

Grundlagen der Kryptographie und Steganographie

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

Inhaltsverzeichnis

1 Einleitung	1
1.1 Symmetrische Verschlüsselung	3
1.2 Modulare Arithmetik	4
1.3 Die Cäsar-Chiffre	8
2 Stromchiffren und Blockchiffren	10
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
3 Das RSA-Kryptosystem	20
3.1 Asymmetrische Verschlüsselung	20
3.2 Zahlentheoretische Grundlagen	23
3.2.1 Der Euklidische Algorithmus	24
3.2.2 Der erweiterte Euklidische Algorithmus	26
3.2.3 Die Eulersche Phi-Funktion	30
3.2.4 Satz von Euler und Fermat	32
3.3 Das RSA-Verfahren	34
4 Ein Steganografischer Algorihtmus	37
4.1 Modifikation von Bilddateien	37
4.2 Beschreibung eines Algorithmus	40
4.3 Die Architektur der Anwendung	44

4.4	Sicherheitsanalyse	46
4.4.1	Risiken der Token basierten Autorisierung	46
4.4.2	Kryptographische Qualität	47
4.4.3	Steganografische Qualität	48

Literaturverzeichnis A

Glossar 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	29

Abbildungsverzeichnis

1.1	Die Kryptologie und ihre Untergebiete (Paar und Pelzl 2010, S. 3)	2
1.2	Kommunikation über einen unsicheren Kanal	3
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)	18
3.1	Das Prinzip der symmetrischen Verschlüsselung	20
3.2	Das Prinzip der asymmetrischen Verschlüsselung	22
3.3	Schlüsselaustausch mit asymmetrischer Verschlüsselung	23
4.1	Farbbild Paprika verändert durch maximalen Fehler für $n \in [0, 8]$	39
4.2	Koordinatenverteilung auf einem 500×500 Pixel Schwarzbild für Nachrichtenlängen von 1200, 3600, 10800 und 32400 Byte.	43
4.3	Sequenzdiagramm JWT Autorisierung	45
4.4	Blumen. Größe 4272×2848 Pixel. Maximale Kapazität $\approx 36,5$ MB. Gute Bildqualität bis zu 20MB versteckter Nachricht.	48
4.5	Paprika. Größe 509×509 Pixel. Maximale Kapazität ≈ 777 KB. Gute Bildqualität bis zu 300-500 KB versteckter Nachricht.	49
4.6	Schwarz. Größe 512×512 Pixel. 10 KB Nachricht, veränderte Pixel sind weiß eingefärbt.	50

Liste der Algorithmen

3.1	Euklidischer Algorithmus	26
3.2	Erweiterter Euklidischer Algorithmus	30
4.1	LSB-Verfahren Schreiben	40
4.2	LSB-Verfahren Lesen	41
4.3	<i>Data Protection Encryption</i>	47
4.4	<i>Data Protection Decryption</i>	47

Kapitel 1

Einleitung

Redet man heutzutage über das Thema Kryptographie, sind im Gespräch wahrscheinlich Themen wie E-Mail-Verschlüsselung, Internetprotokolle oder Anwendungen im Bankenwesen. Auch bekannt sind die Angriffe auf kryptographische Systeme, wie zum Beispiel die Entzifferung der durch die Enigma-Chiffriermaschine verschlüsselten deutschen Funksprüche während des Zweiten Weltkrieges. Es scheint als wäre Kryptographie stark mit den modernen elektronischen Kommunikationstechniken verbunden. Dies ist allerdings nicht so: Frühe Formen der Kryptographie 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 unterschiedliche kryptographische Verfahren unterschieden, diese sind Symmetrische- und Asymmetrische Algorithmen. Die symmetrische Verschlüsselung ist seit langer Zeit ein fester Bestandteil der Kryptographie, 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 Kryptographie, 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 Kryptographie ist in Abbildung 1.1 zu sehen. Es ist zu bemerken, dass an oberster Stelle nicht die Kryptographie, sondern der Oberbegriff Kryptologie zu finden ist, welche sich in die zwei großen Bereiche unterteilt:

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

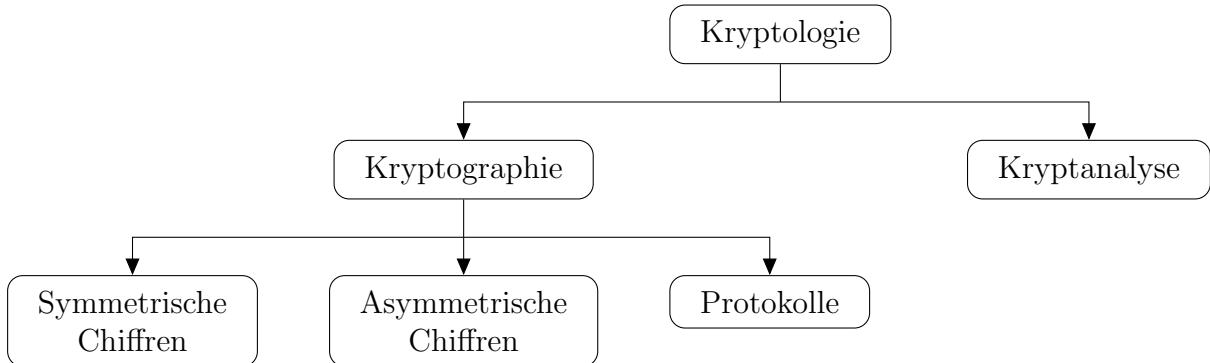


Abbildung 1.1: Die Kryptologie und ihre Untergebiete (Paar und Pelzl 2010, S. 3)

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

Auf den ersten Blick scheint das *Kerckhoffs's principle* nicht sonderlich intuitiv. 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 geringen Aufwand ermöglichten (Barry 2004).

1.1 Symmetrische Verschlüsselung

Denkt man an die Teilbereiche der Kryptographie, 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.

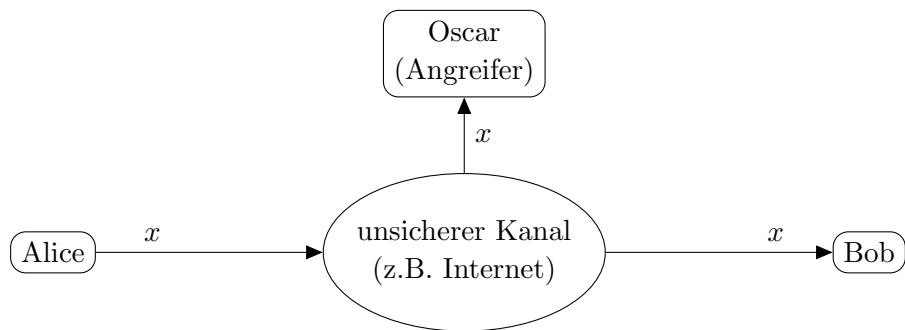


Abbildung 1.2: Kommunikation über einen unsicheren Kanal

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 Bits.

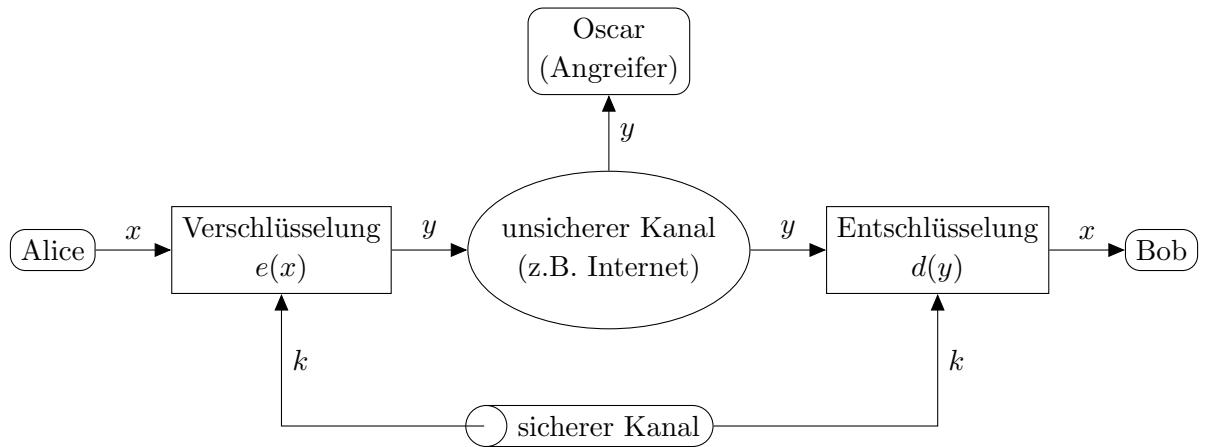


Abbildung 1.3: Kommunikation mit symmetrischer Verschlüsselung

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

- x ist der Klartext (engl. *plaintext*).
- y ist das Chiffrat oder der Geheimtext (engl. *ciphertext*).
- k ist der Schlüssel (engl. *key*).
- $e(\cdot)$ ist die Verschlüsselung (engl. *encryption*).
- $d(\cdot)$ ist die Entschlüsselung (engl. *decryption*), d.h. die Umkehrfunktion von e .

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 kryptographischen 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) in der wir es gewöhnt sind in unendlichen Mengen zu rechnen, z.B. die natürlichen Zahlen oder die reellen Zahlen. Die modulare Arithmetik, d.h. die Division mit Rest, bietet eine gute Möglichkeit um in diesen begrenzten Mengen rechnen zu können.

Lemma 1.2.1 (Remmert und Ullrich 2008, S. 179–180). *Folgende Aussagen über drei ganze Zahlen a, b, m , wobei $m > 0$, sind äquivalent:*

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

Beweis. $\exists q_1, q_2, r_1, r_2 \in \mathbb{Z}$

Es seien $a = q_1m + r_1$ und $b = q_2m + r_2$ mit $0 \leq r_1, r_2 < m$, die Gleichungen, die bei Division mit Rest entstehen.

i) \Rightarrow ii):

Es gilt: $r_1 = r_2$.

Zu zeigen: $m|(a - b)$.

$$\begin{aligned} a - b &= q_1m + r_1 - q_2m - r_2 \\ a - b &= (q_1 - q_2)m + r_1 - r_2 \\ \Leftrightarrow a - b &= (q_1 - q_2)m \end{aligned}$$

Aus der letzten Gleichung folgt: $m|(a - b)$.

ii) \Rightarrow i):

Es gilt: $m|(a - b)$.

Zu zeigen: $r_1 = r_2$.

$$\begin{aligned} m|(a - b) \\ \Leftrightarrow m|(q_1m + r_1) - (q_2m + r_2) \\ \Leftrightarrow m|(q_1 - q_2)m + (r_1 - r_2) \end{aligned}$$

Aus $m|(q_1 - q_2)m + (r_1 - r_2)$ und $m|(q_1 - q_2)m$ folgt $m|(r_1 - r_2)$. ¹

Entweder $r_1 - r_2$ ist ein Vielfaches von m oder 0. Da sich r_1 und r_2 im Bereich $[0, m)$ befinden ist die einzige Lösung $r_1 - r_2 = 0$. Es folgt: $r_1 = r_2$. \square

¹Aus $a|b$ und $a|c$ folgt $a|(xb + yc)$, $\forall x, y \in \mathbb{Z}$ (Remmert und Ullrich 2008, S. 23). Also mit konkreten Werten $m|(q_1 - q_2)m + (r_1 - r_2) - (q_1 - q_2)m \Leftrightarrow m|(r_1 - r_2)$.

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 $m|(a - b)$ 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 1.2.1. Es sind $a = 29$ und $m = 8$. Man schreibt

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

und deshalb $8|(29 - 5)$. \triangle

Die Gleichung der Modulo Rechnung hat unendlich viele Lösungen. Zu einem gegebenen m gibt es beliebig viele Zahlen a , welche den selben Rest ergeben.

Beispiel 1.2.2. Die folgenden Zahlen erfüllen die Gleichung $a \pmod{8} = 5$. Sie werden kongruent modulo 8 genannt:

$$29 \equiv 21 \equiv 13 \equiv 5 \equiv -3 \equiv -11 \pmod{8}$$

Elemente welche bei der Modulo Rechnung den gleichen Rest ergeben, können in eine Klasse zusammengefasst werden, diese nennt man dann eine Restklasse bezüglich m . Die acht Restklassen bezüglich 8 sind:

$$\{\dots, -24, -16, -8, 0, 8, 16, 24, \dots\}$$

$$\{\dots, -25, -17, -9, 1, 9, 17, 25, \dots\}$$

⋮

$$\{\dots, -17, -9, -1, 7, 15, 23, 31, \dots\}$$

\triangle

Alle Elemente einer Restklasse verhalten sich gleich. Zu einem gegebenen Modulo m spielt es keine Rolle, welches Element der Restklasse für eine Berechnung ausgewählt wird. Diese Eigenschaft ist von großem Nutzen, vor allem bei Berechnungen mit großen Zahlen, wie es in der Kryptographie oft der Fall ist.

Beispiel 1.2.3 (Paar und Pelzl 2010, S. 15–16). Die Hauptoperation in vielen Asymmetrischen Chiffren ist die Exponentiation der Form $x^e \pmod{m}$, wobei x, e, m sehr große ganze Zahlen sind. Anhand eines Beispiels können zwei Formen der modularen Exponentiation gezeigt werden. Es soll das Ergebnis der Berechnung $3^8 \pmod{7}$ ermittelt werden. Im ersten Beispiel wird das Ergebnis einfach ausgerechnet, und im zweiten Beispiel wird zwischen den Restklassen gewechselt:

1. $3^8 = 6561 \equiv 2 \pmod{7}$, weil $6561 = 937 \cdot 7 + 2$

Wir erhalten das relative große Zwischenergebnis 6561 obwohl wir wissen, dass das Ergebnis im Bereich $[0, 6]$ liegen muss.

2. Es ist $3^3 = 27 \equiv -1 \pmod{7}$. Man schreibt:

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

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

△

Bemerkung 1.2.1. Paar und Pelzl (ebd., S. 16) haben sich in ihrem Buch darauf geeinigt, in der Kongruenzrelation (1.1) ein b 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 $27 \equiv 13 \pmod{7}$.² Mathematisch macht es jedoch keinen Unterschied.

Wir vereinbaren für den folgenden Text, dass auch die Null eine natürliche Zahl ist, also $\mathbb{N} := \{0, 1, 2, \dots, 1000, 1001, \dots\}$. 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

²Mit dieser Vereinbarung ist das b der Relation $a \equiv b \pmod{m}$ eine Lösung für die Gleichung $a \pmod{m} = r = b$.

nächsten Abschnitt besser beschreiben zu können, wird an dieser Stelle der Ring \mathbb{Z}_m definiert:

Definition 1.2.1 (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 $a + b \pmod{m}$
 - (b) Ein Produkt $a \cdot b \pmod{m}$

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 Buchstabe weitergemacht. Sowohl Klartext- als auch Geheimtextbuchstaben sind somit Teil des Rings \mathbb{Z}_{26} . Die Ver- und Entschlüsselung kann nun folgendermaßen ausgedrückt werden:

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

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

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

Das Verschlüsselungsverfahren funktioniert.

Beweis. Die Entschlüsselung der Verschlüsselung ergibt den Klartext.

Zu zeigen: $d_k(e_k(x)) = x$.

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

□

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

$$\text{ATTACK} = x_1, x_2, \dots, x_6 = 0, 19, 19, 0, 2, 10$$

Der Geheimtext wird wie folgt berechnet:

$$\begin{aligned} e_9(0) &\equiv 9 \pmod{26} \\ e_9(2) &\equiv 11 \pmod{26} \\ e_9(10) &\equiv 19 \pmod{26} \\ e_9(19) &\equiv 28 \equiv 2 \pmod{26} \end{aligned}$$

$$y_1, y_2, \dots, y_6 = 9, 2, 2, 9, 11, 19 = \text{JCCJLT}$$

△

Wie zu erwarten ist die Cäsar-Chiffre natürlich nicht besonders sicher. 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

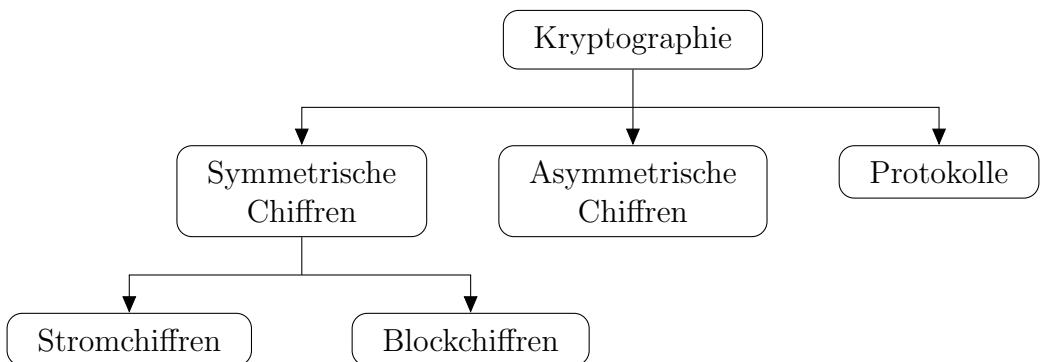


Abbildung 2.1: Unterteilung der Symmetrischen Chiffren (Paar und Pelzl 2010, S. 29)

Werfen wir in Abbildung 2.1 einen genaueren Blick auf die Algorithmen der Kryptographie, stellen wir fest: Das Gebiet der symmetrischen Verschlüsselungsverfahren kann unterteilt werden in Strom- 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.

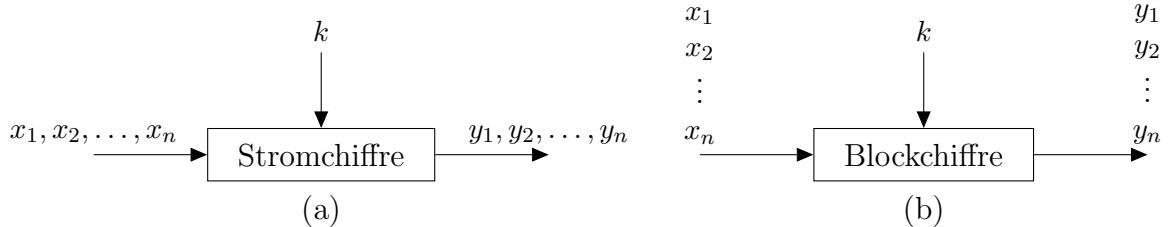


Abbildung 2.2: Das Prinzip der Verschlüsselung von n Bits mittels Strom- (a) und Blockchiffre (b) (Paar und Pelzl 2010, S. 30)

Blockchiffren arbeiten mit Datenblöcken fester Länge, wobei ein Großteil der relevanten Algorithmen eine Eingangsweite von 64 oder 128 Bits 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. **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. **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 (Abschnitt 1.3), 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 uneingeschränkt sicher, erschwert aber dennoch das einfache Abhören.
3. In der Vergangenheit galt der generelle Gedanke, dass Stromchiffren effizienter seien als Blockchiffren. Im Allgemeinen gilt diese Annahme 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

Wie oben erwähnt, verschlüsseln Stromchiffren jedes Bit im Klartext einzeln. Hierfür wird durch einen Schlüssel ein geheimer Bitstrom errechnet, welcher paarweise mit den Bits 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 $x_i, y_i, s_i \in \{0, 1\}$ die einzelnen Bits aus Klartext, Geheimtext und Schlüsselstrom. Es gilt:

Verschlüsselung: $y_i = e_{s_i}(x_i) \equiv x_i + s_i \pmod{2}$

Entschlüsselung: $x_i = d_{s_i}(y_i) \equiv y_i + s_i \pmod{2}$

Betrachtet man die Ver- und Entschlüsselungsfunktion, fallen drei Aspekte auf welche besprochen werden müssen (ebd., S. 31–34):

1. Warum ist Verschlüsselung und Entschlüsselung die selbe Funktion?
2. Warum ist einfache Addition im Ring \mathbb{Z}_2 eine gute Verschlüsselung?
3. Was sind die Eigenschaften der Schlüsselstrombits s_i ?

Warum ist Verschlüsselung und Entschlüsselung die selbe 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.

$$\begin{aligned} y_i &\equiv x_i + s_i \pmod{2} \\ -x_i &\equiv -y_i + s_i \pmod{2} \\ x_i &\equiv y_i + s_i \pmod{2} \end{aligned}$$

□

Im letzten Schritt wird erneut von dem Wechsel innerhalb der Restklasse Gebrauch gemacht, denn es gilt $-1 \equiv 1 \pmod{2}$.

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

Das Rechnen 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 Arithmetik durch einfache boolesche Algebra. Betrachtet man 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	s_i	$y_i \equiv x_i + s_i \pmod{2}$
0	0	0
0	1	1
1	0	1
1	1	0

Das XOR-Gatter spielt eine wesentliche Rolle in vielen kryptographischen Verfahren. Es besitzt besondere Eigenschaften, welche es von anderen Logikgattern unterscheidet und jetzt untersucht werden sollen. Angenommen es soll das Klartextbit $x_i = 0$ verschlüsselt werden. In der Wahrheitstabelle 2.2 des XOR-Gatters befindet man sich demnach in der

Tabelle 2.2: Wahrheitstabelle der Exklusiv-Oder-Verknüpfung

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

ersten oder zweiten Zeile. Je nach Schlüsselbit s_i ist das Geheimtextbit y_i gegeben als $y_i = 0 \oplus 0 \vee 0 \oplus 1 = 0 \vee 1$. Verhalten sich die Schlüsselbit unvorhersehbar, d.h. sie sind mit genau 50 prozentiger Wahrscheinlichkeit entweder null oder eins, ist es nur durch das Chiffraut nicht möglich auf den Klartext zu schließen. Zu jedem Zeitpunkt hat ein Angreifer nur eine 50 prozentige Chance den richtigen Klartext zu erraten. Gleichermaßen kann argumentiert werden, sei $x_i = 1$. 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.

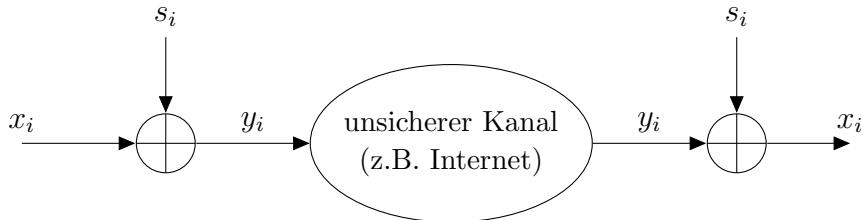


Abbildung 2.3: Der Ver- und Entschlüsselungsprozess bei Stromchiffren

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 $(s_0, s_1, \dots, s_7) = 00111010_2$.

$$\begin{array}{ll}
 x_0, \dots, x_7 = 01010000_2 = 80_{10} = P \\
 \oplus & \text{Alice} \\
 s_0, \dots, s_7 = 00111010_2 & \\
 y_0, \dots, y_7 = 01101010_2 = 106_{10} = j \\
 \\
 j = 01101010_2 & \text{Oscar} \\
 \xrightarrow{\hspace{1cm}} & \\
 \\
 y_0, \dots, y_7 = 01101010_2 = 106_{10} = j & \\
 \oplus & \text{Bob} \\
 s_0, \dots, s_7 = 00111010_2 & \\
 x_0, \dots, x_7 = 01010000_2 = 80_{10} = P &
 \end{array}$$

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

Stromchiffren erscheinen fast zu gut, um wahr zu sein, doch wie später gezeigt wird, gibt es aus auch hier gewisse Nachteile, welche in der Praxis entstehen. Es bleibt die Letzte der drei Fragestellungen zu beantworten.

Was sind die Eigenschaften der Schlüsselstrombits?

Die Sicherheit von Stromchiffren hängt vollständig von der Qualität des Schlüsselstroms ab. Das Generieren dieser Bitfolge (s_1, s_2, \dots, s_n) bildet die zentrale Fragestellung der Verschlüsselungsmethode. Es wurde gezeigt, dass der Schlüsselstrom wie eine zufällige Bitsequenz aussehen muss. Im nächsten Abschnitt soll deshalb das Thema der Zufallszahlen diskutiert werden.

2.3 Zufallszahlengeneratoren

Wie oben beschrieben, ist die Unvorhersehbarkeit des Schlüsselstroms die wesentliche Eigenschaft von Stromchiffren. Im Folgenden sollen deshalb einige Formen von Zufallszahlengeneratoren (engl. *random number generators (RNGs)*) vorgestellt werden (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. Notiert man beispielsweise das Ergebnis von 100 Münzwürfen und erzeugt eine Folge von 100 Bit, ist es quasi unmöglich, die selbe Sequenz ein zweites Mal zu erzeugen. Die Wahrscheinlichkeit das dies passieren würde, beträgt $1/2^{100}$, was verschwindend gering ist. Echte Zufallszahlen basieren auf physikalischen Prozessen. Beispiele umfassen Münzwurf, Würfeln, radioaktiver Zerfall oder atmosphärisches Rauschen. **TRNGs** werden in der Kryptographie und der Schlüsselerzeugung häufig eingesetzt.

Pseudozufallszahlengeneratoren Pseudozufallszahlengeneratoren (engl. *pseudo random 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 eine wirklich zufällige Folge, werden sie durch mathematische Formeln errechnet und sind deterministisch, d.h. die selbe Sequenz kann zu einem späteren Zeitpunkt reproduziert werden. In der Kryptographie können **PRNGs** aufgrund dieser Eigenschaft nicht ohne weiteres eingesetzt werden. Dennoch haben sie außerhalb der Kryptographie weitreichende Anwendungsbereiche, beispielsweise in der Simulation oder während dem Testen von Software und Hardware. Determinismus ist in diesen Bereichen oftmals eine gewünschte Eigenschaft.

Kryptographisch sichere Pseudozufallszahlengeneratoren Kryptographisch sichere Pseudozufallszahlengeneratoren (engl. *cryptographically secure pseudo random number generators (CSPRNGs)*) sind **PRNGs** mit einer speziellen zusätzlichen Eigenschaft: **CSPRNGs** sind unvorhersehbar. Grob gesprochen bedeutet dies, dass es rechentechnisch nicht möglich ist, aus n gegebenen Schlüsselstrombits s_1, s_2, \dots, s_n , die folgenden Bits $s_{n+1}, s_{n+2}, \dots, s_{n+m}$ zu berechnen. Außerdem soll es zeitlich unmöglich sein, eines der vorherigen Bits $s_{n-1}, s_{n-2}, \dots, s_{n-m}$ zu berechnen.

2.4 Das One-Time-Pad

Mit den bis hierher eingeführten Ideen zu Stromchiffren und den verschiedenen Zufallszahlengeneratoren sind alle Teile vorhanden, um ein beweisbar sicheres Verschlüsselungsverfahren zu bauen. Ein System heißt beweisbar sicher, wenn es trotz unendlich vorhandener

Rechenleistung nachweislich nicht gebrochen werden kann. Paar und Pelzl haben den Begriff mit folgender Definition beschrieben (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 (ebd., S. 37):

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

1. Der Schüsselstrom s_1, s_2, \dots, s_n wird durch einen **TRNG** generiert.
2. Nur vertrauenswürdige Gesprächspartner kennen den Schüsselstrom.
3. Jedes Schlüsselstrombit s_i wird nur einmal verwendet.

Das One-Time-Pad ist beweisbar sicher.

Zu jedem Bit im Geheimtext gibt es eine Gleichung der Form:

$$\begin{aligned} y_0 &\equiv x_0 + s_0 \pmod{2} \\ y_1 &\equiv x_1 + s_1 \pmod{2} \\ &\vdots \end{aligned}$$

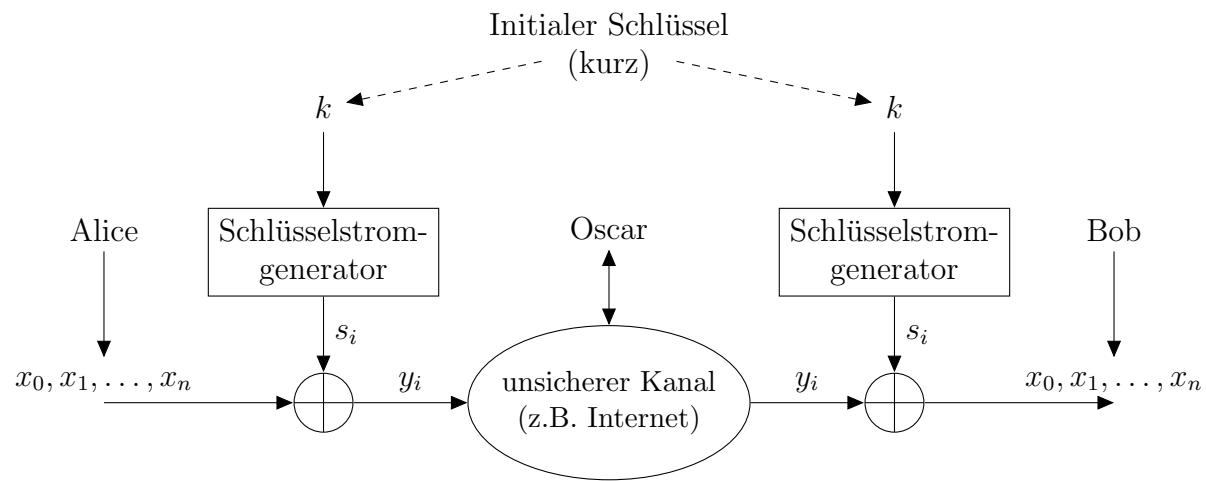
Stammt der Schüsselstrom von einem **TRNG** können diese Gleichungen nicht gelöst werden. Für jedes Geheimtextbit y_i gibt es genau zwei Lösungen mit gleich großer Wahrscheinlichkeit:

$$\begin{aligned} y = 0 &\equiv 1_x + 1_s \equiv 0_x + 0_s \pmod{2} \\ y = 1 &\equiv 1_x + 0_s \equiv 0_x + 1_s \pmod{2} \end{aligned}$$

2.5 Die praktischen Stromchiffren

In der Praxis hat das Verschlüsseln mit **OTP** natürlich einen Hacken, denn warum sonst sollten heutzutage andere Verfahren eingesetzt werden? Als erster Punkt ist die Anforderung eines **TRNG**, dieser benötigt spezielle Hardware und ist in der Regel nicht Teil

eines Standard Computers. Der zweite und wahrscheinlich größte Nachteil ist die Handhabung des Schlüsselstroms: Jedes Schlüsselstrombit s_i wird nur einmal verwendet. Das One-Time-Pad benötigt somit einen Schlüssel, welcher genauso lang ist wie die Nachricht selbst. Es wird 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.



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

Zu nächst 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 gestallten, wird der Angreifer in seiner Rechenleistung beschränkt, man hofft also, dass ein Verfahren berechenbarkeits-theoretisch sicher ist. Diese Eigenschaft ist von Paar und Pelzl folgendermaßen definiert ([ebd.](#), S. 35):

Definition 2.5.1 (Computational Security). „A cryptosystem is computationally secure if the best known algorithm for breaking it requires at least t operations.“

Die Definition besagt, dass ein System 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 Lösen des Faktorisierungsproblems mit großen natürlichen Zahlen. Obwohl viele Faktorisierungsverfahren bekannt sind, ist es nicht klar, ob es andere Verfahren gibt, welche 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 den drei Erfindern Ronald Rivest, Adi Shamir und Leonard Adleman, welche das Verfahren im Jahr 1977 veröffentlichten (Paar und Pelzl 2010, S. 173). In diesem Kapitel sollen die Ideen von asymmetrischer Verschlüsselung (engl. *public-key-cryptography* oder *asymmetric cryptography*) vorgestellt werden. Es wird hierbei genauer auf das RSA-Kryptosystem eingegangen und es werden einige zahlentheoretischen Grundlagen gezeigt.

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](#)).

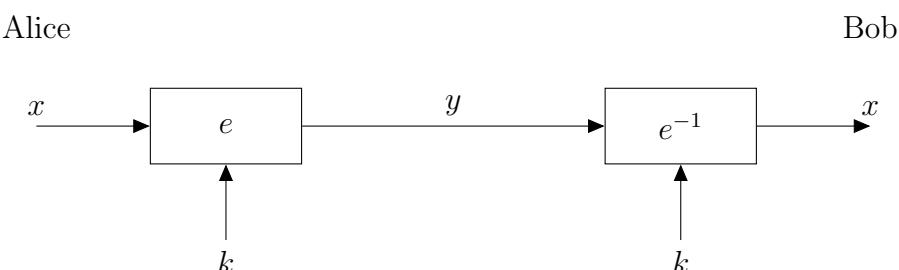


Abbildung 3.1: Das Prinzip der symmetrischen Verschlüsselung

Ein solches System hat zwei symmetrische Eigenschaften:

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

Symmetrische Verfahren wie **AES** sind sehr sicher und schnell, jedoch kommen sie nicht ohne Defizite, wie im Folgenden beschrieben wird.

Schlüsselverteilungsproblem Da beide Parteien den selben 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 $500 \cdot 499 = 249.500$ Schlüssel müssen verteilt werden. \triangle

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

Schutz vor Betrug von Alice und Bob 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 wirklich getätigt hat, anderenfalls könnte Bob jederzeit behaupten, Alice hätte seine Bestellung fälschlicherweise erstellt. \triangle

Der Begriff um dies zu verhindern nennt sich Non-Repudiation (McCullagh und Cealli 2000) und kann erreicht 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 eines geheimen Schlüssels ist. Um solch ein System zu realisieren veröffentlicht Bob einen öffentlichen Schlüssel. Dieser steht jedem Netzteilnehmer zur Verfügung und kann frei verwendet werden. Bob hat außerdem einen privaten Schlüssel, welchen nur er kennt. Bob's 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 Bob's öffentlichen Schlüssel verschlüsselt wurde, nur auch mit Bob's privaten Schlüssel wieder entschlüsselt werden kann. Ein einfaches Protokoll welches nach diesem Prinzip arbeitet ist in [Abbildung 3.2](#) zu sehen.

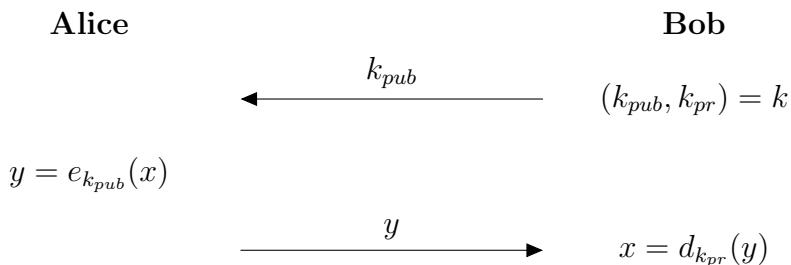


Abbildung 3.2: Das Prinzip der asymmetrischen Verschlüsselung

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 wünschenswert nicht dauerhaft asymmetrisch zu verschlüsseln, da dies im Gegensatz zu dem symmetrischen Gegenstück sehr viel rechenintensiver ist.

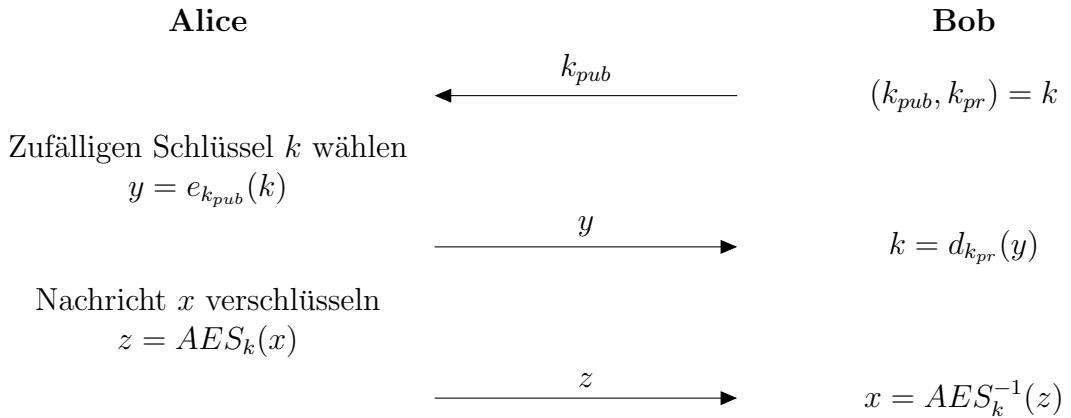


Abbildung 3.3: Schlüsselaustausch mit asymmetrischer Verschlüsselung

Asymmetrische Verfahren basieren alle auf einem zugrundeliegenden Prinzip: Der Einwegfunktion oder auch Falltürfunktion genannt. Es kann folgende Definition gegeben werden (Paar und Pelzl 2010, S. 153):

Definition 3.1.1 (Einwegfunktion). Eine Funktion f ist eine Einwegfunktion, wenn gilt:

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

Es gibt zwei Einwegfunktionen welche in der Praxis häufig eingesetzt werden. Die erste Funktion, welche in RSA verwendet wird, basiert auf dem Faktorisieren großer natürlicher Zahlen. Es ist einfach ein Produkt zu berechnen, jedoch ist es schwierig eine Zahl zu faktorisieren. Die zweite Funktion basiert auf dem Lösen diskreter Logarithmen. Das Lösen diskreter Logarithmen ist schwierig, das gesamte Verfahren hat jedoch keine so intuitive Beschreibung.

3.2 Zahlentheoretische Grundlagen

In diesem Abschnitt sollen die zahlentheoretischen Grundlagen beschrieben werden, welche im RSA-Kryptosystem zum Einsatz kommen.

3.2.1 Der Euklidische Algorithmus

Es wird begonnen mit dem Begriff des größten gemeinsamen Teiler. 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.

$$T(6) = \{1, 2, 3, 6\}$$

$$T(4) = \{1, 2, 4\}$$

$$T(6) \cap T(4) = \{1, 2\}$$

△

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 $g = g'$.

Aus $g|a$ und $g|b \Rightarrow g|a - b \Rightarrow g \in T(a - b) \cap T(b)$.

g ist wie g' gemeinsamer Teiler von $a - b$ und b .

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

Aus $g|a - b$ und $g|b \Rightarrow g|a$.
 $g|a$ und $g|b$ und $g > ggT(a, b)$ führt zum Widerspruch.

□

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

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

△

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:

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

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

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

△

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 :  $ggT(a, b)$ 
begin
     $r \leftarrow a \bmod b$ 
    if  $r = 0$  then
        return  $b$ 
    end if
    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 Kryptographie 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 auch relativ Prim (Symbol $a \perp b$, engl. *relatively prime* oder *coprime*). 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):

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

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, so dass $3 = 132 \cdot x + 51 \cdot y$. 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:

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

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

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:

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

Es ergeben sich die folgenden Regeln:

$$x_i = y_{i+1} \tag{3.3}$$

$$\begin{aligned} y_i + x_i \cdot q_i &= x_{i+1} \\ y_i &= x_{i+1} - x_i \cdot q_i \end{aligned} \tag{3.4}$$

$$y_i = x_{i+1} - y_{i+1} \cdot q_i$$

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
0	132	51	2	30	-5	$3 + 5 \cdot 2 = 13$	$3 = -5 \cdot 132 + 13 \cdot 51$
1	51	30	1	21	3	$-2 - 3 \cdot 1 = -5$	$3 = 3 \cdot 51 - 5 \cdot 30$
2	30	21	1	9	-2	$1 + 2 \cdot 1 = 3$	$3 = -2 \cdot 30 + 3 \cdot 21$
3	21	9	2	3	1	$0 - 1 \cdot 2 = -2$	$3 = 1 \cdot 21 - 2 \cdot 9$
4	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 $a \pmod{m}$ 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 $21^{-1} \pmod{89}$ bestimmt werden. Die Zahlen 21 und 89 sind teilerfremd, d.h. das Inverse existiert und es gilt $\text{ggT}(89, 21) = 1 = x \cdot 89 + y \cdot 21$. Der Euklidische Algorithmus kann tabellarisch durchgeführt werden:

i	a	b	q	r	x	y
0	89	21	4	5	-4	$1 + 4 \cdot 4 = 17$
1	21	5	4	1	1	$0 - 1 \cdot 4 = -4$
2	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:

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

△

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.

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

3.2.3 Die Eulersche Phi-Funktion

Die Eulersche Phi-Funktion befasst sich mit dem auf den ersten Blick seltsamen Problem, die Anzahl von 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.

$$\begin{aligned}\varphi(5) &= |\{1, 2, 3, 4\}| = 4 \\ \varphi(6) &= |\{1, 5\}| = 2 \\ \varphi(20) &= |\{1, 3, 7, 9, 11, 13, 17, 19\}| = 8\end{aligned}$$

△

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 (Spannagle 2012b):

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:

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

□

Beispiel 3.2.7.

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

△

3.2.4 Satz von Euler und Fermat

Es werden im Folgenden zwei Sätze vorgestellt von Euler und Fermat, welche in der asymmetrischen Verschlüsselung sehr hilfreich sind.

Satz 3.2.2 (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. Aus der Vorlesung zur Zahlentheorie (Spannagle 2012b) und in abgewandelter Form (Remmert und Ullrich 2008, S. 187–188).

Es ist $n = \varphi(m)$. Seien x_1, x_2, \dots, x_n die n verschiedenen zu m teilerfremden Zahlen aus

der Menge $\{1, 2, \dots, m\}$. Mit einer vorgegebenen Zahl $a \in \mathbb{N}^\times$ bilden wir die Produkte ax_1, ax_2, \dots, ax_n . Es gilt $a \perp m$ und $x_i \perp m$, weshalb auch das Produkt ax_i teilerfremd zu m ist. Es kann nun die wahre Aussage aufgestellt

$$ax_1 \cdot ax_2 \cdot \dots \cdot ax_n \equiv ax_1 \cdot ax_2 \cdot \dots \cdot ax_n \pmod{m}$$

und die folgende Beobachtung gemacht werden:

$$ax_1 \cdot ax_2 \cdot \dots \cdot ax_n \equiv x_1 \cdot x_2 \cdot \dots \cdot x_n \pmod{m} \quad (*)$$

Um dies zu verifizieren betrachten wir zuerst ein Beispiel (es wird in \mathbb{Z}_8 gerechnet):

$$\begin{aligned} \varphi(8) &= 4 \\ x_1, x_2, x_3, x_4 &= 1, 3, 5, 7 \\ 3x_1, 3x_2, 3x_3, 3x_4 &= 3, 1, 7, 5 \end{aligned}$$

Die Multiplikation von a in Z_m verändert nicht die Menge $\{x_1, x_2, \dots, x_n\}$, für $i \neq j$ muss also gelten:

$$ax_i \not\equiv ax_j \pmod{m}$$

Diese Vermutung kann bestätigt werden. Angenommen $ax_i \equiv ax_j \pmod{m}$, da $a \perp m$ folgt nach der Kürzungsregel $x_i \equiv x_j \pmod{m}$, was nicht sein kann. Es folgt nun aus der Beobachtung (*): 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 3.2.3 (Kleiner Satz von Fermat). *Es seien $a \in \mathbb{N}^\times$ 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 Kryptographie unter anderem dafür verwendet, um Primzahltests durchzuführen.

3.3 Das RSA-Verfahren

Das RSA-Verfahren basiert auf der Annahme, dass die Berechnung der Primfaktorzerlegung einer natürlichen Zahl, ein schwieriges Problem ist. Die Generierung des privaten und öffentlichen Schlüssels kann in die folgenden fünf Schritte unterteilt werden (Paar und Pelzl 2010, S. 176):

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 $1 < e < N$ und $ggT(e, \varphi(N)) = 1$. Jetzt bilden (N, e) den öffentlichen Schlüssel (k_{pub}).
5. Berechne d mit $e \cdot d \equiv 1 \pmod{\varphi(N)}$ und dem erweiterten Euklidischen Algorithmus. Das Inverse existiert, da $e \perp \varphi(N)$ gilt. Jetzt bilden (N, d) den privaten Schlüssel (k_{pr}).

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 (Ver- und Entschlüsselung RSA). Es seien der öffentliche und private Schlüssel gegeben mit $k_{pub} = (N, e)$ und $k_{pr} = (N, d)$. Dann gilt:

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, welcher im Beweis verwendet werden kann:

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

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

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

$$\begin{aligned} a - b &= t_0 \cdot m \\ a - b &= t_1 \cdot k_1 \\ &\vdots \\ a - b &= t_n \cdot k_n \end{aligned}$$

Wobei t_i irgendwelche ganzen Zahlen sind. Es lässt sich schreiben:

$$t_0 \cdot m = t_1 \cdot k_1 = \dots = t_n \cdot k_n$$

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 die Beweisen werden, warum das RSA-Verfahren funktioniert:

Beweis. Die Entschlüsselung des Geheimtextes muss erneut den Klartext ergeben.

$$x = d_{k_{pr}}(e_{k_{pub}}(x)) \equiv (x^e)^d \equiv x^{ed} \pmod{N} \quad (3.5)$$

Wir wissen per Definition, wobei t irgendeine ganze Zahl ist:

$$\begin{aligned} e \cdot d &\equiv 1 \pmod{\varphi(N)} \\ \Leftrightarrow e \cdot d &= t \cdot \varphi(N) + 1 \end{aligned} \quad (3.6)$$

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

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

- (a) Es gilt $p \mid x$ und man kann schreiben: $x \equiv 0 \pmod{p}$ woraus folgt $x^{ed} \equiv 0 \equiv x \pmod{p}$
- (b) Es gilt $p \perp x$, einsetzen von (3.6) in (3.5) 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 ist jetzt zu sehen:

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

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

□

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

Die Ergebnis stimmt, denn es gilt $e \cdot d = 3 \cdot 7 = 21 \equiv 1 \pmod{20}$.

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

△

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 versucht, die visuelle Qualität des Urbilds zu bewahren. Zusätzlich wird auf eine Anwendung eingegangen, welche die beschriebenen Ideen umgesetzt 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 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 des **LSB** verändert den Farbwert nur minimal und die kleinen Abweichungen werden nur durch betrachten des veränderten Bilds nicht zu erkennen sein.

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 $b_7 b_6 \dots b_0$ 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 inklusiv 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 Fehlerparamter $0 \leq n \leq 8$. 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. 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.



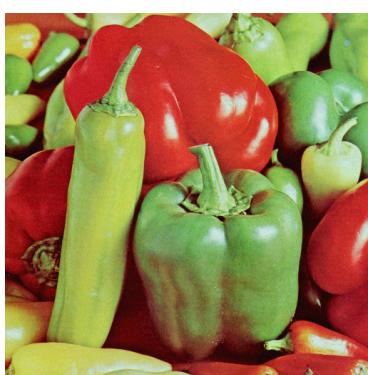
$n = 0$ (Original)



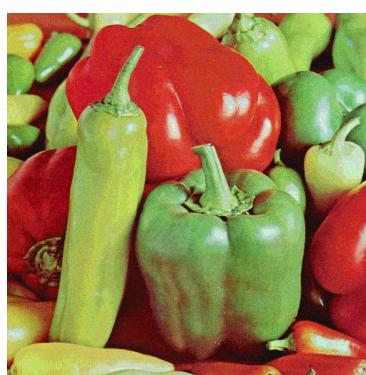
$n = 1$



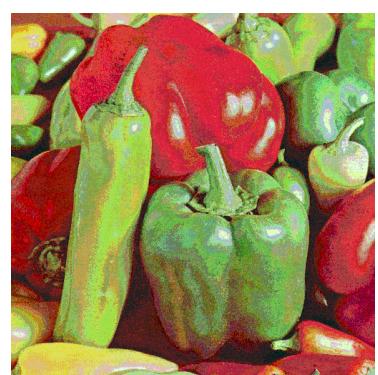
$n = 2$



$n = 3$



$n = 4$



$n = 5$



$n = 6$



$n = 7$



$n = 8$

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

4.2 Beschreibung eines Algorithmus

Im vorherige 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 kryptographischen Algorithmus zu verschlüsseln.

Definition 4.2.1 (LSB-Verfahren). Es sei $p_{xy} = (r, g, b)$ 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 $v : P \setminus \{p_{1,1}, p_{mn}\} \leftarrow v(\mathbf{B})$. [Algorithmus 4.1](#) und [4.2](#) zeigen ein Verfahren für das Schreiben und Lesen einer Nachricht in \mathbf{B} .

Algorithmus 4.1 : LSB-Verfahren Schreiben
--

<p>Input : \mathbf{B} und Nachricht x Output : \mathbf{B} mit versteckter Nachricht y</p> <p>begin</p> <p style="margin-left: 2em;">$y \leftarrow e_k(x)$ $n \leftarrow 1$ schreibe Nachrichtenlänge von y nach p_{11} und p_{mn}</p> <p style="margin-left: 2em;">while true do</p> <p style="margin-left: 3em;">if $n = 9$ then y ist zu lang</p> <p style="margin-left: 3em;">for $p \in v(\mathbf{B})$ do</p> <p style="margin-left: 4em;">for $c \in p$ do</p> <p style="margin-left: 5em;">$b \leftarrow$ lese nächstes Bit von y</p> <p style="margin-left: 5em;">schreibe b nach Position n von c</p> <p style="margin-left: 5em;">if y bearbeitet then return</p> <p style="margin-left: 4em;">end for</p> <p style="margin-left: 3em;">end for</p> <p style="margin-left: 2em;">$n \leftarrow n + 1$</p> <p style="margin-left: 2em;">end while</p> <p>end</p>

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  $p_{11}$  und  $p_{mn}$ 
    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 for
        end for
         $n \leftarrow n + 1$ 
    end while
end

```

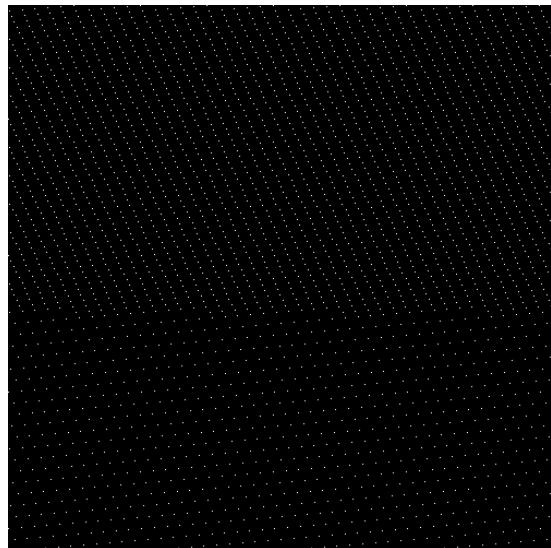
Damit eine Nachricht im Bild möglichst wenig auffällt, macht es Sinn, Pixel so auszuwählen, dass diese gleichmäßig verteilt sind. Die Verteilfunktion v hat genau diese Aufgabe. Das Bild **B** wird als Folge der natürlichen Zahlen $a = 0, 1, \dots, mn - 1$ betrachtet. Durch wiederholtes halbieren von a und speichern der entstehenden Hälften in einer Warteschlange, kann die Folge nach dem Prinzip der Breitensuche abgearbeitet werden und es entstehen relativ gleichmäßig verteilte Koordinaten.

Beispiel 4.2.1. Ein wird ein 3×3 großes Bild betrachtet und die Zahlenfolge ist $a = 0, 1, 2, 3, 4, 5, 6, 7, 8$. Die nachfolgende Tabelle zeigt den Verlauf des Algorithmus:

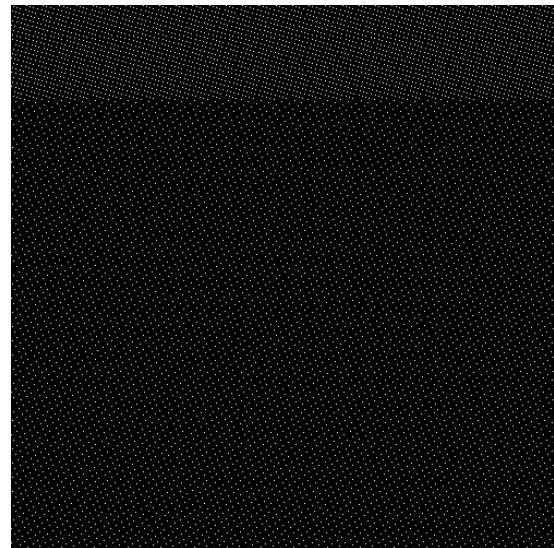
Warteschlange	Mitte	Umformung	Koordinate
[0, 8]	m	$m = y \cdot 3 + x$	(x, y)
[5, 8], [0, 3]	4	$4 = 1 \cdot 3 + 1$	(1, 1)
[2, 3], [0, 0], [5, 8]	1	$1 = 0 \cdot 3 + 1$	(1, 0)
[7, 8], [5, 5], [2, 3], [0, 0]	6	$6 = 2 \cdot 3 + 0$	(0, 2)
[7, 8], [5, 5], [2, 3]	0	$0 = 0 \cdot 3 + 0$	(0, 0)
[3, 3][2, 2], [7, 8], [5, 5]	-	-	-
[3, 3][2, 2], [7, 8]	5	$5 = 1 \cdot 3 + 2$	(2, 1)
[8, 8], [7, 7], [3, 3], [2, 2]	-	-	-
[8, 8], [7, 7], [3, 3]	2	$2 = 0 \cdot 3 + 2$	(2, 0)
[8, 8], [7, 7]	3	$3 = 1 \cdot 3 + 0$	(0, 1)
[8, 8]	7	$7 = 2 \cdot 3 + 1$	(1, 2)
\emptyset	8	$8 = 2 \cdot 3 + 2$	(2, 2)

△

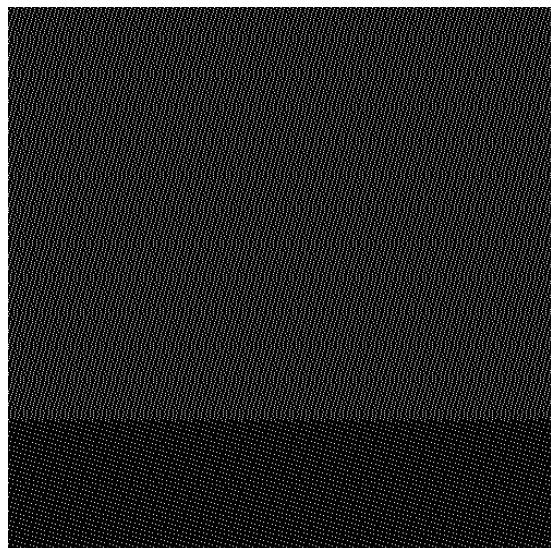
Werden nicht null indizierte Koordinaten benötigt, können diese entsprechend skaliert werden. [Abbildung 4.2](#) zeigt die Koordinatenverteilung für Nachrichten unterschiedlicher Längen auf einem 500×500 Farbbild mit schwarzen Hintergrund. Die durch den Algorithmus errechneten Koordinaten sind weiß eingefärbt.



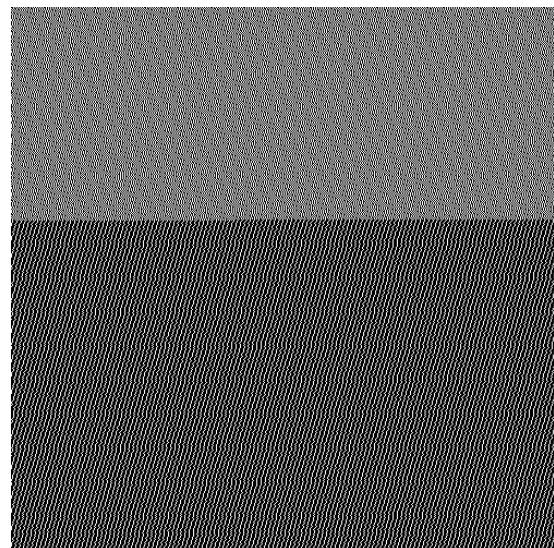
1200 Byte



3600 Byte



10800 Byte



32400 Byte

Abbildung 4.2: Koordinatenverteilung auf einem 500×500 Pixel Schwarzbild für Nachrichtenlängen von 1200, 3600, 10800 und 32400 Byte.

4.3 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 verstecken Informationen gewährleisten. Die wichtigsten Sicherheitsaspekte eines solchen Systems sind die Folgenden:

1. **Vertraulichkeit** einer Nachricht (Geheimhaltung, Verschlüsselung).
2. **Integrität** einer Nachricht (Hashfunktion, MAC).
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, beispielsweise für das Schreiben im Bild, Benutzer gebunden 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.

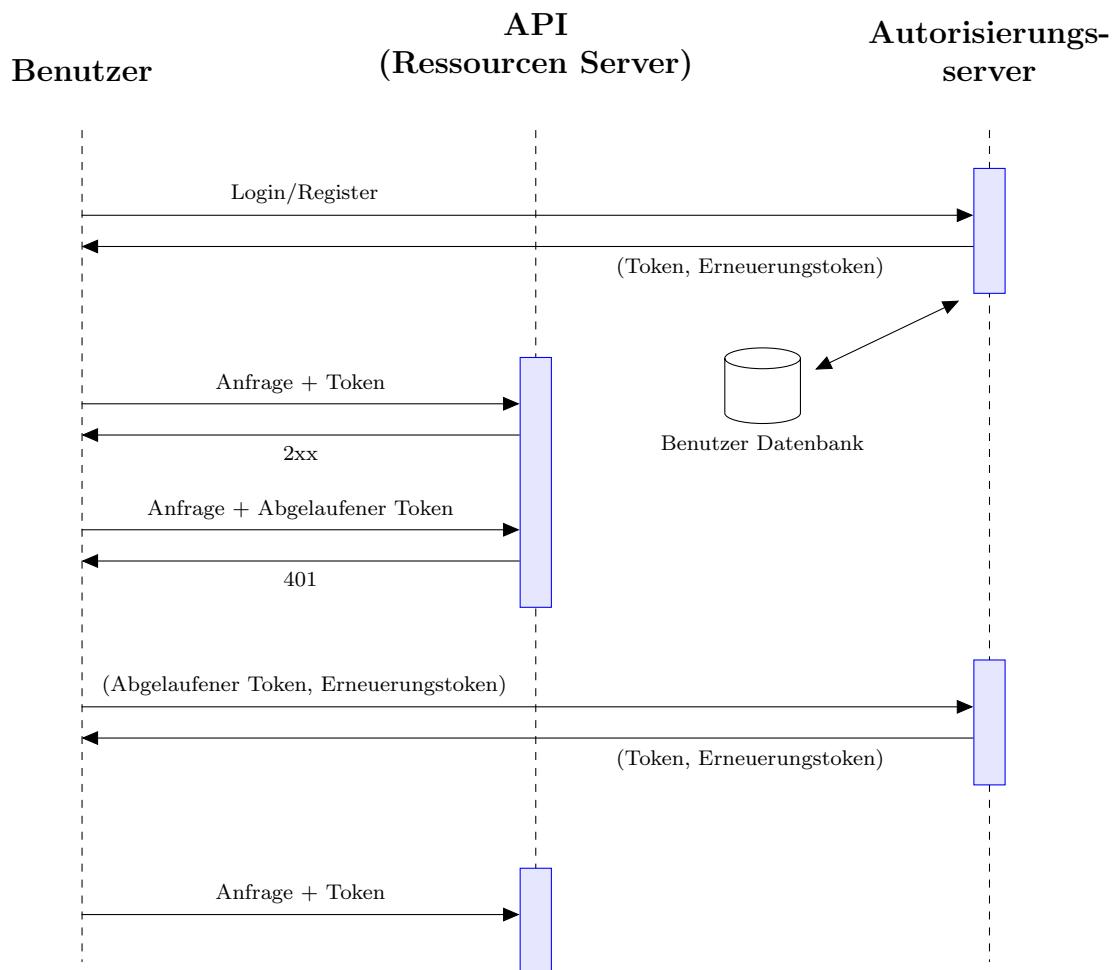


Abbildung 4.3: Sequenzdiagramm **JWT** Autorisierung

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.

4.4 Sicherheitsanalyse

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

4.4.1 Risiken der Token basierten Autorisierung

Besondere Vorsicht muss geboten werden, wenn es um die Handhabung der Zugriffstoken geht. Erlangt ein Angreifer Besitzt eines Tokens, kann dieser im Namen des Opfers beliebig Anfragen durchführen, d. h. geheime Nachrichten lesen und auch Nachrichten schreiben. Zugriffstoken werden Clientseitig gespeichert. Über eine Clientumgebung (wie beispielsweise dem Webbrowser) 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 Angriffsarten, 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 deshalb 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.

4.4.2 Kryptographische 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 wird¹. Als Garantie, dass nur der geplante Empfänger eine Nachricht lesen kann, ist ein eindeutiger *purposes parameter* nötig, welcher der Ver- und Entschlüsselungsfunktion übergeben wird. Der Parameter setzt sich zusammen aus (Datenbank ID || - || Name) und ist somit garantiert eindeutig, das Symbol || beschreibt die Konkatenation. [Algorithmus 4.3](#) und [4.4](#) zeigen das Prinzip der Verschlüsselung. Die *Data Protection API* stellt eine Methode `CreateProtector` zur Verfügung:

Algorithmus 4.3 : Data Protection Encryption
Input : message x and information of the receiver ($id, username$) Output : protected message y begin $(e_k, d_k) \leftarrow \text{CreateProtector}(id \parallel - \parallel username)$ return $e_k(x)$ end

Es ist jetzt zu sehen, warum eine Anfrage in Verbindung mit einem Benutzer gebracht werden muss:

Algorithmus 4.4 : Data Protection Decryption
Input : protected message y Output : message x begin $(id, username) \leftarrow \text{get user information of request}$ $(e_k, d_k) \leftarrow \text{CreateProtector}(id \parallel - \parallel username)$ return $d_k(y)$ end

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

4.4.3 Steganografische Qualität

Um die Qualität des Algorithmus zu beurteilen, lohnt es sich vor allem einige Beispiele anzusehen. Die steganografische Qualität des Verfahrens soll jetzt anhand der Folgenden Bilder untersucht werden.



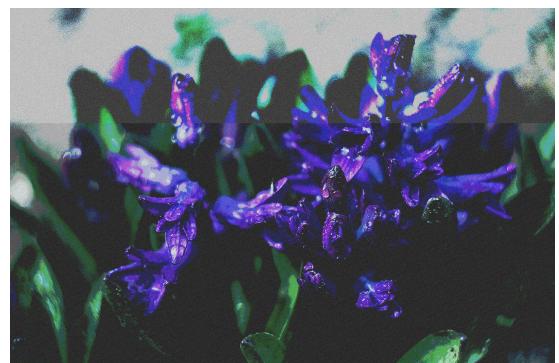
Original



10 MB Nachricht



20 MB Nachricht

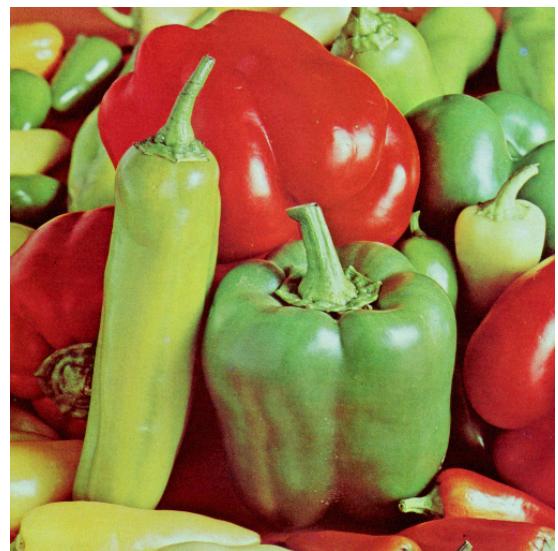


30 MB Nachricht

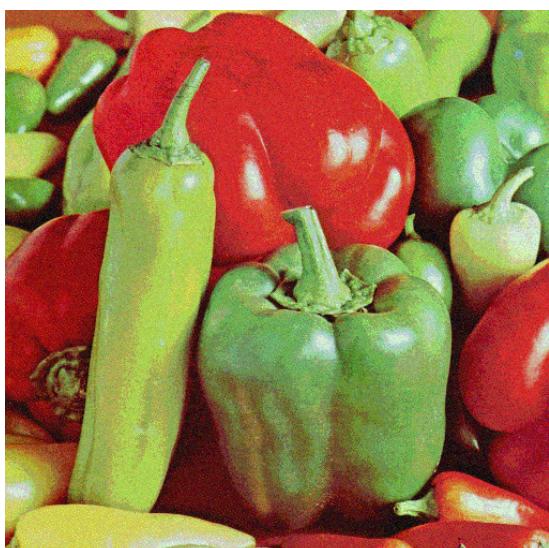
Abbildung 4.4: Blumen. Größe 4272×2848 Pixel. Maximale Kapazität $\approx 36,5$ MB. Gute Bildqualität bis zu 20MB versteckter Nachricht.



Original



300 KB Nachricht



500 KB Nachricht



700 KB Nachricht

Abbildung 4.5: Paprika. Größe 509×509 Pixel. Maximale Kapazität ≈ 777 KB. Gute Bildqualität bis zu 300-500 KB versteckter Nachricht.

Sind die Seitenverhältnisse eines Bilds Zweierpotenzen, entstehen Streifen, welche einfacher erkannt werden können. [Abbildung 4.6](#) zeigt dieses Verhalten, die veränderte Pixel sind weiß eingefärbt.

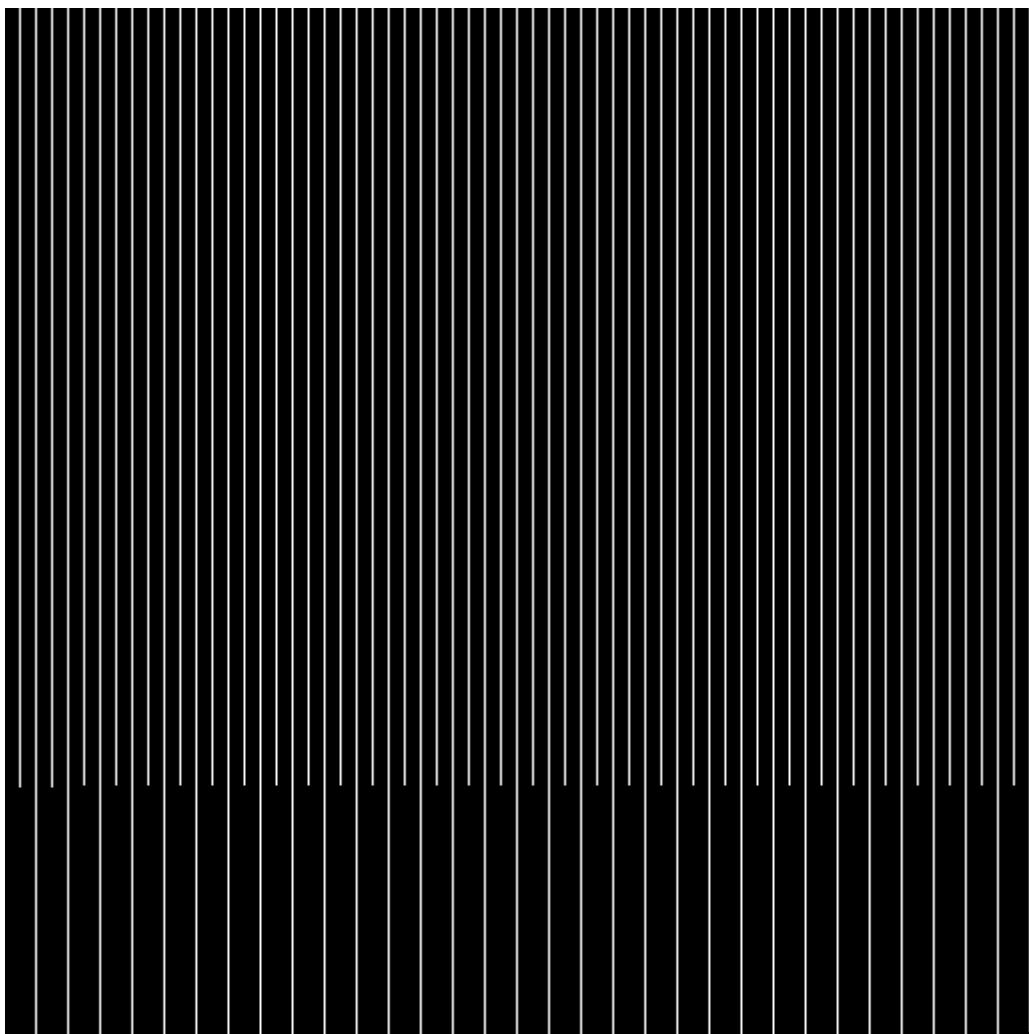


Abbildung 4.6: Schwarz. Größe 512×512 Pixel. 10 KB Nachricht, veränderte Pixel sind weiß eingefärbt.

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.
- (Juni 2012a). *Diophantische Gleichungen*. Einsichtsname: 21.01.2021. URL: 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.

Glossar

AES Advanced Encryption Standard. [11](#), [21](#), [22](#)

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

DES Data Encryption Standard. [11](#), [21](#)

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

LSB *least significant bit.* [V](#), [38](#), [40](#), [41](#)

OTP One-Time-Pad. [17](#), [18](#), [21](#)

PRNG *pseudo random number generator.* [16](#), [18](#)

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

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