. Sie machen dabei Gebrauch von aktiven Sitzungen oder im Falle der Token Autorisierung von Zugriffstoken, welche pro Anfrage automatisch (z. B. per Cookie) versandt werden. Zugriffstoken sollten somit nie direkt als Cookie gespeichert werden. Besser ist es, Sitzungen indirekt über Erneuerungstoken aufrechtzuerhalten oder in extremen Fällen gar keine Sitzungsinformationen zu speichern.

Die Zugriffstoken der hier beschriebenen Anwendung haben eine Gültigkeitsdauer von 30 Sekunden und es werden keine Sitzungsinformationen gespeichert.

#### 4.5.2 Kryptografische Qualität

Nachrichten werden verschlüsselt, bevor sie in einem Bild versteckt werden. Verschlüsselung geschieht durch die *ASP.NET Core Data Protection API*, welche als sicher angenommen wird.<sup>1</sup> Als Garantie, dass nur der geplante Empfänger eine Nachricht lesen kann, ist ein eindeutiger *purposes parameter* nötig, welcher der Verschlüsselungsfunktion übergeben wird. Der Parameter setzt sich zusammen aus (Datenbank ID || - || Name) und ist somit garantiert eindeutig. Das Symbol || beschreibt hierbei die Konkatenation zweier Zeichenketten. [Algorithmus 4.4](#) und [4.5](#) zeigen das Prinzip der Verschlüsselung. Die *Data Protection API* stellt eine Methode `CreateProtector` zur Verfügung:

<b>Algorithmus 4.4 : Data Protection API Verschlüsselung</b>
--------------------------------------------------------------

<b>Input :</b> Nachricht $x$ and Informationen des Empfängers ( $id, username$ ) <b>Output :</b> Verschlüsselte Nachricht $y$ <b>begin</b> $(e_k, d_k) \leftarrow \text{CreateProtector}(id \parallel - \parallel username)$ <b>return</b> $e_k(x)$ <b>end</b>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>Algorithmus 4.5 : Data Protection API Entschlüsselung</b>
--------------------------------------------------------------

<b>Input :</b> Verschlüsselte Nachricht $y$ <b>Output :</b> Entschlüsselte Nachricht $x$ <b>begin</b> $(id, username) \leftarrow$ Benutzerinformationen der Anfrage $(e_k, d_k) \leftarrow \text{CreateProtector}(id \parallel - \parallel username)$ <b>return</b> $d_k(y)$ <b>end</b>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Das Funktionspaar  $(e_k, d_k)$  der `CreateProtector` Methode verwendet **MACs**, um die unbemerkte Nachrichtenmanipulation zu verhindern. Wäre die Nachricht während der

---

<sup>1</sup><https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/introduction?view=aspnetcore-5.0>

Übertragung verändert worden, würde es während der Entschlüsselung auffallen.

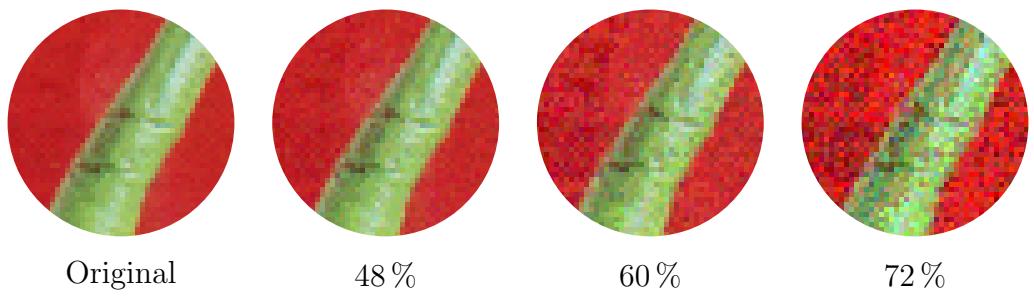
### 4.5.3 Steganografische Qualität

Um die Qualität des steganografischen Algorithmus zu beurteilen, lohnt es sich vor allem einige Beispiele anzusehen. Nach der Überlegung in [Abschnitt 4.2](#) können wir abschätzen, wie lang eine Nachricht sein sollte, um mit den Beispielbildern sinnvolle Beobachtungen machen zu können. Die nachfolgende Tabelle zeigt die nötige Länge einer Nachricht, um auf allen  $n$  [LSB](#) der Farbkanäle Änderungen auszulösen. Die Nachrichtenlänge  $l$  ist als prozentualer Anteil der maximalen Bildkapazität  $\mathbf{B}_{max}$  angegeben:

Tabelle 4.1: Die Zuordnung von  $n$  zu der Nachrichtenlänge  $l$  in Bezug auf  $\mathbf{B}_{max}$

n	$l$
0	0
1	12.5
2	25
3	37.5
4	50
5	62.5
6	75
7	87.5
8	100

Es macht keinen Sinn, Nachrichten zu betrachten mit  $l < 37.5\%$ , da diese in den allermeisten Fällen eine gute Bildqualität aufweisen. Zusätzlich macht es keinen Sinn, sehr lange Nachrichten zu untersuchen, da diese immer eine schlechte Bildqualität erzeugen. In den folgenden Beispielen wird deshalb nur der mittlere Bereich  $n \in [4, 6]$  beachtet. Es werden jeweils drei Bilder miteinander verglichen mit einer versteckten Nachrichtenlänge von 48, 60 und 72 % der maximalen Bildkapazität. Die Nachrichten sind jeweils zufällig erzeugt:

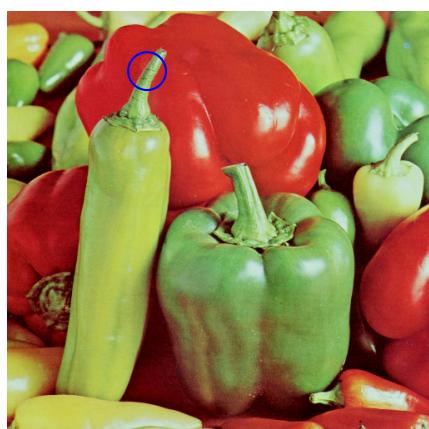


Original

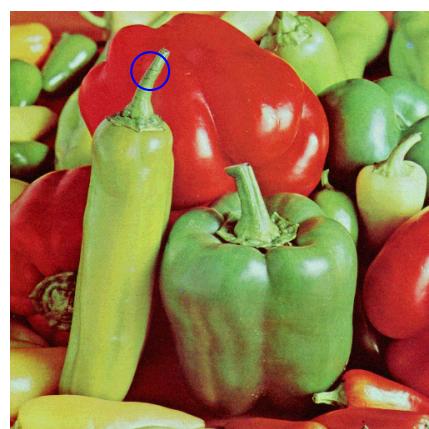
48 %

60 %

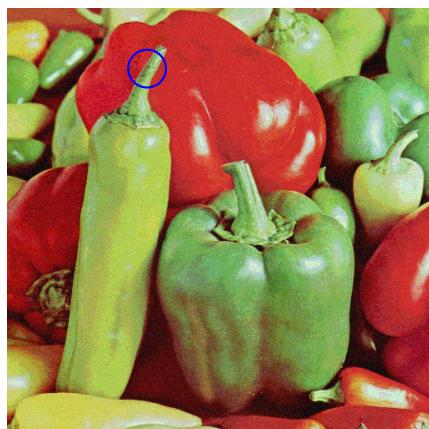
72 %



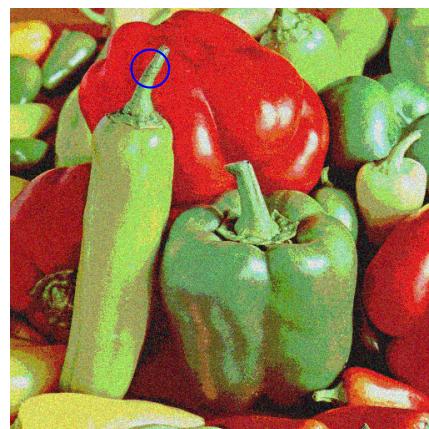
Original



48 % (373 kB)

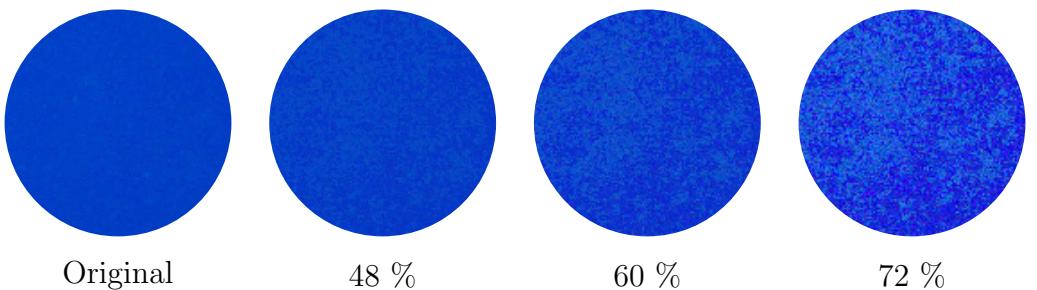


60 % (466 kB)



72 % (560 kB)

Abbildung 4.4: Paprika mit einer Größe von  $509 \times 509$  Pixel. Gute Bildqualität zwischen 373 und 466 kB versteckter Nachricht ( $B_{max} \approx 777$  kB).



Original



48 % (3.6 MB)

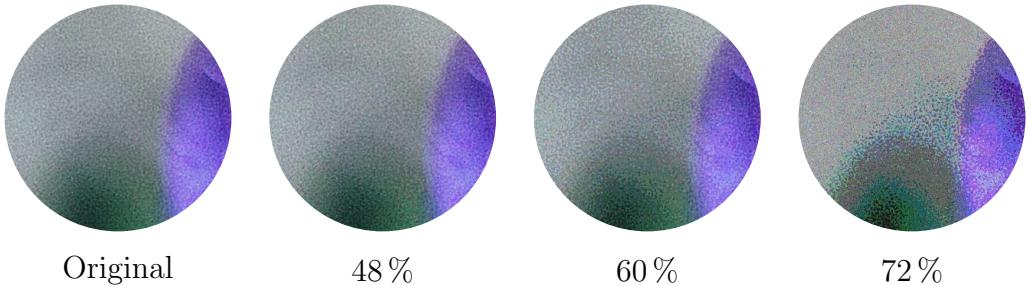


60 % (4.5 MB)



72 % (5.4 MB)

Abbildung 4.5: Schildkröte mit einer von Größe  $1368 \times 1824$  Pixel. Gute Bildqualität bis zu 3.6 MB versteckter Nachricht ( $\mathbf{B}_{max} \approx 7.5$  MB).



Original



48 % (17.5 MB)



60 % (21.9 MB)



72 % (26.3 MB)

Abbildung 4.6: Blumen mit einer Größe von  $2848 \times 4272$  Pixel. Gute Bildqualität bis zu 22 MB versteckter Nachricht ( $\mathbf{B}_{max} \approx 36.5$  MB).

Es kann beobachtet werden, dass eine gute Bildqualität erreicht wird, wenn die Grenze von vier Bitänderungen pro Farbkanal ( $n = 4$ ) nicht überschritten wird. Nachrichten mit einer Länge größer 50 % der maximalen Bildkapazität werden bis in die höheren Bitstellen geschrieben und sind deshalb zunehmend schnell zu erkennen. Nicht alle Bilder sind gleich gut geeignet um lange Nachrichten zu verstecken. Änderungen bei 4.5 (Schildkröte) mit viel einfarbigen Hintergrund lösen schneller einen sichtbaren Effekt aus als bei Bild 4.4 (Paprika) oder 4.6 (Blumen), wo auch Nachrichten länger 50 % eher schwer zu erkennen sind. Bilder mit versteckten Nachrichten sind empfindlich gegenüber Änderungen und sollten daher nicht als Grafikformat mit verlustbehafteter Datenkompression gespeichert werden (z. B. JPEG). Der Autor möchte dem Leser empfehlen, die Anwendung selbst einmal auszuprobieren. Es wird hierzu ein Computer mit Docker benötigt und eine kurze Anleitung sowie der Quelltext zum Projekt sind über den folgenden Link auf GitHub zu finden.<sup>2</sup> Die folgenden zwei Bilder zeigen einen Ausschnitt aus der Benutzeroberfläche:

---

<sup>2</sup><https://github.com/jens-studienarbeit/image-data-hiding>

Send To

Alice



Message

Geheime Nachricht an Alice

Remove

Image

Upload a file or drag and drop

Leave this empty and receive a random image



peppers.png

Encode

Abbildung 4.7: Benutzeroberfläche Nachrichten schreiben

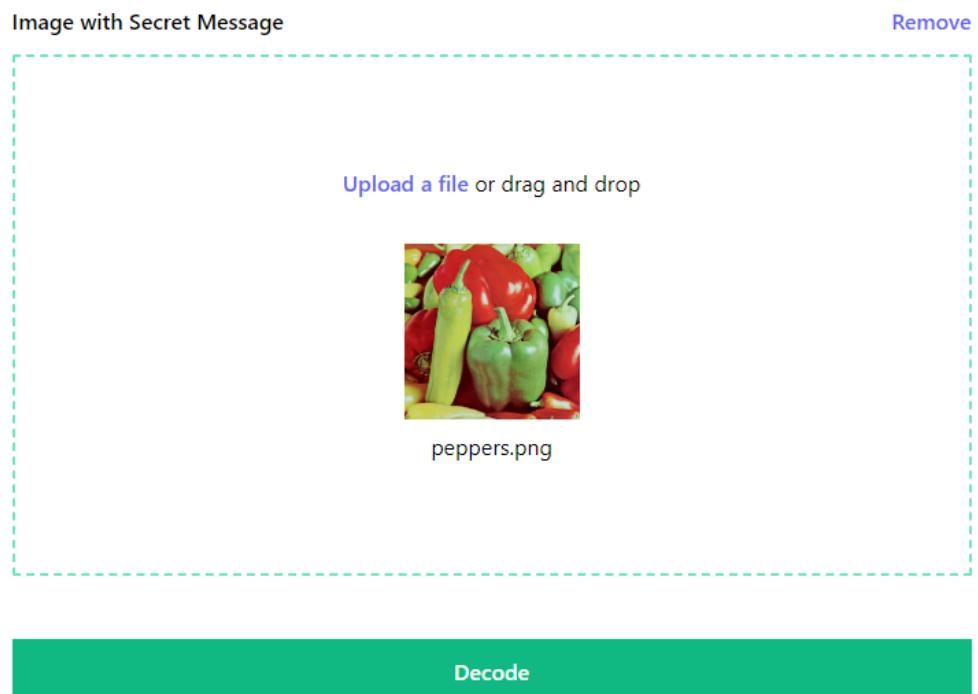


Abbildung 4.8: Benutzeroberfläche Nachrichten lesen

# Kapitel 5

## Reflektion und Ausblick

Die vorliegende Studienarbeit befasst sich mit der Frage „Wie kann ein Algorithmus entwickelt werden, um eine Nachricht in einer Bilddatei zu verstecken, ohne dabei die visuelle Qualität sichtbar zu beeinflussen?“. Zusätzlich wird den Fragen nachgegangen „Wie kann die Vertraulichkeit dieser Nachricht gesichert werden?“ und „Wie ist der Algorithmus als Teil einer größeren Anwendung einzuordnen?“. Um diese Fragen zu beantworten, wurden die Prinzipien der symmetrischen und asymmetrischen Kryptografie kritisch aufgearbeitet und die Möglichkeiten eines steganografischen Verfahrens untersucht.

Als Ergebnis einer ersten Überlegung (4.2) hat sich gezeigt, dass Änderungen an einem Bild bis zur viert niedrigwertigsten Bitstelle im Farbkanal nur schwer zu erkennen sind. Es wurde hieraus geschlossen, dass ein Algorithmus, welcher nach dem **LSB**-Verfahren arbeitet, grundsätzlich ein steganografisch gutes Ergebnis liefert. Insbesondere kann nach (4.2) und [Tabelle 4.1](#) mit Zuversicht behauptet werden, dass diese Qualität für ein Großteil der Bilder mit  $l < 37.5$  erhalten bleibt.

Es konnte anhand dieser Überlegungen schließlich ein Algorithmus vorgestellt werden mit guter steganografischer Qualität. In Bezug auf eine Anwendung wurde gezeigt, wie Nachrichten benutzergebunden verschlüsselt werden. Zusammen mit dem Algorithmus konnte somit dann ein System beschrieben werden, welches einen sicheren Nachrichtenaustausch über in Bildern versteckten Informationen ermöglicht.

# Literaturverzeichnis

- Barry, Mark (Juni 2004). *Cryptography in Home Entertainment*. Einsichtsname: 03.01.2021.  
URL: <http://www.math.ucsd.edu/~crypto/Projects/MarkBarry/index.htm>.
- Cimato, Stelvio und Ching-Nung Yang (2017). *Visual Cryptography and Secret Image Sharing*. CRC Press. ISBN: 978-1-4398-3722-1.
- Haahr, Mads (2021). *Introduction to Randomness and Random Numbers*. Einsichtsname: 14.01.2021. URL: <https://www.random.org/randomness>.
- Jones, Michael B., John Bradley und Nat Sakimura (Mai 2015). *JSON Web Token (JWT)*.  
Einsichtsname: 27.03.2021. URL: <https://tools.ietf.org/html/rfc7519>.
- McCullagh, Adrian und William Cealli (2000). *Non-Repudiation in the Digital Environment*. Einsichtsname: 19.01.2021. URL: <https://firstmonday.org/ojs/index.php/fm/article/view/778/687>.
- Paar, Christof und Jan Pelzl (2010). *Understanding Cryptography*. Springer. ISBN: 978-3-642-04100-6.
- Remmert, Reinhold und Peter Ullrich (2008). *Elementare Zahlentheorie*. 3. Aufl. Birkhäuser.  
ISBN: 978-3-7643-7730-4.
- Spannagle, Christian (Jan. 2011). *Der Euklidische Algorithmus*. Einsichtsname: 20.01.2021.  
URL: [https://wiki.zum.de/wiki/PH\\_Heidelberg/Bausteine/Der\\_Euklidische\\_Algorithmus](https://wiki.zum.de/wiki/PH_Heidelberg/Bausteine/Der_Euklidische_Algorithmus).
- (Juni 2012a). *Diophantische Gleichungen*. Einsichtsname: 21.01.2021. URL: [https://wiki.zum.de/wiki/PH\\_Heidelberg/Zahlentheorie/Diophantische\\_Gleichungen](https://wiki.zum.de/wiki/PH_Heidelberg/Zahlentheorie/Diophantische_Gleichungen).
- (Juni 2012b). *Sätze von Euler und Fermat*. Einsichtsname: 23.01.2021. URL: [https://wiki.zum.de/wiki/PH\\_Heidelberg/Bausteine/S%C3%A4tze\\_von\\_Euler\\_und\\_Fermat](https://wiki.zum.de/wiki/PH_Heidelberg/Bausteine/S%C3%A4tze_von_Euler_und_Fermat).

# Glossar

**AES** Advanced Encryption Standard. [11](#), [20](#), [21](#)

**CSPRNG** *cryptographically secure pseudorandom number generator.* [16](#), [18](#)

**DES** Data Encryption Standard. [11](#), [20](#)

**JWT** JSON Web Token. [VI](#), [44](#), [45](#)

**LSB** *least significant bit.* [VIII](#), [38](#), [40](#), [41](#), [48](#), [55](#)

**MAC** *Message Authentication Code.* [44](#), [47](#)

**OTP** One-Time-Pad. [17](#), [18](#), [20](#)

**PRNG** *pseudorandom number generator.* [16](#), [18](#), [41](#)

**RNG** *random number generator.* [15](#)

**TRNG** *true random number generator.* [15](#), [16](#), [17](#)