Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

Kryptologische Methoden und Alternativen

Bachelorarbeit

im Studiengang Informatik

von

Phillip Luttmann
Matrikelnummer: 2942830

Hannover, 06.09.2015

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

_____

Abbildungsverzeichnis

2.1 Schematische Darstellung von kryptografischen Systemen 13

3.1 Beispiel für eine Permutation des lateinischen Alphabets 20
3.2 Erwartete Häufigkeitsverteilung der Buchstaben in längeren deutschen Texten, die Umlaute Ä, Ö, Ü wurden wie AE, OE, UE behandelt und ß wurde durch SS ersetzt [Freiermuth et al., 2014, S. 77] 22
3.3 Beispiel für eine Vigenère-Verschlüsselung 23
3.4 Matrixdarstellung eines 128-Bit-Blocks durch Byte-Variablen im AES-Algorithmus 26
3.5 Anzahl der AES-Runden in Abhängigkeit von Block- und Schlüssellänge . 26
3.6 AES-SBOX 28
3.7 ShiftRow auf einem 128-Bit-Block im AES-Verfahren 28

4.1 Veranschaulichung des erzeugenden Elements 3 in der Gruppe $(\mathbb{Z}_7, \cdot_7)$ . . . 34
4.2 Man-in-the Middle – alle Daten fließen über einen Angreifer 37
4.3 Die elliptische Kurve $y^2 = x^3 - 3x + 5$ 37

5.1 Die 9 am häufigsten verwendeten Passwörter [Bube, 2014] 44
5.2 Schematische Darstellung der magischen Tür 49
5.3 Ablauf einer Protokollrunde im Fiat-Shamir-Protokoll zwischen dem Beweiser A und dem Verifizierer B 50
5.4 Schematische Darstellung von digitalen Signaturen unter Einbeziehung von Hashfunktionen 52

6.1 Schematische Darstellung von steganografischen Systemen 54
6.2 Beispiel für eine einfache verschleierte Geheimschrift durch eine binäre Codierung [Nowka, 2004] 57
6.3 Vergleich von zwei minimal abweichenden Farben im RGB-Format – links die Farbe (30, 144, 255), rechts die Farbe (31, 145, 254) 58
6.4 Pixelbeschreibung eines 3×3 Bildes vor und nach dem Einfügen der Nachricht 101101010 mit dem LSB-Verfahren 59
6.5 Pixelbeschreibung eines 3×3 Bildes vor und nach dem Einfügen der Nachricht 00101 mit dem schlüsselgesteuertem LSB-Verfahren und dem Schlüssel 00211 59
6.6 Pixelbeschreibung eines 3×3 Bildes vor und nach dem Einfügen der Nachricht 001 mit dem LSB-Verfahren mit Paritätskodierung und dem Parameter $n = 3$ 60

Liste der Algorithmen

1 Caesar-Verschlüsselung 19
2 Caesar-Entschlüsselung 19
3 Verschlüsselung mit Substitutions-Chiffren 20
4 Entschlüsselung mit Substitutions-Chiffren 20
5 Vigenère-Verschlüsselung 23
6 Vigenère-Entschlüsselung 24
7 One-Time-Pad 25
8 AES-Verschlüsselung 27

9 RSA Verschlüsselung 32
10 RSA Entschlüsselung 32
11 Punktaddition in elliptischen Kurven 38
12 Verschlüsselung im McEliece Verfahren 40
13 Entschlüsselung im McEliece Verfahren 40

Inhaltsverzeichnis

1 Einleitung **11**

2 Grundlagen **12**
2.1 Kryptosysteme . . . 12
2.2 Sicherheit von Kryptosystemen . . . 12
2.3 Angriffe auf Kryptosysteme . . . 14
2.4 Blockchiffren und Stromchiffren . . . 14
2.5 Betriebsmodi von Blockchiffren . . . 15
2.5.1 Electronic Codebook Mode (ECB) . . . 15
2.5.2 Cipherblock Chaining Mode (CBC) . . . 15
2.5.3 Cipher Feedback Mode (CFB) . . . 16
2.5.4 Output Feedback Mode (OFB) . . . 16
2.5.5 Counter Mode (CTR) . . . 16
2.6 Hashfunktionen . . . 17

3 Symmetrische Verschlüsselung **18**
3.1 Monoalphabetische Substitutionschiffren . . . 18
3.1.1 Caesar-Verschlüsselung . . . 18
3.1.2 Substitutions-Chiffren . . . 19
3.2 Polyalphabetische Substitutionschiffren . . . 23
3.2.1 Vigenère-Verschlüsselung . . . 23
3.2.2 One-Time-Pad . . . 24
3.3 Advanced Encryption Standard (AES) . . . 25
3.3.1 Einführung . . . 26
3.3.2 Schlüsselexpansion . . . 27
3.3.3 SubBytes . . . 27
3.3.4 ShiftRow . . . 28
3.3.5 MixColumn . . . 28
3.3.6 Sicherheit von AES . . . 29
3.4 Alternativen zu AES . . . 29
3.5 Probleme von symmetrischen Algorithmen . . . 30

4 Asymmetrische Verschlüsselung **31**
4.1 RSA . . . 31
4.1.1 Schlüsselerzeugung . . . 31
4.1.2 Verschlüsselung und Entschlüsselung . . . 31

4.1.3 Sicherheit von RSA . . . 32
4.1.3.1 Berechnung der Primfaktorzerlegung . . . 33
4.1.3.2 Angriffe auf kurze Nachrichten . . . 33
4.1.3.3 Bedrohung durch Quantencomputer . . . 33
4.2 Diffie-Hellman-Schlüsselaustausch . . . 34
4.2.1 Mathematische Grundlagen . . . 34
4.2.2 Schlüsselaustausch . . . 35
4.2.3 Sicherheit von Diffie-Hellman . . . 35
4.2.3.1 Algorithmen zur Berechnung des diskreten Algorithmus . 35
4.2.3.2 Man-in-the-Middle-Angriff . . . 36
4.2.4 Diffie-Hellman auf elliptischen Kurven . . . 37
4.3 Code-based Kryptografie: McEliece Verfahren . . . 38
4.3.1 Schlüsselerzeugung . . . 39
4.3.2 Verschlüsselung . . . 39
4.3.3 Entschlüsselung . . . 40
4.3.4 Eigenschaften des McEliece-Verfahren . . . 41
4.4 Vergleich mit symmetrischen Algorithmen . . . 42

5 Methoden der Authentifizierung 43
5.1 Einführung . . . 43
5.1.1 Authentifizierung durch Wissen . . . 44
5.1.1.1 Passwörter . . . 44
5.1.1.2 Passgesten . . . 44
5.1.1.3 Persönliche Informationen . . . 45
5.1.2 Authentifizierung durch Besitz . . . 45
5.1.2.1 Personalausweis . . . 45
5.1.2.2 Einmalkennwort . . . 45
5.1.3 Authentifizierung durch Eigenschaft . . . 46
5.2 Protokolle . . . 46
5.2.1 Passwortbasierte Authentifizierung . . . 47
5.2.2 Challenge-Response-Authentifikation . . . 47
5.2.3 Zwei-Faktor-Authentifizierung . . . 47
5.2.4 Zero-Knowledge-Verfahren . . . 48
5.2.4.1 Die magische Tür . . . 48
5.2.4.2 Fiat-Shamir-Protokoll . . . 49
5.3 Authentifizierung mit Signaturen . . . 51

6 Steganografie 53
6.1 Grundlagen . . . 53
6.2 Altertümliche Methoden . . . 55
6.2.1 Verwendung von Geheimtinte . . . 55
6.2.2 Steganografie in historischen Konflikten . . . 55
6.3 Linguistische Steganografie . . . 55
6.3.1 Semagramme . . . 56

6.3.2 Open Codes . . . 56
6.3.2.1 Maskierte Geheimschriften . . . 56
6.3.2.2 Verschleierte Geheimschriften . . . 56
6.4 Steganografie mit digitalen Bildern . . . 57
6.4.1 Grundlagen zu digitalen Bildern . . . 58
6.4.2 Algorithmen . . . 58
6.4.2.1 LSB-Methode . . . 58
6.4.2.2 LSB-Methode mit schlüsselgesteuerter Pixelauswahl . . . 59
6.4.2.3 LSB-Methode mit Paritätskodierung . . . 60
6.5 Digitale Wasserzeichen . . . 60
6.6 Spreu-und-Weizen-Algorithmus . . . 61
6.7 Vergleich mit der Kryptografie . . . 63

7 Fazit **64**

1 Einleitung

Der erste Gedanke bei dem Begriff Kryptografie (altgr. für *kryptós* „verborgen“ und *gráphein* „schreiben“), geht bei vielen Menschen Richtung Geheimdienste, Spione und Verschwörungen. Doch die Kryptografie ist im 21. Jahrhundert so präsent, dass die meisten Menschen damit täglich zu tun haben. Grund dafür ist hauptsächlich das Internet und die zunehmende Vernetzung der Welt. Ohne Kryptografie würden die meisten der Internetanwendungen wie E-Mail, Onlineshopping oder Onlinebanking nicht funktionieren.

Die Ursprünge der Kryptografie gehen viele Jahrhunderte zurück, schon die Römer unter Cäsar und die alten Griechen wussten ihre Kommunikation zu verschlüsseln, damit ihre Feinde abgefangene Nachrichten nicht lesen können. Die ursprüngliche Wissenschaft der Kryptografie, hat sich hauptsächlich mit der Verschlüsselung von Daten befasst. Es geht um die Frage, wie man Nachrichten so verschlüsseln kann, dass ein Dritter nicht in der Lage, ist die Nachricht zu lesen.

Die moderne Kryptografie befasst sich, neben dem Schutz der Vertraulichkeit durch Verschlüsselung, zusätzlich mit der Integrität von Daten sowie der Authentizität. Dies ist beispielsweise für die Kommunikation zwischen einem Client und einem Server wichtig, da so garantiert werden kann, dass die Nachrichten vom richtigen Absender kommen und nicht verändert wurden.

Die Kryptoanalyse hingegen befasst sich mit der Analyse von Geheimtexten. Sie untersucht Möglichkeiten, um aus einem Geheimtext Informationen zu extrahieren – ohne im Besitz des Schlüssels zu sein. Die Kryptografie bildet zusammen mit der Kryptoanalyse den Begriff der Kryptologie.

Neben der Kryptografie existiert die Wissenschaft der Steganografie, welche sich ebenfalls mit dem Schutz der Vertraulichkeit von Nachrichten beschäftigt. Sie unterscheidet sich allerdings grundlegend im Hinblick auf die verwendeten Methoden. Es geht hauptsächlich darum, Nachrichten so zu übertragen, dass die Übertragung erst gar nicht bemerkt wird. Dies geschieht in der Regel durch die Einbettung einer Nachricht in einer zweiten Nachricht. Die Existenz der zweiten Nachricht ist jedoch nur dem Sender und dem Empfänger bekannt.

In Reaktion auf die NSA-Affäre, wiederholte Nachrichten über gestohlene Datensätze sowie der öffentlichen Angst vor unsicherer Kommunikation, soll in dieser Arbeit ein Überblick über bestehende Verfahren und Methoden der Kryptografie erstellt werden. Zudem soll die Wissenschaft der Steganografie untersucht werden, um die Frage zu klären, inwieweit sich die Steganografie als Alternative zur Kryptografie anbietet.

2 Grundlagen

In diesem Kapitel sollen zunächst einige grundlegende Begriffe der Kryptografie vorgestellt werden, die dem Verständnis der nachfolgenden Kapitel dienlich sein können.

2.1 Kryptosysteme

Ein *kryptografisches System* [Karpfinger und Kiechle, 2009, S. 9] (oder auch *Kryptosystem*) ist ein 5-Tupel $(\mathbb{P}, \mathbb{C}, \mathbb{K}, f, g)$ mit nicht-leeren Mengen $\mathbb{P}, \mathbb{C}, \mathbb{K}$ und Abbildungen

$$f\colon \mathbb{P} \times \mathbb{K} \to \mathbb{C} \text{ und } g : \mathbb{C} \times \mathbb{K} \to \mathbb{P}$$

mit der Eigenschaft:

$$\forall k \ \in \mathbb{K}\ \exists k' \in \mathbb{K} : g(f(x,k),k') = x\ \forall x \in \mathbb{P}.$$

Es heißen:

- $\mathbb{P}$ die *Klartextmenge* ($\mathbb{P}$ wie plain text),
- $\mathbb{C}$ die *Geheimtextmenge* ($\mathbb{C}$ wie cipher text),
- $\mathbb{K}$ die *Schlüsselmenge* ($\mathbb{K}$ wie key),
- f *Verschlüsselungsfunktion*,
- g *Entschlüsselungsfunktion.*

Kryptosysteme in denen $k = k'$ für alle k gilt, nennt man symmetrisch. Im anderen Fall nennt man das Kryptosystem asymmetrisch.

In Abbildung 2.1 ist das Schema von Kryptosystemen visualisiert. Zunächst wird die zu übertragende Nachricht m unter Zuhilfenahme des Schlüssels k mit der Funktion f verschlüsselt. Nachdem sie über einen Kanal übertragen wurde, kann der Empfänger sie mit der Funktion g entschlüsseln. Hierzu braucht er den Schlüssel k'.

2.2 Sicherheit von Kryptosystemen

Ein Kryptosystem gilt als perfekt sicher, wenn ein Geheimtext zu jedem möglichen Klartext der gleichen Länge mit derselben Wahrscheinlichkeit von einem Angreifer zugeordnet wird. Das bedeutet, dass ein Angreifer, der einen Geheimtext abfängt, nicht in der Lage sein darf, aus diesem Informationen (bis auf die Länge) über den Klartext zu extrahieren.

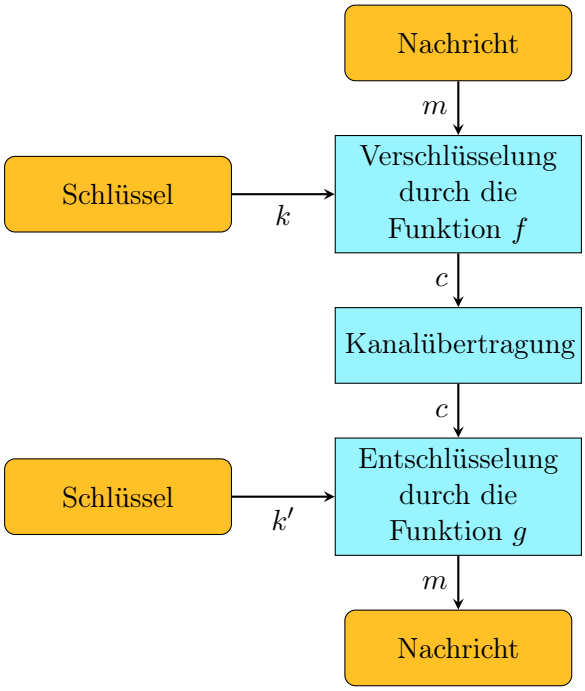

Abbildung 2.1: Schematische Darstellung von kryptografischen Systemen

Diese Aussage lehnt sich an einer Definition von Claude Elwood Shannon aus dem Jahr 1949 über perfekte Sicherheit an [Shannon, 1949].

Diese Aussage täuscht jedoch über etwas hinweg. Falls es möglich ist, mit *vertretbarem* Aufwand alle möglichen Klartexte zu berechnen, kann das System nicht sicher sein, da in der Regel nur eine sehr kleine Teilmenge der Klartexte eine sinnvolle Nachricht darstellt. Für eine *praktisch* hohe Sicherheit wird also zusätzlich ein möglichst großer Schlüsselraum benötigt.

Neben seiner Arbeit zu perfekter Sicherheit, hat Shannon vor allem zwei weitere Begriffe geprägt, die heute als Designziele für kryptografische Algorithmen gelten, um eine möglichst hohe Sicherheit zu gewährleisten. Eine hohe Konfusion bedeutet, dass es sehr schwer ist vom Geheimtext auf den verwendeten Schlüssel zu schließen. Diffusion hingegen ist ein Prinzip zur Auflösung von statistischen Strukturen eines Klartextes im Zuge einer Verschlüsselung. Ein Beispiel für eine statistische Struktur eines Klartextes ist die Buchstabenhäufigkeit.

Ein anderes wichtiges Prinzip für die Sicherheit von kryptografischen Algorithmen wurde bereits 1883 von Kerckhoff [Kerckhoffs, 1883] vorgeschlagen und wird bis heute von den meisten Algorithmen berücksichtigt. So besagt das *Kerckhoffs'sche Prinzip:*

> Die Sicherheit eines Verschlüsselungsverfahrens darf nur von der Geheimhaltung des Schlüssels abhängen, nicht jedoch von der Geheimhaltung des Algorithmus.

Nach Ertel [Ertel, 2012, S. 23] gibt es einige Beispiele für kryptografische Algorithmen,

die dieses Prinzip nicht beachtet haben und anschließend gebrochen wurden. Beispielsweise beruhte der Verschlüsselungsalgorithmus für GSM-Handy-Telefonate auf Geheimhaltung, er wurde jedoch 1999 geknackt.

Dies verdeutlicht nochmal die Notwendigkeit einer sehr großen Schlüsselmenge, denn wenn das Kerckhoffs'sche Prinzip angewendet wird, nutzt einem Angreifer das Wissen über den verwendeten Algorithmus noch nicht viel. Doch an dieser Stelle sei bereits angemerkt, dass ein großer Schlüsselraum alleine, ohne Beachtung von Shannons Definitionen, nicht unbedingt zu hoher Sicherheit führen muss. Im Folgenden werden verschiedene Angriffe vorgestellt.

2.3 Angriffe auf Kryptosysteme

Die einfachste Form eines Angriffs ist die sogenannte *vollständige Suche* (auch als *Brute-Force-Methode* bezeichnet). Hier versucht man, den zu einem Geheimtext gehörigen Klartext zu berechnen, indem nacheinander alle möglichen Schlüssel ausprobiert werden. Die vollständige Suche ist der einfachste Cipher-Text-Only-Angriff. Durch eine sehr große Mächtigkeit von $\mathbb{K}$, kann man sich im Allgemeinen vor diesem Angriff schützen. In Abhängigkeit von den Möglichkeiten des Angreifers unterscheidet man in der Literatur verschiedene Angriffe, die wichtigsten sollen im Folgenden vorgestellt werden [Karpfinger und Kiechle, 2009, S. 7]:

- *Cipher-Text-Only:* Der Angreifer kennt nur den verschlüsselten Text c
- *Known-Plain-Text:* Der Angreifer kennt einen Klartext p und den dazugehörigen verschlüsselten Text c
- *Chosen-Plain-Text:* Der Angreifer kann einen Klartext p auswählen und den dazugehörigen Geheimtext c erzeugen (lassen)
- *Adaptive-Chosen-Plain-Text:* Wie beim Chosen-Plain-Text-Angriff, aber mit mehreren Runden, in denen der Angreifer seine Wahl anpassen kann
- *Chosen-Cipher-Text:* Der Angreifer kann einen selbst bestimmten Geheimtext c entschlüsseln lassen

Kryptosysteme sollten im besten Fall gegen jeden von diesen Angriffen geschützt sein. Beispiele für konkrete Angriffe folgen in den nächsten beiden Kapiteln.

2.4 Blockchiffren und Stromchiffren

Verschlüsselungsverfahren, in denen ein kontinuierlicher Klartext-Strom beliebiger Länge durch einen bestimmten Schlüsselstrom verschlüsselt wird, bezeichnet man als Stromchiffren. Ein bekanntes Beispiel für eine Stromchiffre ist das One-Time-Pad [Schmeh, 2013, S. 281–282].

Blockchiffren hingegen sind Verfahren, in denen einzelne Klartextblöcke m_i, fester Länge, zu Geheimtextblöcken n_i, ebenfalls fester Länge, verschlüsselt werden. Eine gängige

Blockgröße ist 128 Bit. Falls der Klartext größer als die Blocklänge ist, muss dieser auf verschiedene Blöcke verteilt werden. Wenn die Blocklänge kein Teiler der Klartextgröße ist, muss der letzte Block zudem durch ein bestimmtes Verfahren aufgefüllt werden (Padding). Ein einfaches Paddingverfahren besteht darin, die fehlenden Bits durch Nullen zu ersetzen [Schwenk, 2014, S. 10].

2.5 Betriebsmodi von Blockchiffren

Wenn zwei Teilnehmer mehr als einen Block austauschen möchten, gibt es verschiedene Möglichkeiten wie dies geschehen kann. Es ist möglich, denselben Klartextblock jedes mal auf denselben Geheimtextblock abzubilden. Es ist jedoch auch vorstellbar, dass nachfolgende Geheimtextblöcke von den Vorgängern abhängen. Im Folgenden werden die wichtigsten Betriebsmodi vorgestellt, wobei sich die Sicherheit der verschiedenen Modi jedoch unterscheidet. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) empfiehlt beispielsweise in einer Technische Richtlinie von 2015 die Benutzung von GCM, CBC und CTR [Bundesamt für Sicherheit in der Informationstechnik, 2015, S. 23].

2.5.1 Electronic Codebook Mode (ECB)

Der Electronic Codebook Mode ist ein sehr simpler Betriebsmodus. Nachdem ein Klartext p auf verschiedene Blöcke aufgeteilt wurde, wird jeder Block unabhängig mit einer Blockchiffre verschlüsselt. Formal werden die Geheimtextblöcke n_i wie folgt berechnet:

$$n_i = f(m_i, k)$$

Dies führt jedoch dazu, dass gleiche Klartextblöcke jedes mal zu demselben Geheimtextblock verschlüsselt werden. Diese Regelmäßigkeiten bieten einem potentiellen Angreifer Möglichkeiten zur Kryptoanalyse. Ein anderes Problem ist, dass ein Angreifer die Reihenfolge der Geheimtextblöcke vertauschen kann, ohne dass der Empfänger dies bemerkt. Für längere Klartexte ist der Electronic Codebook Mode nicht zu empfehlen [Buchmann, 2010, S. 69–71].

2.5.2 Cipherblock Chaining Mode (CBC)

Der Cipherblock Chaining Mode behebt das Problem, dass gleiche Klartextblöcke immer zu demselben Geheimtextblock verschlüsselt werden. Dies geschieht, indem ein Klartextblock vor dem Anwenden der Blockchiffre mit dem vorherigen Geheimtextblock XOR-verknüpft wird. Damit auch der erste Klartextblock XOR-verknüpft werden kann, führt man einen Initialisierungsvektor (IV) ein, der zur Berechnung des ersten Geheimtextblocks benutzt wird. Dieser Vektor muss vor dem Beginn der Verschlüsselung ausgetauscht werden und braucht nicht zwingend geheim gehalten werden. Für mehrere Übertragungen sollte jedoch jedes Mal ein neuer Vektor ausgetauscht werden. Formal werden die Geheimtextblöcke wie folgt ausgerechnet [Schmeh, 2013, S. 369–370]:

$$n_i = f(m_i \oplus n_{i-1}, k)$$

Da Fehler in der Übertragung sich auf die Berechnung der weiteren Geheimtextblöcke auswirken, ist dieses Verfahren nicht für unzuverlässige Transportprotokolle wie beispielsweise UDP geeignet.

2.5.3 Cipher Feedback Mode (CFB)

Ähnlich wie beim CBC-Mode hängt beim Cipher Feedback Mode die Verschlüsselung nachfolgender Blöcke von dem Ergebnis vorheriger Verschlüsselungen ab. Konkret wird ein Geheimtextblock erzeugt, indem der vorherige Geheimtextblock mit der Blockchiffre verschlüsselt wird und das Ergebnis mit dem aktuellen Klartextblock verschlüsselt wird. Für den ersten Geheimtextblock wird wieder ein Initialisierungsvektor benötigt. Mathematisch wird das Verfahren wie folgt beschrieben [Schmeh, 2013, S. 371–372]:

$$n_i = f(n_{i-1}, k) \oplus m_i$$

2.5.4 Output Feedback Mode (OFB)

Beim Output Feedback Mode wird die Blockchiffre als Stromchiffre verwendet. Die Idee ist, dass die Geheimtextblöcke an sich, nicht durch die Blockchiffre berechnet werden, sondern durch eine XOR-Verknüpfung mit einem bestimmten Block s_i. Der Block s_0 heißt Initialisierungsvektor und wird vorher ausgetauscht. Die nachfolgenden Blöcke s_i entstehen aus der Anwendung der Blockchiffre auf den Block s_{i-1}. Mathematisch werden die Berechnungen wie folgt definiert [Schmeh, 2013, S. 370]:

$$\begin{aligned} s_i &= f(s_{i-1}, k) \\ n_i &= s_i \oplus m_i \end{aligned}$$

Der OFB-Modus hat gegenüber anderen Modi folgende Vorteile [Schmeh, 2013, S. 371]:

- OFB ist sehr schnell, wenn der Sender die Blöcke s_i im Voraus berechnet. Die Verschlüsselung der Klartextblöcke besteht dann nur noch aus der Anwendung der XOR-Verknüpfung.
- Es können kurze Blöcke gesendet werden, ohne dass diese aufgefüllt werden. Im Falle von kurzen Nachrichten, kann die zu übertragende Datenmenge erheblich gesenkt werden.

2.5.5 Counter Mode (CTR)

Beim Counter Mode wird zunächst eine Zufallszahl z_0 gebildet, welche Zähler genannt wird, und nach jeder Blockverschlüsselung inkrementiert wird. Um einen Geheimtextblock zu generieren wird die Blockchiffre auf den Zähler angewandt und mit dem zugehörigen Klartextblock XOR-verknüpft. Mathematisch werden folgende Berechnungen benutzt [Schmeh, 2013, S. 373]:

$$z_i = z_{i-1} + 1 \tag{2.1}$$
$$n_i = m_i \oplus f(z_i, k) \tag{2.2}$$

2.6 Hashfunktionen

Unter einer Hashfunktion verstehen wir eine Abbildung

$$H\colon \{0,1\}^* \to \{0,1\}^n, n \in \mathrm{N}.$$

Es werden also Bitstrings beliebiger Länge auf einen Bitstring fester Länge abgebildet. Hashfunktionen spielen beispielsweise im Zusammenhang mit Signaturverfahren und bei der Speicherung von Passwörtern eine Rolle. Sei m eine Nachricht, dann ist $H(m)$ der Hashwert von m.

Eine Hashfunktion, die im Zusammenhang mit kryptografischen Verfahren eingesetzt werden soll, muss je nach Anwendung verschiedene Bedingungen erfüllen:

- Einweg-Eigenschaft: Für gegebenes $h \in \{0,1\}^n$ ist es praktisch unmöglich, einen Wert $m \in \{0,1\}^*$ mit $H(m) = h$ zu finden
- 2nd-Preimage-Eigenschaft: Für gegebenes $m \in \{0,1\}^*$ ist es praktisch unmöglich, einen Wert $m' \in \{0,1\}^* \backslash \{\mathrm{m}\}$ mit $H(m) = H(m')$ zu finden
- Kollisionsresistenz: Es ist praktisch unmöglich, zwei Werte $m, m' \in \{0,1\}^*$ so zu finden, dass $m \neq m'$ und $H(m) = H(m')$ gilt

Eine Hashfunktion die alle Kriterien erfüllt, wird als kryptografisch stark bezeichnet. Einige Beispiele für kryptografisch starke Hashfunktionen sind SHA-256, SHA-384 sowie SHA-512 [Bundesamt für Sicherheit in der Informationstechnik, 2015, S. 38]. Eine ausführlichere Behandlung von Hashfunktionen sowie einer Beschreibung der SHA-Familie findet sich in *Moderne Kryptographie* [Küsters und Wilke, 2011, S. 193–204].

3 Symmetrische Verschlüsselung

In diesem Kapitel wird zunächst anhand von einfachen, historischen[1] und symmetrischen Kryptoalgorithmen das Prinzip der symmetrischen Verschlüsselung veranschaulicht und mit einigen Angriffen grundlegende Methoden der Kryptoanalyse vorgestellt. Am Ende des Kapitels wird der aktuelle Standardalgorithmus AES vorgestellt. Für alle Algorithmen in diesem Kapitel gilt $k = k'$, es wird also zum ver- und entschlüsseln jeweils derselbe Schlüssel benutzt.

3.1 Monoalphabetische Substitutionschiffren

Algorithmen, welche die sogenannte monoalphabetische Substitution verwenden, sind dadurch gekennzeichnet, dass alle Buchstaben aus dem Klartext jedes Mal durch denselben Geheimtext Buchstaben substituiert werden, beispielsweise könnte jedes a durch ein e ersetzt werden. Im Folgenden werden zwei Verfahren vorgestellt.

3.1.1 Caesar-Verschlüsselung

Die Caesar-Verschlüsselung ist ein sehr altes Verfahren und wurde nach dem einstigen römischen Feldherrn Gaius Julius Caesar benannt, welcher dieses Verfahren angeblich für seine militärische Kommunikation benutzt hat. Die folgende Beschreibung des Verfahrens orientiert sich an Karpfinger und Kiechle [Karpfinger und Kiechle, 2009]. Zur Vereinfachung definieren wir das Verfahren nur auf dem Alphabet

$$\Sigma = \{a, b, c, ..., z\} \text{ mit } |\Sigma| = 26.$$

Die Mengen $\mathbb{P}$ und $\mathbb{C}$ können also durch Σ^* beschrieben werden. Wörter aus den Mengen $\mathbb{P}$ und $\mathbb{C}$ werden also aus beliebig vielen Zeichen aus Σ zusammengesetzt. Für eine reale Implementierung auf heutigen Computern kann stattdessen beispielsweise das ASCII-Alphabet verwenden werden. Weiter sei $\mathbb{K} = \Sigma$, sodass jeder Buchstabe einen Schlüssel darstellt. Die Funktionsweise der Caesar-Verschlüsselung wird durch den Algorithmus 1 beschrieben. Die Funktion length() liefert dabei die Länge eines Strings, die Funktion posInAlphabet(x) die Position des Zeichen x im Alphabet (hier gilt $\texttt{posInAlphabet}(a) = 0$ und $\texttt{posInAlphabet}(z) = 25$) und $\texttt{toChar}(y)$ das Zeichen welches im Alphabet an der Stelle y steht (hier gilt $\texttt{toChar}(0) = a$).

Beispiel 3.1.1. *Der Klartext „ichschreibemeinebachelorarbeit“ wird mit dem Schlüssel „g“ zu dem Geheimtext „oinyinxkohkskotkhginkruxgxhkoz“.*

[1] Eine Vielzahl von weiteren historischen Verfahren wird zum Beispiel in [Singh, 2004] beschrieben.

Algorithmus 1 : Caesar-Verschlüsselung

Eingabe : $p \in \mathbb{P}$, $k \in \mathbb{K}$
Ausgabe : $c \in \mathbb{C}$

```
1  c := "";
2  for i := 0; i < p.length(); i++ do
3  |  newValue := (posInAlphabet(p[i]) + posInAlphabet(k)) mod |Σ|;
4  |  c := c + toChar(newValue);
5  end
6  return c;
```

Die Entschlüsselung wird durch Algorithmus 2 beschrieben. Bei der Entschlüsselung muss die Verschlüsselung rückgängig gemacht werden, es muss also auf dem Geheimtext die inverse Operation durchgeführt werden: wenn ein Buchstabe beispielsweise um 10 Zeichen nach vorne rotiert wurde, muss er nun um 10 Zeichen nach hinten rotiert werden.

Algorithmus 2 : Caesar-Entschlüsselung

Eingabe : $c \in \mathbb{C}$, $k \in \mathbb{K}$
Ausgabe : $p \in \mathbb{P}$

```
1  p := "";
2  for i := 0; i < c.length(); i++ do
3  |  newValue := (posInAlphabet(c[i]) - posInAlphabet(k)) mod |Σ|;
4  |  p := p + toChar(newValue);
5  end
6  return p;
```

Die Caesar-Verschlüsselung ist jedoch kein sicheres Verfahren, da aufgrund der sehr kleinen Schlüsselmenge selbst ein Cipher-Text-Only-Angriff sehr einfach durch das Probieren aller möglichen Schlüssel durchgeführt werden kann. Im Falle eines Known-Plain-Text-Angriffs kann sogar direkt durch das Vergleichen der ersten beiden Buchstaben von Klartext und Geheimtext der Schlüssel rekonstruiert werden.

3.1.2 Substitutions-Chiffren

Substitutions-Chiffren können als eine Erweiterung der Caesar-Verschlüsselung aufgefasst werden [Karpfinger und Kiechle, 2009, S.5], wobei statt den 26 möglichen Verschiebungen der Caesar-Verschlüsselung alle möglichen Permutationen des Alphabets als Schlüssel zugelassen sind. Als Alphabet verwenden wir wieder Σ, für $\mathbb{P}$ und $\mathbb{C}$ ändert sich also nichts. Ein Beispiel, wie ein Schlüssel in diesem Verfahren aussehen kann, ist in Abbildung 3.1 aufgeführt.

Die Funktionsweise der Verschlüsselung ist durch Algorithmus 3 beschrieben, die Funktion getPermutatedLetter(p,k) liefert dabei den Buchstaben zu dem der Buchstabe p unter der Permutation k substituiert wird (Sei k die Permutation aus Abbildung 3.1, dann

liefert getPermutatedLetter(f,k) den Buchstaben z).

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
l a s x h z i b u m e o w r v c g d p n q f j y t k
```

Abbildung 3.1: Beispiel für eine Permutation des lateinischen Alphabets

Algorithmus 3 : Verschlüsselung mit Substitutions-Chiffren

Eingabe : $p \in \mathbb{P}$, $k \in \mathbb{K}$
Ausgabe : $c \in \mathbb{C}$
1 $c :=$ "";
2 **for** $i := 0$; $i < p.\texttt{length}()$; $i++$ **do**
3 $\quad newLetter :=$ `getPermutatedLetter`$(p[i], k)$;
4 $\quad c := c + newLetter$;
5 **end**
6 **return** c;

Beispiel 3.1.2. *Der Klartext „ichschreibemeinebachelorarbeit“ wird mit dem Schlüssel aus Abbildung 3.1 zu dem Geheimtext „usbpsbdhuahwhurhalsbhovdldahun“.*

Die Entschlüsselung ist durch Algorithmus 4 beschrieben, dieser kehrt die Verschlüsselung wieder um. Die Synergie zwischen Ver- und Entschlüsselung ist dabei dieselbe, wie bei der Caesar-Verschlüsselung – die Substitution der Buchstaben muss rückgängig gemacht werden. Die Funktion getInversePermutatedLetter(g,k) funktioniert umgekehrt zu getPermutatedLetter(f,k) (sei k wieder die Permutation aus Abbildung 3.1, dann liefert getInversePermutatedLetter(b,k) den Buchstaben h).

Algorithmus 4 : Entschlüsselung mit Substitutions-Chiffren

Eingabe : $c \in \mathbb{C}$, $k \in \mathbb{K}$
Ausgabe : $p \in \mathbb{P}$
1 $p :=$ "";
2 **for** $i := 0$; $i < c.\texttt{length}()$; $i++$ **do**
3 $\quad newLetter :=$ `getInversePermutatedLetter`$(p[i], k)$;
4 $\quad p := p + newLetter$;
5 **end**
6 **return** p;

Durch die Ausweitung des Schlüsselraums auf alle möglichen Permutationen erhöht sich in unserem Fall die Anzahl der möglichen Schlüssel auf 26! = 403 291 461 126 605 635 584 000 000. Dies ist genug, damit eine vollständige Suche *praktisch* nicht durchgeführt werden kann.

Das Problem bei diesem Verfahren ist jedoch, dass einige statische Eigenschaften wie die Buchstabenhäufigkeit erhalten bleiben. In Abbildung 3.2 ist eine exemplarische Buchstabenhäufigkeit des lateinischen Alphabets aufgeführt [Freiermuth et al., 2014, S. 77], welche man durch die Analyse eines entsprechend langen deutschen Textes bekommt. In diesem Diagramm sieht man deutliche Unterschiede in dem Aufkommen bestimmter Buchstaben. Der Buchstabe E kommt beispielsweise mit Abstand am häufigsten vor, während Y,X und Q kaum vorkommen. Ein Cipher-Text-Only-Angriff kann diese Eigenschaft ausnutzen, indem er für den Geheimtext ebenfalls eine Häufigkeitsanalyse erstellt. Diese vergleicht er mit der deutschen Häufigkeitstabelle (sofern der Angreiffer weiß, dass der Klartext deutsch ist) und kann so den Klartext rekonstruieren. Für einen optimierten Angriff kann der Angreifer zusätzlich die Bigrammverteilung und die Trigrammverteilung der Texte vergleichen. Ist der Angriff gelungen, kann der Angreifer den Schlüssel rekonstruieren und weitere Geheimtexte derselben Übertragung entschlüsseln.

Dieser Algorithmus zeigt nochmal, dass obwohl der Schlüsselraum extrem groß ist, ein großer Schlüsselraum allein kein ausreichendes Kriterium für sichere Kryptoalgorithmen darstellt, sondern lediglich die vollständige Suche verhindert.

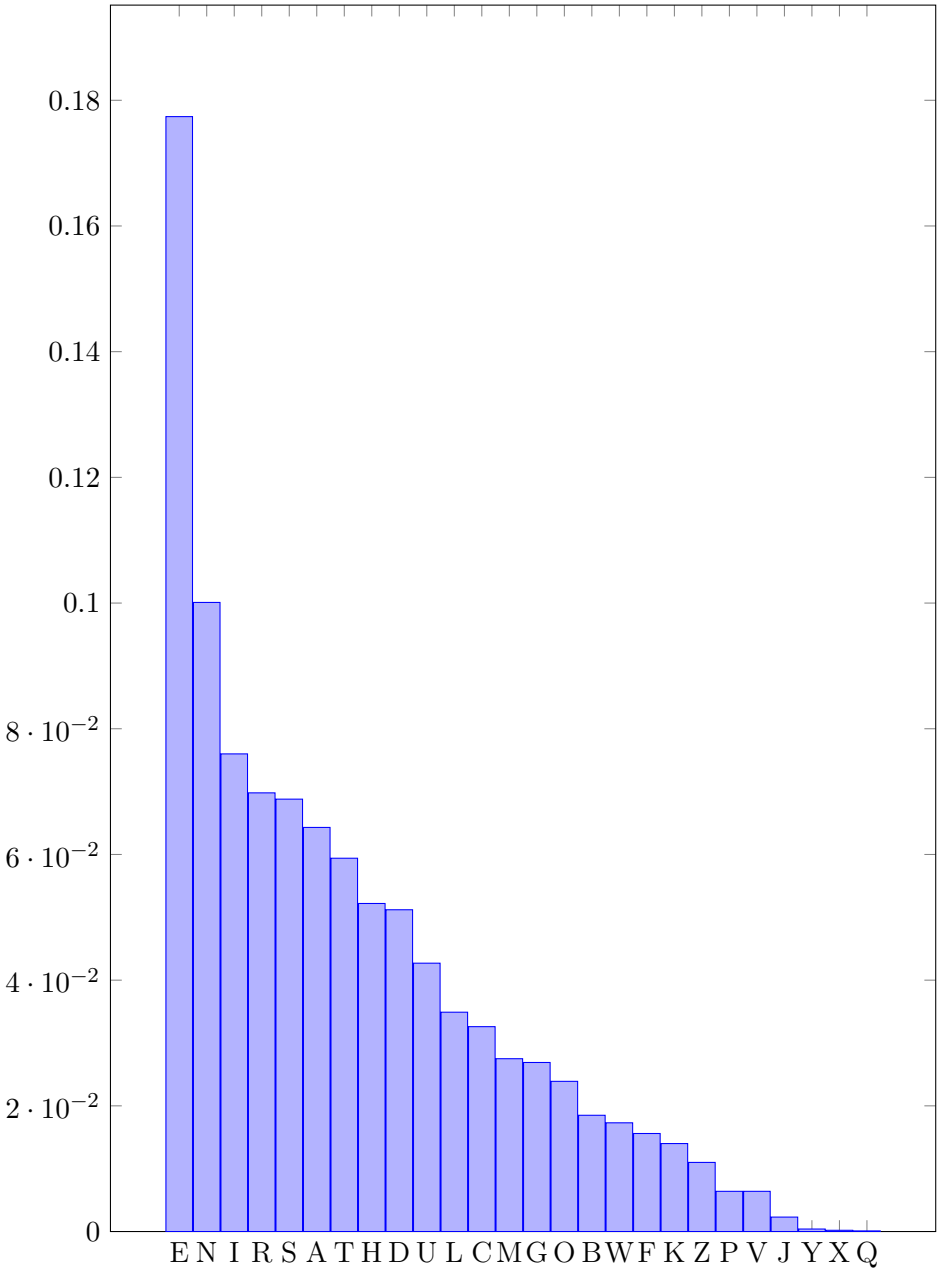

Abbildung 3.2: Erwartete Häufigkeitsverteilung der Buchstaben in längeren deutschen Texten, die Umlaute Ä, Ö, Ü wurden wie AE, OE, UE behandelt und ß wurde durch SS ersetzt [Freiermuth et al., 2014, S. 77]

3.2 Polyalphabetische Substitutionschiffren

Die beiden zuvor vorgestellten Verfahren waren monoalphabetisch, was bedeutete, dass beispielsweise der Klartextbuchstabe E jedes Mal zu einem G verschlüsselt wird. Verfahren in denen dies nicht gilt, also ein E das erste Mal zu einem A und das zweite Mal zu einem Y verschlüsselt werden kann, heißen polyalphabetisch.

3.2.1 Vigenère-Verschlüsselung

Eines der bekanntesten polyalphabetischen Verschlüsselungsverfahren ist die Vigenère-Chiffre von Blaise de Vigenère aus dem 16. Jahrhundert [Schmeh, 2013]. Die grundlegende Funktionsweise soll durch folgendes Beispiel mit dem Klartext „ichschreibemeinebachelorarbeit“ und dem Schlüssel „abc“ illustriert werden:

```
ichschreibemeinebachelorarbeit
abcabcabcabcabcabcabcabcabcabc
idjsdjrfkbfoejpeccciglptasdejv
```

Abbildung 3.3: Beispiel für eine Vigenère-Verschlüsselung

Der Schlüssel wird periodisch unter den Klartext geschrieben, um anschließend Buchstabe für Buchstabe eine Caesar-Verschlüsselung anzuwenden. Im Gegensatz zur reinen Caesar-Verschlüsselung ist dieses Verfahren jedoch polyalphabetisch (sofern der Schlüssel aus mehr als einem Buchstaben besteht), da die Caesar-Verschlüsselung in diesem Fall, je nach Position des Buchstaben, einen anderen Schlüssel benutzt. In Algorithmus 5 ist die genaue Funktionsweise der Verschlüsselung beschrieben, während in Algorithmus 6 die Entschlüsselung beschrieben ist.

Algorithmus 5 : Vigenère-Verschlüsselung

Eingabe : $p \in \mathbb{P}$, $k \in \mathbb{K}$
Ausgabe : $c \in \mathbb{C}$
1 $c := ""$;
2 **for** $i := 0$; $i < p.\texttt{length}()$; $i++$ **do**
3 $\quad key := k[i \ mod \ k.\texttt{length}()]$;
4 $\quad newValue := (\texttt{posInAlphabet}(p[i]) + \texttt{posInAlphabet}(key)) \ mod \ |\Sigma|$;
5 $\quad c := c + \texttt{toChar}(newValue)$;
6 **end**
7 **return** c;

Der im vorherigen Abschnitt vorgestellte Angriff, welcher die Buchstabenhäufigkeiten von Geheim- und Klartext vergleicht, funktionierte nur, weil das eingesetzte Verfahren monoalphabetisch ist. Allerdings ist ein anderer Angriff bekannt, mit dem man die Vigenère-Verschlüsselung dennoch brechen kann [Karpfinger und Kiechle, 2009, S. 13–18]. Die Grundidee des Angriffs ist es zunächst die Schlüssellänge l zu berechnen, da dann das Verfahren auf das Brechen einer Caesar-Verschlüsselung reduziert werden kann.

Algorithmus 6 : Vigenère-Entschlüsselung

Eingabe : $c \in \mathbb{C}$, $k \in \mathbb{K}$
Ausgabe : $p \in \mathbb{P}$

```
1 p := "";
2 for i := 0; i < c.length(); i++ do
3 |   key := k[i mod k.length()];
4 |   newValue := (posInAlphabet(c[i]) − posInAlphabet(key))mod|Σ|;
5 |   p := p + toChar(newValue);
6 end
7 return p;
```

Um eine Menge von möglichen Schlüssellängen zu berechnen, kann man den Kasiski-Test verwenden, vorausgesetzt die Länge des Geheimtext ist deutlich länger als der Schlüssel. Die Idee hierbei ist, dass Buchstabenfolgen des Klartexts auf dieselbe Buchstabenfolge im Geheimtext abgebildet werden, wenn ihr Abstand ein Vielfaches der Schlüssellänge ist. Man sucht also im Geheimtext sich wiederholende Buchstabenfolgen und insbesondere deren Abstände d_1,...,d_n. Die Schlüssellänge l ist ein Teiler von diesen Abständen. Um die möglichen Teiler noch weiter einzuschränken, berechnet man nun $t = \text{ggt}(d_1,...,d_n)$. Als mögliche Schlüssellänge l kommen nun alle Teiler von t infrage. Um Abstände d_j, welche durch zufällige Folgen erzeugt wurden zu erkennen, entfernt man alle Abstände d_j, die zu den meisten Abständen $d_i \neq d_j$ teilerfremd sind.

Um eine Abschätzung der Größenordnung der Schlüssellänge zu bekommen, kann man den Friedman-Test, welcher beispielsweise in *Kryptologie Algebraische Methoden und Algorithmen* von Karpfinger und Kiechle [Karpfinger und Kiechle, 2009, S. 14–18] erklärt wird, durchführen. Durch die Kombination beider Tests erhält man in den meisten Fällen die tatsächliche Schlüssellänge l.

3.2.2 One-Time-Pad

Das One-Time-Pad wurde 1917 von Major J. Mauborgne und G. Vernam von AT&T erfunden [Kahn, 1967] und gilt, sofern einige Voraussetzungen beachtet werden, als ein Kryptosystem mit perfekter Sicherheit. Das One-Time-Pad kann man beispielsweise über dem binären Alphabet $\Sigma = \{0,1\}$ definieren. Klartexte, Geheimtexte und Schlüssel sind also Strings die aus Nullen und Einsen bestehen.

Die Funktionsweise des One-Time-Pads wird in Algorithmus 7 dargestellt. Die Operation p[i] $\oplus$ k[i] entspricht der XOR-Verknüpfung von zwei Bits. Bemerkenswerterweise wird für die Ver- und Entschlüsselung exakt derselbe Algorithmus benutzt, da ein Bit das zweimal mit demselben Schlüsselbit XOR-verknüpft wurde, seinen ursprünglichen Wert erhält.

Um perfekte Sicherheit zu erreichen, was äquivalent dazu ist, dass ein Angreifer aus einem Geheimtext keine Informationen über den zugehörigen Klartext extrahieren kann, müssen bei der Verschlüsselung mit dem One-Time-Pad folgende Voraussetzungen eingehalten werden [Karpfinger und Kiechle, 2009, S. 24]:

- Der Schlüssel k muss mindestens solang sein wie der Klartext p
- Der Schlüssel k wurde komplett zufällig erzeugt
- Jeder Schlüssel k wird nur einmal benutzt

Algorithmus 7 : One-Time-Pad

Eingabe : $p \in \mathbb{P}$, $k \in \mathbb{K}$
Ausgabe : $c \in \mathbb{C}$

```
1  c := "";
2  for i := 0; i < p.length(); i++ do
3  |   newValue := p[i] ⊕ k[i];
4  |   c := c + newValue;
5  end
6  return c;
```

Obwohl die Benutzung des One-Time-Pad vom Aspekt der Sicherheit ausgehend sehr sinnvoll wäre, machen die hohen Anforderungen an den Schlüssel das Verfahren für die meisten Anwendungen unbrauchbar, da im Vergleich zu anderen Algorithmen der Aufwand sehr hoch ist. Für Anwendungen in denen jedoch höchstmögliche Sicherheit gefordert ist, kann es dennoch Verwendung finden, beispielsweise wurde es früher im Geheimdienst- und im Militärbereich benutzt [Schmeh, 2013, S. 52]. Einige der Probleme sind folgende [Karpfinger und Kiechle, 2009, S. 25]:

- Das Verfahren verlangt *echte* Zufallszahlen, die vom Computer generierten Zahlen sind in der Regel Pseudozufallszahlen
- Dadurch, dass jeder Schlüssel so lang wie eine Nachricht ist, wird der Datenaustausch verdoppelt
- Das häufige Erzeugen von Schlüsseln hat einen hohen Zeitbedarf, da jeder Schlüssel zunächst erzeugt werden muss und dann noch übermittelt werden muss

3.3 Advanced Encryption Standard (AES)

AES (Advanced Encryption Standard) ist der Nachfolger von DES (Data Encryption Standard) und der aktuelle Standardalgorithmus für symmetrische Verschlüsselung. Die größte Schwäche von DES ist die kurze Schlüssellänge von 56-bit, was dazu führte, dass bereits 1999 ein Angriff mit vollständiger Suche erfolgreich durchgeführt werden konnte [Beutelspacher et al., 2010, S. 81]. DES sollte deshalb heute nicht mehr verwendet werden. Sowohl DES als auch AES sind Block-Chiffren. Der Algorithmus hinter AES wurde durch einen öffentlichen Wettbewerb in den Jahren 1997 bis 2001 ermittelt. Der Sieger des Wettbwerbs war der Algorithmus Rijndael, welcher von Joan Daemen und Vincent Rijmen entwickelt wurde, und seitdem als AES bezeichnet wird [Ertel, 2012, S. 69]. Die

folgende Beschreibung des Verfahrens orientiert sich an der von Schmeh [Schmeh, 2013], wobei im Folgenden nur die Verschlüsselung beschrieben wird. Die Entschlüsselung besteht aus der umgekehrten Anwendung der inversen Operationen.

3.3.1 Einführung

Der AES-Algorithmus ist eine iterierte Blockchiffre, welche aus r Runden besteht. Die Blocklänge b und die Schlüssellänge l können unabhängig die Bitwerte 128, 192 und 256 annehmen, wobei die Standardblocklänge 128-Bit ist. Ein 128-Bit Klartextblock m = (a_0,a_1,a_2,a_3,b_0,b_1,b_2,b_3,c_0,c_1,c_2,c_3,d_0,d_1,d_2,d_3) wird durch eine 4x4 Matrix aus Byte-Variablen wie in Abbildung 3.4 beschrieben.

$$\begin{pmatrix} a_0 & b_0 & c_0 & d_0 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{pmatrix}$$

Abbildung 3.4: Matrixdarstellung eines 128-Bit-Blocks durch Byte-Variablen im AES-Algorithmus

r	b=128	b=192	b=256
l=128	10	12	14
l=192	12	12	14
l=256	14	14	14

Abbildung 3.5: Anzahl der AES-Runden in Abhängigkeit von Block- und Schlüssellänge

Zu Beginn werden in Abhängigkeit des Schlüssels k, infolge einer Schlüsselexpansion r+1 Rundenschlüssel k_0,...,k_r der Länge b erzeugt. Der Wert von r ist abhängig von b und l. Die spezifischen Werte für r sind in Abbildung 3.5 angegeben.

Nachdem die Schlüsselexpansion erfolgt ist, beginnt das eigentliche Verfahren. Zunächst wird einmal AddRoundKey ausgeführt. AddRoundKey ist eine XOR-Verknüpfung des aktuellen Zustands S_i mit dem aktuellen Rundenschlüssel k_i. Die Zustände S_i enthalten den aktuellen Cipherblock und entsprechen der Matrixdarstellung aus Abbildung 3.4. Der Zustand S_0 entspricht dabei der XOR-Verknüpfung von m mit k_0 und S_r entspricht dem engültigen Cipherblock. Nachdem S_0 gebildet wurde, werden r Runden durchgeführt die aus folgenden Schritten bestehen: SubBytes, ShiftRow, MixColumn und AddRoundKey. In der letzten Runde fällt MixColumn weg. Algorithmus 8 fasst den Ablauf der Verschlüsselung noch einmal zusammmen, die einzelnen Funktionen werden in den folgenden Abschnitten genauer beschrieben. Die Eingabe p entspricht genau einem Klartextblock der Länge b. Die Entschlüsselung wird, wie bereits erwähnt, durch die Anwendung der inversen Operationen in umgekehrter Reihenfolge erreicht, auf die hier aber nicht näher eingegangen wird. Für eine genauere Betrachtung des Algorithmus, inklusive der Entschlüsselung, kann beispielsweise das von den Erfindern herausgegebene Buch

The Design of Rijndael. AES: The Advanced Encryption Standard verwendet werden [Daemen und Rijmen, 2002].

Algorithmus 8 : AES-Verschlüsselung

Eingabe : $p \in \mathbb{P}$, $k \in \mathbb{K}$
Ausgabe : $c \in \mathbb{C}$
1 $r :=$ `calcR`$(p.$`length`$(), k.$`length`$())$;
2 $K_{array} :=$ `calcRoundKeys`(k) ; `/* Schlüsselexpansion */`
3 $S := p \oplus K_{array}[0]$;
4 **for** $i := 1$; $i \leq r + 1$; $i++$ **do**
5 $S :=$ `SubByte`(S);
6 $S :=$ `ShiftRow`(S);
7 **if** $i \neq r + 1$ **then**
8 $S :=$ `MixColumn`(S);
9 **end**
10 $S := S \oplus K_{array}[i]$;
11 **end**
12 **return** c;

3.3.2 Schlüsselexpansion

Die Schlüsselexpansion expandiert den Schlüssel k auf eine Länge von $(r+1) \cdot b$ Bit. Die Erzeugung der Schlüsselbits erfolgt iterativ, indem aus bereits bekannten Schlüsselbits neue Schlüsselbits erzeugt werden. Im Folgenden nehmen wir für die Längen l und b die Bitwerte 128 an. Die Expansion geschieht Wortweise, wobei die ersten vier Wörter nicht verändert werden. Zu Beginn der Expansion haben wir den Schlüssel k = (W_0,W_1,W_2,W_3). Jedes Wort W_i besteht aus 32 Bit. Das Wort W_5 wird durch eine XOR-Verknüpfung von W_0 und W_4, das Wort W_6 durch eine XOR-Verknüpfung von W_1 und W_5 gebildet. Die nachfolgenden Wörter werden nach demselben Schema erzeugt.

Eine Ausnahme bilden die Wörter W_i mit i = 4, 8, 12, 16... . Bei der Erzeugung dieser Wörter werden die Wörter W_{i-1} vor der XOR-Verknüpfung wie folgt modifiziert: zunächst wird W_{i-1} um 4 Stellen rotiert, anschließend wird die nichtlineare Abbildung SubBytes angewendet und eine Rundenkonstante addiert.

Die einzelnen Runden-Schlüssel K_{array}[0] bis K_{array}[r] werden nun aus je vier dieser Wörter zusammengesetzt: Der Schlüssel K_{array}[i] entspricht dem Tupel ($W_{(i\cdot4)}$,$W_{(i\cdot4)+1}$,$W_{(i\cdot4)+2}$,$W_{(i\cdot4)+3}$), der Schlüssel K_{array}[1] entspricht also beispielsweise dem Tupel (W_4,W_5,W_6,W_7).

3.3.3 SubBytes

SubBytes ist eine Substitution, welche jedes der 16 Bytes des Zustands S durch ein anderes Byte substituiert. Die spezifischen Werte für die Substitution wurden beim Design des Algorithmus festgelegt und sind in Abbildung 3.6 dargestellt. Die Werte in dieser

Tabelle entsprechen der hexadezimalen Schreibweise. Ein Byte mit dem hexadezimalen Wert 30 wird beispielsweise durch den hexadezimalen Wert 04 substituiert.

00-0F	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10-1F	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20-2F	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30-3F	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40-4F	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50-5F	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60-6F	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70-7F	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80-8F	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90-9F	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
A0-AF	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
B0-BF	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
C0-CF	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
D0-DF	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
E0-EF	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
F0-FF	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Abbildung 3.6: AES-SBOX

3.3.4 ShiftRow

ShiftRow ist eine lineare Permutation auf den Zeilen der Matrix des Zustands S. Konkret rotiert die Permutation die Zeile i um $i-1$ Zeichen nach links, die vierte Zeile wird also beispielsweise um 3 Zeichen nach links rotiert. In Abbildung 3.7 ist die Permutation visualisiert, die rechte Matrix entspricht dabei der Permutation der linken Matrix.

$$\begin{pmatrix} a_0 & b_0 & c_0 & d_0 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{pmatrix} \rightarrow \begin{pmatrix} a_0 & b_0 & c_0 & d_0 \\ b_1 & c_1 & d_1 & a_1 \\ c_2 & d_2 & a_2 & b_2 \\ d_3 & a_3 & b_3 & c_3 \end{pmatrix}$$

Abbildung 3.7: ShiftRow auf einem 128-Bit-Block im AES-Verfahren

3.3.5 MixColumn

MixColumn sorgt für eine Vermischung der Spalten, ist jedoch deutlich komplexer als das Zeilen-Gegenstück ShiftRow. Die neuen Spalteneinträge a'_0 bis a'_3 lassen sich durch eine mathematische Formel wie folgt berechnen:

$$
\begin{aligned}
a'_0 &= (2 \bullet a_0) \oplus (3 \bullet a_1) \oplus a_2 \oplus a_3 \\
a'_1 &= a_0 \oplus (2 \bullet a_1) \oplus (3 \bullet a_2) \oplus a_3 \\
a'_2 &= a_0 \oplus a_1 \oplus (2 \bullet a_2) \oplus (3 \bullet a_3) \\
a'_3 &= (3 \bullet a_0) \oplus a_1 \oplus a_2 \oplus (2 \bullet a_3)
\end{aligned}
$$

Die Verknüpfung $\bullet$ ist wie folgt definiert:

$$
2 \bullet b := \begin{cases} \text{Linksshift von b} & , \text{ falls } b < 128 \\ (\text{Linksshift von } b) \oplus 00011011 & , \text{ falls } b >= 128 \end{cases}
$$
$$
3 \bullet b := (2 \bullet b) \oplus b
$$

Die Elemente der anderen Spalten lassen sich äquivalent berechnen, es muss in der Formel lediglich der Buchstabe a durch ein b, c beziehungsweise d ersetzt werden.

3.3.6 Sicherheit von AES

Zunächst wird begründet, warum AES mit 128 Bit als sicher angesehen werden kann, obwohl DES, ein unsicheres Verfahren, nur knapp die Hälfte an Schlüsselbits hat. Dies liegt daran, dass eine Verdoppelung der Schlüsselbits, nicht zu einer Verdoppelung der Anzahl der Schlüssel führt, sondern zu deutlich mehr. Für eine Verdoppelung der Schlüssel reicht die Hinzunahme eines Schlüsselbits. Die Anzahl der DES Schlüssel beträgt ausgeschrieben 72057594037927936. Der AES-Algorithmus hat im Vergleich jedoch ganze 340282366920938463463374607431768211456 verschiedene Schlüssel, die Anzahl der Schlüssel ist somit um den Faktor 2^{72} größer. Ein einfacher Angriff auf AES-128 mit vollständiger Suche hatte 2012 einen Zeitbedarf von ungefähr 3,4 x 10^{34} Jahren, woran sich auch in nächster Zeit nicht viel ändern wird [Arora, 2012].

Nach der bereits erwähnten Richtlinie des BSI, sind zudem keine anderen praktisch durchführbaren Angriffe bekannt, welche eine signifikant niedrigere Laufzeit aufweisen [Bundesamt für Sicherheit in der Informationstechnik, 2015, S. 23].

3.4 Alternativen zu AES

Neben dem Standardalgorithmus AES gibt es noch einige weitere symmetrische Algorithmen, die zum Verschlüsseln benutzt werden können und eine ähnliche Sicherheit wie AES aufweisen. Im Folgenden soll eine kleine Auswahl weiterer symmetrischer Algorithmen vorgestellt werden, ohne zu detailliert auf die einzelnen Algorithmen einzugehen:

- Serpent
- Twofish
- Salsa20

Serpent war unter den Top 3 der Algorithmen im Wettbewerb zur Bestimmung von AES. Der Algorithmus galt sogar als der sicherste von den damaligen Algorithmen. Ähnlich wie bei AES werden mehrere Runden durchgeführt, in diesem Falle 32. Die hohe Sicherheit wird daher mit einer vergleichsweise langsamen Geschwindigkeit bezahlt, was vermutlich der Grund dafür war, dass Serpent nicht zu AES wurde [Schmeh, 2013, S. 143]. Eine ausführliche Beschreibung von Serpent findet sich in *Serpent: A Proposal for the Advanced Encryption Standard* [Anderson et al., 1998].

Twofish war ebenfalls unter den Top 3 und wird zwischen dem eleganteren AES und dem sehr konservativen Serpent eingeordnet. Es sind keine bekannten Sicherheitsprobleme bekannt. Daher bietet sich Twofish als Alternative zu AES an [Schmeh, 2013, S. 145]. Die vollständige Beschreibung des Algorithmus findet sich in *Twofish: A 128-Bit Block Cipher* [Schneier et al., 1998].

Salsa20 wurde 2005 von Daniel J. Bernstein entwickelt und umfasst eine Familie von 256-Bit-Stromchiffren. Im Vergleich zu AES habe es ein deutlich verbessertes Verhältnis zwischen Geschwindigkeit und Sicherheit, so seien 20 Runden im Salsa20-Algorithmus deutlich schneller als 14 AES-Runden. Mehr Informationen zu Salsa20 finden sich in *The Salsa20 family of stream ciphers* [Bernstein, 2005].

3.5 Probleme von symmetrischen Algorithmen

Symmetrische Algorithmen haben in der Praxis hauptsächlich mit zwei Problemen zu kämpfen. Das erste Problem ist der Schlüsselaustausch. Wenn zwei Teilnehmer ihren Datenaustausch verschlüsseln liegt dies im Allgemeinen daran, dass der zugrunde liegende Übertragungskanal unsicher ist und kein sicherer Übertragungskanal verfügbar ist. Dies bedeutet, dass dritte die Daten abfangen und mitlesen können. Bevor die beiden Teilnehmer ihre Daten jedoch verschlüsseln, müssen sie sich über einen Schlüssel einigen. Würden sie den Schlüssel als Klartext übertragen, könnte dieser abgefangen werden und die gesamte Verschlüsselung kann mitgelesen werden. Im Folgenden Kapitel werden einige Methoden zum Schlüsselaustausch vorgestellt.

Das zweite Problem symmetrischer Algorithmen ist die Anzahl der benötigten Schlüssel. Wenn N Teilnehmer paarweise miteinander vertraulich kommunizieren möchten, dann müssen insgesamt $\binom{N}{2} = \frac{N \cdot (N-1)}{2}$ Schlüssel erzeugt und verwaltet werden. Bei asymmetrischen Algorithmen, welche im folgenden Kapitel thematisiert werden, müssen hingegen nur N Schlüssel erzeugt und verwaltet werden [Swoboda et al., 2008, S. 25].

4 Asymmetrische Verschlüsselung

In diesem Kapitel werden zum einen verschiedene asymmetrische Verfahren wie beispielsweise RSA untersucht und zum anderen sollen allgemeine Stärken und Schwächen von asymmetrischen Algorithmen, insbesondere im Vergleich zu symmetrischen Verfahren herausgestellt werden. Weiter wird gezeigt, inwieweit sich das Problem des Schlüsselaustausches von symmetrischen Verfahren durch asymmetrische Verfahren lösen lässt.

Allgemein sind asymmetrische Verschlüsselungsalgorithmen Verfahren, in denen in der Regel zur Verschlüsselung ein anderer Schlüssel als zur Entschlüsselung benutzt wird. Der Schlüssel zur Verschlüsslung wird öffentlicher Schlüssel, der zur Entschlüsselung privater Schlüssel genannt.

4.1 RSA

Der RSA-Algorithmus wurde bereits 1978 von R. Rivest, A. Shamir und L. Adleman entwickelt und ist heute einer der wichtigsten asymmetrischen Algorithmen. Neben der Möglichkeit Nachrichten zu verschlüsseln, ist es auch möglich den RSA-Algorithmus zur Signierung von Nachrichten zu verwenden [Rivest et al., 1978]. Der Name RSA setzt sich aus den Anfangsbuchstaben der drei Erfinder zusammen.

4.1.1 Schlüsselerzeugung

Bevor man den RSA-Algorithmus verwenden kann, ist es notwendig, dass sich jeder Teilnehmer ein Schlüsselpaar wie folgt erzeugt:

- Wähle zwei zufällige große Primzahlen p und q
- Berechne das Modul $n = p \cdot q$ sowie $\varphi(n) = (p\text{-}1) \cdot (q\text{-}1)$
- Wähle eine natürliche, zufällige und zu $\varphi(n)$ teilerfremde Zahl e
- Berechne das zu e inverse Element $d = e^{-1} \bmod \varphi(n)$
- Das Tupel (n,e) ist der öffentliche Schlüssel, (n,d) entspricht dem privaten Schlüssel

4.1.2 Verschlüsselung und Entschlüsselung

Der RSA-Algorithmus wird zur Verschlüsselung von natürlichen Zahlen verwendet, Klartexte und Geheimtexte sind also jeweils ganze Zahlen. Es können jedoch nur Zahlen verschlüsselt werden, die kleiner als n sind. Sollen größere Nachrichten verschlüsselt werden, müssen diese in Blöcke geteilt werden. RSA ist daher eine Blockchiffre.

Die Schlüssel haben die zuvor definierte Form. Wenn ein Sender A einem Empfänger B eine Nachricht p schicken möchte, muss er sich zunächst den öffentlichen Schlüssel $k = (n,e)$ von B besorgen. Dann verschlüsselt er wie in Algorithmus 9 beschrieben seine Nachricht und verschickt sie anschließend. Um die Nachricht c zu entschlüsseln wird der private Schlüssel $k' = (n,d)$ benötigt. Mit diesem kann, wie in Algorithmus 10 beschrieben, die ursprüngliche Nachricht wieder hergestellt werden.

Ohne zu sehr auf die mathematischen Grundlagen einzugehen soll kurz begründet werden, warum nach der Anwendung von Ver- und Entschlüsselung die ursprüngliche Nachricht erhalten bleibt. Dies liegt daran, dass d invers zu e ist:

$$c^d = (p^e)^d = p^{1(\mod \varphi(n))} = p(\mod n)$$

Algorithmus 9 : RSA Verschlüsselung

Eingabe : $p \in \mathbb{P}$, $k \in \mathbb{K}$
Ausgabe : $c \in \mathbb{C}$
1 $c := p^e \mod n$;
2 **return** c;

Algorithmus 10 : RSA Entschlüsselung

Eingabe : $c \in \mathbb{C}$, $k \in \mathbb{K}$
Ausgabe : $p \in \mathbb{P}$
1 $p := c^d \mod n$;
2 **return** p;

4.1.3 Sicherheit von RSA

Die Sicherheit von RSA beruht hauptsächlich auf der Annahme, dass die Berechnung der Primfaktorzerlegung von n ein schwieriges mathematisches Problem ist. Schwierig bedeutet hierbei, dass es keinen effizienten Algorithmus zur Berechnung der Lösung des Problems gibt. Für die Primfaktorzerlegung ist allerdings nicht klar, ob es wirklich ein schwieriges Problem ist, es wurde jedoch bisher kein effizienter Algorithmus gefunden [Swoboda et al., 2008, S. 125]. Falls in der Zukunft ein effizienter Algorithmus zur Primfaktorzerlegung gefunden wird, ist RSA nicht mehr sicher, da jedem Angreifer die Zahl n zur Verfügung steht. Für die Verwendung empfiehlt das BSI eine Schlüssellänge von mindestens 2000 Bit [Bundesamt für Sicherheit in der Informationstechnik, 2015, S. 28].

Im Folgenden werden verschiedene Angriffsmöglichkeiten auf die RSA-Verschlüsselung vorgestellt.

4.1.3.1 Berechnung der Primfaktorzerlegung

Da das Modul n Teil des öffentlichen Schlüssels ist, kann jeder Angreifer versuchen die RSA-Verschlüsselung zu brechen, indem er die Primfaktorzerlegung von n berechnet. Anschließend kann er den privaten Schlüssel ausrechnen.

Eine einfache Methode zur Berechnung der Primfaktorzerlegung besteht darin, alle möglichen Teiler auszuprobieren. Man beginnt bei 3 und prüft, ob der Rest bei der Division von n durch den jeweiligen Teiler 0 ist. Falls der Rest nicht 0 ist, inkrementiert man den Teiler um 2 (die geraden Zahlen brauchen nicht betrachtet werden). Hat man einen Teiler, bei dem der Rest 0 ist, gefunden, hat man die Primfaktorzerlegung berechnet. Dieser sehr simple Algorithmus hat jedoch eine Laufzeit im Bereich von $\mathcal{O}(2^b)$, b ist dabei die Anzahl der Bits die zur Darstellung von n benötigt werden [Swoboda et al., 2008, S. 125].

Neben diesem sehr simplen Verfahren gibt es eine Vielzahl von weiteren Algorithmen zur Berechnung der Primfaktorzerlegung, von denen jedoch keiner eine polynomielle Laufzeit aufweist. Eine Auswahl von weiteren Algorithmen zur Berechnung der Primfaktorzerlegung findet sich in *Kryptologie Algebraische Methoden und Algorithmen* [Karpfinger und Kiechle, 2009].

4.1.3.2 Angriffe auf kurze Nachrichten

Benutzt man RSA in der oben beschriebenen Form für die Verschlüsselung von kurzen Nachrichten, kann unter bestimmten Umständen ein Angreifer diese entschlüsseln ohne den privaten Schlüssel zu kennen.

Hierzu soll folgendes Szenario dienen: Ein Client möchte sich bei einem Bank-Server authentifizieren. Der Client schickt dazu seinen mit RSA verschlüsselten vierstelligen PIN-Code zum Server. Ein Angreifer der den verschlüsselten PIN-Code abfängt und weiß, dass dieser ein vierstelliger PIN-Code ist, kann jetzt alle möglichen PIN- Kombinationen mit dem öffentlichen Schlüssel der Bank verschlüsseln. Dabei vergleicht er den aktuell verschlüsselten PIN-Code mit dem abgefangenen. Sobald der Angreifer eine Übereinstimmung gefunden hat, hat er den richtigen PIN-Code gefunden.

Um diese Art von Angriff zu verhindern, kann man beispielsweise die eigentliche Nachricht vor der Verschlüsselung, durch das Hinzufügen von Zufallsbits vergrößern, um eine vollständige Suche unmöglich zu machen [Forster, 2015, S. 120].

4.1.3.3 Bedrohung durch Quantencomputer

Eine noch in der Zukunft liegende Bedrohung von RSA ist der Bau von leistungsstarken Quantencomputern. Eine Einführung in das Thema Quantencomputing liefert beispielsweise [Homeister, 2005]. Quantencomputer basieren auf Quantenbits, eine Erweiterung der klassischen Bits. Quantenbits zeichnen sich unter anderem dadurch aus, dass sie mehrere Zustände auf einmal annehmen, was die Entwicklung von völlig neuen Algorithmen ermöglicht.

Einer der bekanntesten Quantenalgorithmen ist der Shor-Algorithmus. Dieser berechnet die Primfaktorzerlegung einer Zahl in polynomieller Zeit. Bisher ist es jedoch nicht ge-

lungen einen Quantencomputer zu bauen, der eine entsprechend große Zahl faktorisieren könnte. Sollte in der Zukuft jedoch ein entsprechend leistungsfähiger Quantencomputer gebaut werden, wäre RSA nicht mehr sicher [Bernstein et al., 2008].

4.2 Diffie-Hellman-Schlüsselaustausch

Der Diffie-Hellman-Schlüsselaustausch ist ein Verfahren mit dem es möglich ist einen Schlüssel über einen unverschlüsselten Kanal auszutauschen. Es ist jedoch kein asymmetrischer Algorithmus wie beispielsweise das RSA-Verfahren. Der Diffie-Hellman-Schlüsselaustausch basiert ähnlich wie RSA auf einem schwierigen mathematischen Problem. In diesem Fall auf der Berechnung des diskreten Logarithmus in zyklischen Gruppen [Buchmann, 2010, S. 155]. Bevor das eigentliche Verfahren vorgestellt wird, sollen die mathematischen Grundlagen erläutert werden.

4.2.1 Mathematische Grundlagen

Zunächst soll der Begriff der Gruppe definiert werden. Eine Gruppe ist ein Tupel (𝕄,*), bestehend aus einer Menge 𝕄 und einer Verknüpfung *, sodass folgende Eigenschaften gelten [Steger, 2007, S.196]:

- Für alle a, $b \in \mathbb{M}$ ist $a * b \in \mathbb{M}$
- Für alle a, b, $c \in \mathbb{M}$ ist $(a * b) * c = a * (b * c)$
- Es gibt ein neutrales Element $e \in \mathbb{M}$ sodass für alle $a \in \mathbb{M}$ gilt $e * a = a = a * e$
- Jedes Element $a \in \mathbb{M}$ besitzt ein inverses Element $b \in \mathbb{M}$ sodass $a * b = e$

Eine Gruppe (𝕄,*) heißt zyklisch, wenn sie ein erzeugendes Element $g \in \mathbb{M}$ in dem Sinne besitzt, dass alle ihre Elemente durch iteriertes Verknüpfen des erzeugenden Elementes g und seines Inversen entstehen.

Eine für Diffie-Hellman geeignete Gruppe ist die Gruppe $(\mathbb{Z}_n,\cdot_n)$ mit $\mathbb{Z}_n = \{1,2,...,n-1\}$, wobei n eine Primzahl ist. Die Verknüpfung $\cdot_n$ entspricht der Multiplikation von ganzen Zahlen modulo n.

Der diskrete Logarithmus in $(\mathbb{Z}_n,\cdot_n)$ zu einer Zahl A und einer Basis g, ist definiert durch die Zahl x sodass $A = g^x \ mod \ p$.

	i=1	i=2	i=3	i=4	i=5	i=6
$3^i \ mod \ 7$	3	2	6	4	5	1

Abbildung 4.1: Veranschaulichung des erzeugenden Elements 3 in der Gruppe $(\mathbb{Z}_7,\cdot_7)$

Beispiel 4.2.1. *Betrachten wir die Gruppe $(\mathbb{Z}_7,\cdot_7)$. Ein erzeugendes Element der Gruppe ist $g = 3$ wie man in Abbildung 4.1 sieht. Der diskrete Logarithmus von 6 in der Gruppe $(\mathbb{Z}_7,\cdot_7)$ zur Basis 3 ist also 3, denn* $3^3 \mod 7 = 6$.

4.2.2 Schlüsselaustausch

Im Folgenden wird beschrieben, wie in diesem Verfahren zwei Teilnehmer A und B schrittweise einen Schlüssel erzeugen können, ohne diesen selbst zu übertragen.

- A und B einigen sich auf eine zyklische Gruppe primer Ordnung p und ein erzeugendes Element g dieser Gruppe
- A wählt eine geheime Zufallszahl $a < p$
- B wählt eine geheime Zufallszahl $b < p$
- A berechnet $A = g^a \ (mod\ p)$ und schickt A an B
- B berechnet $B = g^b \ (mod\ p)$ und schickt B an A
- A berechnet $k = B^a \ (mod\ p)$
- B berechnet $k = A^b \ (mod\ p)$

Nachdem alle Schritte durchgeführt worden sind, haben A und B jeweils denselben Schlüssel k erzeugt, ohne dass dieser selber übertragen wurde. Dieser kann anschließend beispielsweise für die Verschlüsselung von Nachrichten durch einen symmetrischen Verschlüsselungsalgorithmus verwendet werden.

4.2.3 Sicherheit von Diffie-Hellman

Ähnlich wie RSA beruht die Sicherheit von Diffie-Hellman hauptsächlich auf einem schwierigen mathematischen Problem, in diesem Fall auf der Berechnung des diskreten Logarithmus. Dies bedeutet auch, dass wenn ein effizienter Algorithmus dafür gefunden wird, Diffie-Hellman nicht mehr sicher ist. Im Folgenden sollen zunächst Algorithmen zur Berechnung des diskreten Logarithmus vorgestellt werden. Anschließend werden weitere Bedrohungen für Diffie-Hellman vorgestellt.

4.2.3.1 Algorithmen zur Berechnung des diskreten Algorithmus

Für die Berechnung des diskreten Logarithmus gibt es eine Vielzahl von verschiedenen Algorithmen, von denen an dieser Stelle jedoch nur eine exemplarische Auswahl vorgestellt werden kann. Mehr dazu lässt sich beispielsweise in *Einführung in die Kryptographie* [Buchmann, 2010] nachlesen.

Im Folgenden seien A, g und p gegeben, sodass $A = g^x \ (mod\ p)$ gelte. Die folgenden Algorithmen sollen also x bestimmen.

4.2.3.1.1 Enumeration

Das Enumerationsverfahren ist ein sehr einfaches Verfahren, welches schrittweise alle möglichen Werte für x ausprobiert. Im schlimmsten Fall müssen x-1 Multiplikationen und x Vergleiche durchgeführt werden. Für hinreichend große x ist der Rechenaufwand zu hoch, um ihn praktisch durchzuführen. Der Speicherbedarf hingegen ist ziemlich gering, da lediglich A, g, p sowie g^x $(mod\ p)$ gespeichert werden müssen [Buchmann, 2010, S. 178].

4.2.3.1.2 Shanks Babystep-Giantstep-Algorithmus

Der Babystep-Giantstep-Algorithmus von Shanks hat eine deutlich verbesserte Laufzeit im Vergleich zum Enumerationsverfahren, allerdings dafür einen viel höheren Speicherbedarf [Buchmann, 2010, S.178-180].

Zunächst berechnet man $m = \lceil \sqrt{p} \rceil$ und macht den Ansatz $x = qm+r,\ 0 \leq r < m$. Die Werte q und r werden durch den Algorithmus berechnet. Dazu berechnet man zunächst die Menge der Babysteps $\mathbb{B} = \{(Ag^{-r}, r) \mid 0 \leq r < m\}$. Falls ein Paar (1,$r$) in $\mathbb{B}$ zu finden ist, kann man $x = r$ setzen und man hat den diskreten Logarithmus gefunden. Falls dies nicht der Fall ist, bestimmt man zunächst $y = g^m$. Anschließend überprüft man für q = 1,2,3... ob y^q als erste Komponente in einem Element in $\mathbb{B}$ vorkommt, also ob (y^q,r) $\in \mathbb{B}$. Sobald dies der Fall ist, hat man diskreten Logarithmus $x = qm + r$ gefunden.

Der Zeit- und Platzbedarf von Shanks Babystep-Giantstep-Algorithmus liegt in der Größenordnung $\sqrt{p}$. Die Laufzeit konnte im Vergleich zum vorherigen Algorithmus zwar verbessert werden, doch für $p > 2^{160}$ war dies 2010 praktisch nicht mehr durchführbar.

Beispiel 4.2.2. *Wir betrachten die Gruppe $(\mathbb{Z}_{89}, \cdot_{89})$, welche das erzeugende Element 3 besitzt und wollen den diskreten Logarithmus von 55 berechnen. Gesucht ist also ein x, sodass*

$$55 = 3^x mod\ 89$$

gilt.

Zunächst berechnen wir $m = \lceil \sqrt{89} \rceil = 10$ sowie die Menge $\mathbb{B} = \{(55,0), (48,1), (16,2), (35,3), (71,4), (83,5), (87,6), (29,7), (69,8), (23,9), (67,10)\}$.

Da es kein Element (1,r) in $\mathbb{B}$ gibt, berechnen wir $y = 3^m \mod 89 = 3^{10} \mod 89 = 42$. Nun berechnen wir die ersten zehn Giantsteps, das heißt die Menge $G = \{42^q\}$ mit $q = 1, \ldots, 10$. Also $G = \{42, 73, 40, 78, 72, 87, 5, 32, 9, 22\}$.

Wir sehen, dass die Zahl $42^6 \mod 89 = 87$ eine erste Komponente in $\mathbb{B}$ ist, also (87,6) $\in \mathbb{B}$. Der diskrete Logarithmus lautet also $x = qm + r = 6 \cdot 10 + 6 = 66$.

4.2.3.2 Man-in-the-Middle-Angriff

Ein Man-in-the-Middle-Angriff ist ein Angriff, bei dem sich ein Angreifer in die Leitung zwischen die Teilnehmer A und B einschaltet. Der Angreifer gibt sich gegenüber B als A und gegenüber A als B aus. Jegliche Kommunikation zwischen A und B läuft dann, wie in Abbildung 4.2 dargestellt, über den Angreifer. Falls A und B diesen Angreifer nicht bemerken, werden zwei Schlüssel erzeugt: Einer zwischen A und dem Angreifer und

einer zwischen B und dem Angreifer. Dies führt dazu, dass der Angreifer die gesamte verschlüsselte Sitzung abhören kann. Um dies zu verhindern, müssen Maßnahmen ergriffen werden, welche die Authentizität des Absenders garantieren. Eine Möglichkeit besteht darin, die Nachrichten mit digitalen Signaturen zu signieren.

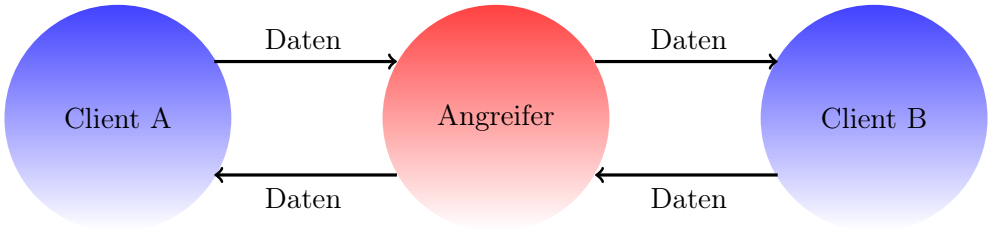

Abbildung 4.2: Man-in-the Middle – alle Daten fließen über einen Angreifer

4.2.4 Diffie-Hellman auf elliptischen Kurven

Bisher haben wir den Diffie-Hellman-Schlüsselaustausch nur auf der muliplikativen, zyklischen Gruppe modulo p betrachtet. Allerdings ist der Diffie-Hellman-Schlüsselaustausch auf allen Gruppen denkbar, in denen die Gruppenoperation einfach durchzuführen ist und die Berechnung des diskreten Logarithmus schwer ist. Einige dieser Gruppen basieren auf den elliptischen Kurven. Ein großer Vorteil dieser Gruppen ist die geringe Schlüssellänge im Vergleich zu anderen asymmetrischen Verfahren wie beispielsweise RSA [Swoboda et al., 2008, S. 135-139].

Eine elliptische Kurve ist eine Kurve welche durch folgende Gleichung definiert ist:

$$y^2 = x^3 + ax + b$$

In Abbildung 4.3 sieht man ein Beispiel für eine elliptische Kurve mit den Parametern a = -3 und b = 5.

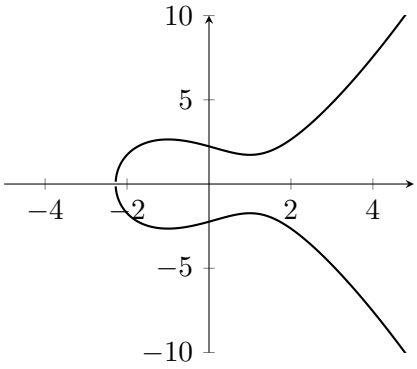

Abbildung 4.3: Die elliptische Kurve $y^2 = x^3 - 3x + 5$

Die Punkte einer elliptischen Kurve bilden zusammen mit einer Punktaddition eine Gruppe. Die Menge der Punkte einer elliptischen Kurve bezeichnen wir im Folgenden als $\mathbb{E}$. Damit die Punktaddition für alle Punkte definiert ist, müssen wir an die Parameter a und b folgende Anforderung stellen:

$$4a^3 + 27b^2 \neq 0$$

Die folgende Punktaddition in elliptischen Kurven ist geometrisch begründet, wobei man zwei Fälle unterscheiden muss. Wenn zwei Punkte P und Q eine unterschiedliche x-Koordinate haben, kann man durch diese Punkte eine Gerade ziehen, welche die Kurve in genau einem weiteren Punkt R^* schneidet. Durch die Spiegelung von R^* mit der x-Achse wird der Punkt $R = P + Q$ gebildet. Falls die Punkte P und Q dieselbe x-Koordinate haben, ist die Gerade vertikal und schneidet keinen weiteren Punkt der Kurve. Für diesen Fall definiert man einen Punkt im Unendlichen der als 0 bezeichnet wird und das neutrale Element der Gruppe bildet. Die Punkte P und Q sind also invers zueinander, wenn ihre x-Koordinate gleich ist. In Algorithmus 11 ist die Punktaddition in elliptischen Kurven noch einmal zusammengefasst.

Algorithmus 11 : Punktaddition in elliptischen Kurven

Eingabe : P, $Q \in \mathbb{E}$
Ausgabe : $R \in \mathbb{E}$
1 **if** $P.getX() \neq Q.getX()$ **then**
2 | *Bilde die Gerade f welche durch P und Q geht*;
3 | *Ermittle den dritten Schnittpunkt R^* von f und der Kurve $\mathbb{E}$*;
4 | *Bilde R durch die Spiegelung von R^* mit der $x - Achse$*;
5 | **return** R;
6 **else**
7 | **return** 0 ; `/* Die Punkte sind invers */`
8 **end**

Für kryptografische Anwendungen verwendet man in der Regel elliptische Kurven auf diskreten Mengen. Hierfür gibt es verschiedene Möglichkeiten, welche jedoch den Rahmen dieser Arbeit sprengen. Mehr Informationen zu elliptischen Kurven in der Kryptografie lassen sich in [Swoboda et al., 2008] und [Werner, 2002] finden.

4.3 Code-based Kryptografie: McEliece Verfahren

Aufgrund der potentiellen Gefahr von Quantencomputern für die Kryptografie sind viele Kryptologen auf der Suche nach Kryptosystemen, die auch gegen Angriffe mit Hilfe von Quantencomputer sicher sind. Die zuvor vorgestellten Verfahren RSA und Diffie-Hellman sind beispielsweise nicht post-quanten sicher, im Gegensatz zum symmetrischen AES-Verfahren [Bernstein et al., 2008]. Im Folgenden soll mit dem McEliece Verfahren ein asymmetrisches Verfahren vorgestellt werden, bei dem davon ausgegangen wird,

dass es post-quanten sicher ist, also auch dann noch verwendet werden kann, wenn es leistungsstarke Quantencomputer gibt.

Das McEliece-Verfahren ist ein asymmetrisches Verschlüsselungsverfahren, welches bereits 1978 von Robert J. McEliece vorgestellt wurde [McEliece, 1978]. Das Verfahren basiert auf fehlerkorrigierenden Codes. Klartexte werden zunächst über eine Generatormatrix in einen Goppa-Code, das ist eine Klasse von Codes die in polynomialzeit decodiert werden können, umgewandelt und anschließend als linearer Code getarnt. Zudem wird noch ein Fehlervektor addiert.

Formal kann ein McEliece-Kryptosystem durch ein Tripel (n,k,d) beschrieben werden, wobei die Parameter folgende Bedeutung haben:

- n: Blocklänge eines Cipherblocks
- k: Blocklänge eines Klartextblocks
- d: Minimaler Hamming-Abstand des Codes $\mathbb{C}$

Daraus lässt sich zudem die Anzahl der Fehler $t = \frac{d-1}{2}$ berechnen, die der Code $\mathbb{C}$ korrigieren kann.

Im Folgenden soll die verschiedenen Bestandteile des McEliece-Verfahrens genauer beschrieben werden.

4.3.1 Schlüsselerzeugung

Die Schritte um einen Schlüssel zu erzeugen sind:

- Wahl eines binären, linearen (n,k) Code $\mathbb{C}$ welcher t Fehler korrigieren kann und einen effizienten Dekodierungsalgorithmus besitzt
- Generierung einer Generationsmatrix G der Größe k x n für den Code $\mathbb{C}$
- Erzeugung einer zufälligen, binären Matrix S der Größe k x k deren Determinante nicht 0 ist
- Erzeugung einer zufälligen Permutationsmatrix P der Größe n x n
- Berechnung der k x n Matrix $\hat{G} = SGP$
- Der öffentliche Schlüssel ist ($\hat{G}$,t), der private Schlüssel ist (G,S,P)

4.3.2 Verschlüsselung

Mit dem McEliece-Verfahren können nur binäre Blocke $m \in (\mathbb{Z}_2)^k$ verschlüsselt werden. Klartextblöcke sind also binäre Vektoren der Länge k. Falls die zu sendende Nachricht länger ist, muss diese zunächst in verschiedene Blöcke geteilt werden.

Um einen Block m zu verschlüsseln wird dieser zunächst mit der Matrix $\hat{G}$ multipliziert, welche Teil des öffentlichen Schlüssel ist. Anschließend wird auf den so erzeugten Vektor ein zufälliger binärer Vektor $z \in (\mathbb{Z}_2)^n$ addiert, welcher maximal t Einsen besitzt. In Algorithmus 12 ist die Verschlüsselung noch einmal dargestellt.

Algorithmus 12 : Verschlüsselung im McEliece Verfahren

Eingabe : $m \in (\mathbb{Z}_2)^k$, $(\hat{G},t)$
Ausgabe : $c \in (\mathbb{Z}_2)^n$
1 $c := m \cdot \hat{G}$;
2 $z :=$ `randomVector`(n,t) ; `/* Bilde Zufallsvektor` $z \in (\mathbb{Z}_2)^n$ `mit maximal` t `Einsen */`
3 $c := c + z$;
4 **return** c;

4.3.3 Entschlüsselung

Bei der Entschlüsselung wird mit Hilfe des privaten Schlüssels die ursprüngliche Nachricht wieder hergestellt. Dazu wird zunächst die Permutation rückgängig gemacht, indem der Geheimtext c mit der inversen Permutationsmatrix P^{-1} multipliziert wird. Anschließend wird der Dekodierungsalgorithmus von $\mathbb{C}$ benutzt, um einen Vektor $\hat{m}$ zu erzeugen, welcher von den hinzugefügten Fehlern bereinigt wurde. Anschließend kann die ursprüngliche Nachricht m durch Muliplikation von $\hat{m}$ und der zu S inversen Matrix berechnet werden. In Algorithmus 13 ist die Entschlüsselung beschrieben.

Eine ausführlichere Beschreibung zu fehlerkorrigierenden Codes findet sich in *Informations- und Kodierungstheorie* [Schönfeld et al., 2012]. Weitere Informationen zu Goppa-Codes sind insbesondere in der Doktorarbeit *Goppa Codes & the McEliece Cryptosystem* [Jochemsz, 2002] beschrieben.

Algorithmus 13 : Entschlüsselung im McEliece Verfahren

Eingabe : $c \in (\mathbb{Z}_2)^n$, (G,S,P)
Ausgabe : $p \in (\mathbb{Z}_2)^k$
1 *Berechne die zu* P *inverse Matrix* P^{-1};
2 *Berechne die zu* S *inverse Matrix* S^{-1};
3 $\hat{c} := cP^{-1}$;
4 $\hat{m} :=$ `decode`$(\hat{c})$;
5 $m := \hat{m} \cdot S^{-1}$;
6 **return** p;

Im Folgenden soll noch ein Beispiel für einen Verschlüsselungs- und Entschlüsselungsvorgang im McEliece-Verfahren angegeben werden [Hochschule Augsburg, 2015].

Beispiel 4.3.1. *Wir betrachten einen (7,4) Hamming-Code* $\mathbb{C}$ *der einen Hamming-Abstand von 3 hat, welcher einen Fehler korrigieren kann und durch folgende Generatormatrix*

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

erzeugt wird. Des Weiteren wurden die zufälligen Matrizen

$$S = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

sowie

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

erzeugt. Durch Multitplikation bekommen wir die Matrix

$$\hat{G} = SGP = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix},$$

der öffentliche Schlüssel ist also $(\hat{G}, 1)$*.*

Jetzt soll die Nachricht $m = \begin{pmatrix} 1 & 1 & 0 & 1 \end{pmatrix}$ *verschlüsselt werden. Dazu wählen wir den Vektor* $z = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$.

Wir berechnen den Geheimtext $c = m\hat{G} + z = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$.

Um die Verschlüsselung wieder rückgängig zu machen, berechnen wir zunächst $\hat{c} = cP^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$*. Anschließend ermitteln wir den dekodierten Code* $\hat{m} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$ *(wir suchten ein* $\hat{m}$*, sodass der Hamming-Abstand von* $\hat{m}\hat{G}$ *und* $\hat{c}$ *kleiner gleich eins ist). Jetzt können wir* $m = \hat{m}S^{-1} = \begin{pmatrix} 1 & 1 & 0 & 1 \end{pmatrix}$ *rekonstruieren.*

4.3.4 Eigenschaften des McEliece-Verfahren

Das McEliece-Verfahren hat einige sehr positive Eigenschaften, dennoch wird es in der Praxis nicht verwendet. Es gilt als sicher gegen Angriffe durch klassische sowie durch Quantencomputer. Zudem sind die Verschlüsselung sowie die Entschlüsselung vergleichsweise schnell.

Der Grund warum es dennoch nicht verwendet wird, sind die großen Schlüssellängen, welche durch die verschiedenen Matrizen erreicht werden. Dies führt dazu, dass die Schlüsselgröße in den Megabyte-Bereich gehen kann [Bernstein et al., 2008, S. 95].

In der ursprünglichen Variante hat McEliece ein System mit den Parametern (1024, 524, 101) für eine sichere Verschlüsselung vorgeschlagen [McEliece, 1978]. In einer Untersuchung der TU Darmstadt welche 2012 veröffentlicht wurde [Niebuhr et al., 2012], werden jedoch deutlich höhere Parameter empfohlen. Für 2016 sollten die Parameter beispielsweise (1833, 1356, 44) sein und für eine Sicherheit bis 2050 (2804, 2048, 66).

4.4 Vergleich mit symmetrischen Algorithmen

Ein großer Vorteil von asymmetrischen Algorithmen ist die Aufteilung in öffentlichen und privaten Schlüssel. Somit entfällt der größte Nachteil von symmetrischen Algorithmen: der Schlüsselaustausch. Dies wurde durch schwierige mathematische Probleme ermöglicht. Falls diese Probleme in der Zukunft gelöst werden sollten, verfällt die Sicherheit der jeweiligen Algorithmen. Diese Gefahr besteht bei symmetrischen Algorithmen wie AES nicht.

Ein Nachteil der meisten asymmetrischen Algorithmen ist zudem die benötigte Rechenkraft. RSA ist beispielsweise um den Faktor 1000 langsamer als AES [Schmeh, 2013, S. 199].

Um die Stärken von symmetrischen und asymmetrischen Algorithmen zu vereinen, kann man Hybrid-Verfahren verwenden [Schmeh, 2013, S. 199]. Dies funktioniert in der Regel so, dass zunächst mit Hilfe eines asymmetrischen Verfahrens, wie beispielsweise RSA, ein Sitzungsschlüssel ausgetauscht wird. Dieser wird dann für eine symmetrische Verschlüsselung verwendet. Somit wird das symmetrische Problem des Schlüsselaustausches gelöst und die langsame Verschlüsslung asymmetrischer Verfahren auf den Schlüssel beschränkt.

5 Methoden der Authentifizierung

In diesem Kapitel sollen Methoden vorgestellt werden, wie sich eine Instanz A gegenüber einer Instanz B authentifizieren kann. Unter einer Authentifizierung verstehen wir die Erbringung eines Beweises, welcher die Echtheit von A beweist. Nachdem A von B authentisiert wurde, kann B also sicher sein, dass A die Person ist, für die sie sich A ausgibt [Schmeh, 2013, S. 403–404].

Zunächst sollen einige Anwendungen aufgezählt werden, in denen eine Authentifizierung notwendig oder zumindest sinnvoll ist:

- Grenzkontrolle bei einer Reise ins Ausland
- Abholen eines Paketes bei der Post
- Schreiben einer Klausur an der Universität
- Email mit beweisbarer Echtheit
- Onlinebanking
- Einloggen in einen Account auf einem Computer / im Internet
- Geldabheben am Bankautomat
- Entsperren eines Smartphones
- Schließberechtigungen von Schließfächern und Türen
- Schreiben eines persönlichen Briefes

Die Liste ließe sich noch beliebig weiterführen. Es gibt also eine Vielzahl an Situationen, in denen eine Authentifizierung notwendig ist. Zudem ist der Vorgang der Authentifizierung kein Aufgabenbereich, der sich nur auf den Computer beschränkt. Im Gegenteil, es gibt auch im echten Leben viele Situationen, in denen man sich authentifizieren muss. Hierauf soll aber nicht im Detail eingegangen werden.

5.1 Einführung

Im vorherigen Abschnitt haben wir gesehen, dass eine Authentifizierung auf verschiedene Weisen erfolgen kann. Konkret unterscheidet man drei Möglichkeiten, worauf eine Authentifizierung basieren kann: auf Wissen, Besitz oder eine bestimmte Eigenschaft [Schmeh, 2013, S. 404].

Der Vorgang der Authentifizierung von Computern, beispielsweise wenn ein Client sich gegenüber einem Server authentifizieren muss, wird in der Regel durch Protokolle geregelt. Ein Protokoll gibt im Prinzip den genauen Ablauf der zu übertragenden Nachrichten vor.

Neben Protokollen gibt es noch die Möglichkeit, digitale Signaturen zu verwenden. Eine digitale Signatur kann mit einer Unterschrift auf einem Brief verglichen werden.

5.1.1 Authentifizierung durch Wissen

Bei einer Authentifizierung durch Wissen geschieht die Authentifizierung durch etwas, was im besten Falle nur die Person weiß, die sich authentifizieren muss. Im Folgenden sollen verschiedene Möglichkeiten zur wissensbasierten Authentifizierung thematisiert werden.

5.1.1.1 Passwörter

Passwörter (auch als PIN oder Geheimnummer bekannt) sind eines der am meisten verwendeten Mittel zur Authentifizierung. Authentifizierungen durch Passwörter sind im Prinzip sehr simpel und effizient. Der Benutzer sucht sich in der Regel beim Erstellen eines Accounts ein Passwort aus. Der Betreiber speichert einen Hashwert von dem Passwort in der Datenbank. Wenn sich der Benutzer authentifizieren möchte, schickt er das Passwort an den Betreiber. Dieser berechnet den Hashwert und vergleicht mit dem Eintrag in der Datenbank. Bei einer Übereinstimmung ist der Benutzer authentifiziert.

Ein großes Problem bei Passwörtern ist jedoch der Mensch selbst, da wir oft zu Bequemlichkeit neigen. Wenn der Benutzer bei jeder Website im Internet dasselbe Passwort benutzt, und dazu noch ein sehr einfach zu erratendes, wird einem Angreifer das Abgreifen der Daten sehr einfach gemacht. In Abbildung 5.1 [Bube, 2014] sieht man die 9 am häufigsten verwendeten Passwörter aus dem Jahr 2013. Auf Platz 1 ist beispielsweise das Passwort *password*, welches nicht gerade sicher scheint.

1.	password	2.	123456	3.	12345678
4.	1234	5.	qwerty	6.	12345
7.	dragon	8.	pussy	9.	baseball

Abbildung 5.1: Die 9 am häufigsten verwendeten Passwörter [Bube, 2014]

5.1.1.2 Passgesten

Passgesten sind eine Alternative zu gewöhnlichen Passwörtern und werden insbesondere auf Smartphones eingesetzt. Sie können als musterartige Passwörter verstanden werden, bei denen man zum Beispiel mit der Hand Punkte durch Linien verbindet. Passgesten gelten zum einen als besser merkbar und zum anderen sind sie durch den Touchscreen leicht umzusetzen [Schmeh, 2013, S. 406]. Problematisch können jedoch hinterlassene

Spuren auf dem Display sein, welche zum Beispiel durch fettige Finger erzeugt werden können.

5.1.1.3 Persönliche Informationen

Eine weitere Möglichkeit zur Authentifizierung mit Hilfe von Wissen bilden persönliche Informationen. Dies kann das Geburtstagsdatum sein, das Sternzeichen, der Name des besten Freundes oder die Lieblingsfarbe. Ein großer Vorteil von persönlichen Informationen ist es, dass der Benutzer die Information mit hoher Wahrscheinlichkeit nicht vergisst. Zudem hat ein Angreifer in der Regel keinen Zugriff auf diese Informationen [Schmeh, 2013, S. 406].

5.1.2 Authentifizierung durch Besitz

Bei einer Authentifizierung durch Besitz erfolgt die Authentifizierung durch eine Sache, welche im Besitz der Person ist, die sich authentifizieren möchte. Diese Sache muss also vorher bereits erzeugt worden sein und unmittelbar mit der Identität des Benutzers verknüpft sein.

5.1.2.1 Personalausweis

Ein Beispiel, für einen Gegenstand, der zur Authentifizierung durch Besitz verwendet wird und den jeder kennt, ist der Personalausweis. Der Personalausweis wird in der Regel von einer bestimmten Behörde ausgestellt und muss persönlich beantragt und abgeholt werden. Mit dem Personalausweis kann sich ein Bürger gegenüber anderen Menschen und Behörden ausweisen. Er kann damit also beweisen, dass er wirklich eine bestimmte Person ist. Im Internet wird der Personalausweis eher selten benutzt, da dies vergleichsweise aufwendig ist. In bestimmten Situationen kann er dennoch zur Authentifizierung über das Internet verwendetet werden, beispielsweise indem ein Foto übertragen wird.

Mit der Einführung des neuen Personalausweises sind allerdings deutlich verbesserte Möglichkeiten hinzugekommen, so bietet er nun eine Online-Ausweisfunktion sowie eine Unterschriftsfunktion an [Bundesministerium des Innern, 2015].

5.1.2.2 Einmalkennwort

Ein Einmalkennwort ist im Prinzip ein Passwort, welches genau einmal verwendet werden kann. Es muss also für jeden Authentifizierungsvorgang ein neues Einmalkennwort benutzt werden. Der Vorteil von Einmalkennwörtern ist, dass wenn ein Einmalkennwort abgefangen wird, ein Angreifer dieses nicht weiter für Angriffe verwenden kann, da es nicht mehr gültig ist. Für die Realisierung von Authentifizierungssystemen mit Einmalkennwörtern gibt es weitestgehend zwei Möglichkeiten.

Die erste Möglichkeit ist die Verwendung einer Kennwortliste. Eine Kennwortliste muss zunächst für beide Seiten verfügbar gemacht werden, zum Beispiel in dem eine Organisation sie per Post an den Benutzer sendet. Anschließend kann sie zur Authentifizierung benutzt werden. Bei jedem Authentifizierungsvorgang wird ein Kennwort aus der Liste

gestrichen und nicht weiter verwendet. Sobald die Liste leer ist, muss eine neue Liste ausgetauscht werden.

Eine weitere Möglichkeit bieten Kennwortgeneratoren, welche in Hardware und in Software vorzufinden sind. Kennwortgeneratoren werden in der Regel am Anfang mit dem Server synchronisiert. Anschließend erzeugen sie laufend neue Einmalkennwörter, welche für eine bestimmte Zeitspanne gültig sind, zum Beispiel für 60 Sekunden. Aktuelle Beispiele für Kennwortgeneratoren sind SecurID [Wikipedia, 2015b] und der Battle.net Authenticator [Blizzard, 2015].

Einmalkennwörter sind häufig Bestandteil der Zwei-Faktor-Authentifizierung.

5.1.3 Authentifizierung durch Eigenschaft

Bei einer Authentifizierung durch Eigenschaft werden in der Regel Merkmale eines Menschen benutzt, die unmittelbar mit ihm verknüpft sind. Diese Art der Authentifizierung ist auch unter dem Namen biometrische Authentifizierung bekannt [Schmeh, 2013, S. 407]. Der Vorteil bei dieser Art der Authentifizierung ist, dass sie relativ schwer zu fälschen ist, da ein Angreifer beispielsweise erst einmal den Fingerabdruck des Ziels bekommen muss. Als mögliche Eigenschaft eignen sich unter anderem:

- Fingerabdruck
- Stimme
- Gesichtserkennung
- Iriserkennung

5.2 Protokolle

Im vorherigen Kapitel wurden Möglichkeiten vorgestellt, worauf eine Authentifizierung beruhen kann. Um eine Authentifizierung zwischen zwei Computern, zum Beispiel über das Internet, durchzuführen, muss jedoch zusätzlich genau spezifiziert werden, welcher Computer wann welche Nachricht überträgt. Diese Spezifizierung wird durch Protokolle geregelt. Falls eine Aktion vom Protokoll abweicht, zum Beispiel wenn ein falsches Passwort eingegeben wird, erfolgt keine Authentifizierung. Ein Angreifer wird also dadurch enttarnt, dass seine Aktionen vom Protokoll abweichen [Swoboda et al., 2008, S. 149]. Zunächst soll der Begriff des Protokolls definiert werden.

Definition 5.2.1. *Ein Kommunikationsprotokoll – kurz Protokoll – ist die Gesamtheit aller semantischen und syntaktischen Festlegungen, die das Kommunikationsverhalten miteinander kooperierender Einheiten definiert [Gabele et al., 1991, S. 87].*

Im Folgenden werden verschiedene kryptografische Protokolle mit dem Ziel der Authentifizierung vorgestellt. Wir gehen im Folgenden davon aus, dass sich ein Client A bei einem Server B authentifizieren möchte.

5.2.1 Passwortbasierte Authentifizierung

Passwortbasierende Authentifizierungsprotokolle sind im Prinzip sehr simpel und kommen häufig im Internet vor. Der Client A schickt seinen Benutzernamen zusammen mit seinem Passwort zum Server. Der Server vergleicht das gesendete Passwort mit dem in der Datenbank. Bei einer Übereinstimmung ist A authentifiziert.

Neben den in Abschnitt 5.1.1.1 vorgestellten Problemen von Passwörtern in Bezug auf den Menschen, hat dieses Protokoll noch weitere Schwächen. Zum einen muss das Passwort zunächst übertragen werden. Falls das Passwort als Klartext übertragen wird, kann ein Angreifer dieses abfangen und benutzen, um sich als A auszugeben. Es müssen also Vorkehrungen getroffen werden, um das Passwort verschlüsselt zu übertragen.

Zudem kann ein Angreifer ein abgefangenes Passwort für beliebig viele Authentifizierungsvorgänge in der Zukunft benutzen, da die Authentifizierung allein von dem Passwort abhängt.

Eine Ausnahme bilden Protokolle, bei denen Einmalpasswörter benutzt werden. Hier ermöglicht das Abfangen des Passwortes dem Angreifer in der Regel keine zukünftigen Möglichkeiten zur Authentifizierung [Swoboda et al., 2008, S. 151–152].

5.2.2 Challenge-Response-Authentifikation

Eine Challenge-Response-Authentifikation ist ein interaktives Protokoll, das aus zwei Bestandteilen besteht: der Challenge und dem Response. Bevor das Protokoll durchgeführt werden kann, muss einmal ein Schlüssel k zwischen A und B ausgetauscht werden [Schmeh, 2013, S. 414-415].

Wenn sich A bei B authentifizieren möchte, bekommt A von B eine Challenge r. Die Challenge ist eine Aufgabe, welche nur mit einem Geheimnis, dem Schlüssel, gelöst werden kann. Zum Beispiel könnte r eine Zufallszahl sein. Der Response ist die Antwort, die B erwartet und A zu berechnen hat. Dies könnte zum Beispiel ein gemeinsamer Hashwert von k und r sein. Wenn A an B den richtigen Response sendet, ist A authentifiziert.

Der Vorteil bei diesem Verfahren ist, dass das Passwort, nachdem es einmal ausgetauscht wurde, nicht mehr übertragen werden muss. Zudem ist der Response jedes mal anders, falls unterschiedliche Challenges verwendet werden. Das heißt ein Angreifer kann in diesem Fall die abgefangenen Reponsewerte nicht für zukünftige Angriffe verwenden.

5.2.3 Zwei-Faktor-Authentifizierung

Unter einer Zwei-Faktor-Authentifizierung versteht man eine Authentifizierung, die aus zwei unabhängigen Komponenten besteht. Ein sehr bekanntes Beispiel, ist das Geldabheben am Bankautomat. Hier braucht man in der Regel eine Bankkarte (Besitz) sowie eine PIN-Nummer (Wissen).

Im Internet wird von großen Unternehmen häufig eine Kombination aus Passwort und Einmalkennwort verwendet. Hierdurch werden die bereits erwähnten Probleme von Passwörtern abgemildert, da ein Angreifer mit dem Passwort allein noch keinen Zugang hat.

Nachteil der Zwei-Faktor-Authentifizierung sind zusätzlicher Aufwand und teilweise höhere Kosten, beispielsweise durch den Erwerb eines hardwarebasierenden Kennwort-

generators. Allerdings steigt hierdurch zweifelsfrei die Sicherheit der Authentifizierung, sodass mittlerweile viele Anbieter wie Google oder Facebook diese Form der Authentifizierung anbieten [Weck, 2014].

An der ETH Zürich wird derzeit an einer Zwei-Faktor-Authentifizierung geforscht [Karapanos et al., 2015], die auf Umgebungsgeräuschen basiert und den Aufwand gegenüber den meist eingesetzten Einmalkennwörtern reduzieren soll. Die grundsätzliche Idee ist, dass der Rechner sowie ein Smartphone Umgebungsgeräusche aufzeichnen, die anschließend verglichen werden. Somit müsste ein Angreifer das Smartphone des Opfers besitzen, oder sich am selben Ort befinden.

5.2.4 Zero-Knowledge-Verfahren

Zero-Knowledge-Verfahren verwenden sogenannte Zero-Knowledge-Beweise. Bei einem Zero-Knowledge-Beweis, versucht der Beweiser A, den Verifizierer B davon zu überzeugen, dass A ein Geheimnis kennt. Dabei kennt B das Geheimnis nicht und A verrät keine Informationen darüber. Sobald B davon überzeugt ist, dass A das Geheimnis wirklich kennt ist A durch B authentifiziert.

In der Regel laufen Zero-Knowledge-Protokolle über mehrere Runden ab, da in einer einzelnen Runde, A das Geheimnis einfach raten könnte. Falls A, den Verifizierer B jedoch über beliebig viele Runden davon überzeugen kann das Geheimnis zu kennen, konvergiert die Wahrscheinlichkeit dafür, dass A das Geheimnis nicht kennt gegen 0 [Beutelspacher et al., 2006, S. 36-43].

Im Folgenden soll mit der magischen Tür zunächst ein veranschaulichendes Beispiel dafür gegeben werden, wie ein Zero-Knowledge-Beweis aussehen kann. Anschließend wird mit dem Fiat-Shamir-Verfahren ein auf Computern praktisch durchführbares Verfahren vorgestellt.

5.2.4.1 Die magische Tür

In der Literatur ist das folgende Beispiel unter dem Namen Magische Tür bekannt [Beutelspacher et al., 2010, S. 196]. In diesem Szenario möchte A beweisen, dass er ein geheimes Passwort kennt, mit dem er eine magische Tür öffnen kann. Er will dies jedoch tun, ohne das Passwort zu verraten. In Abbildung 5.2 ist der schematische Aufbau dargestellt. Um in die linke oder rechte Seite zu gelangen, muss zunächst der Vorraum betreten werden. Das Protokoll besteht aus folgenden Schritten:

- A betritt den Vorraum und geht zufällig in den linken oder rechten Flügel, B sieht dabei nicht welchen Weg A nimmt
- B betritt den Vorraum und verlangt, dass A auf einer bestimmten Seite raus kommt

Wenn A das Passwort nicht kennt, besteht eine Wahrscheinlichkeit von 50%, dass er aus der richtigen Seite kommt. Falls A das Passwort jedoch kennt, kommt er in jedem Fall auf der richtigen Seite raus. Wenn man diese Prozedur beispielsweise 10 Mal durchführt, muss ein Angreifer 10 Mal richtig raten, um nicht entlarvt zu werden. Die Wahrscheinlichkeit

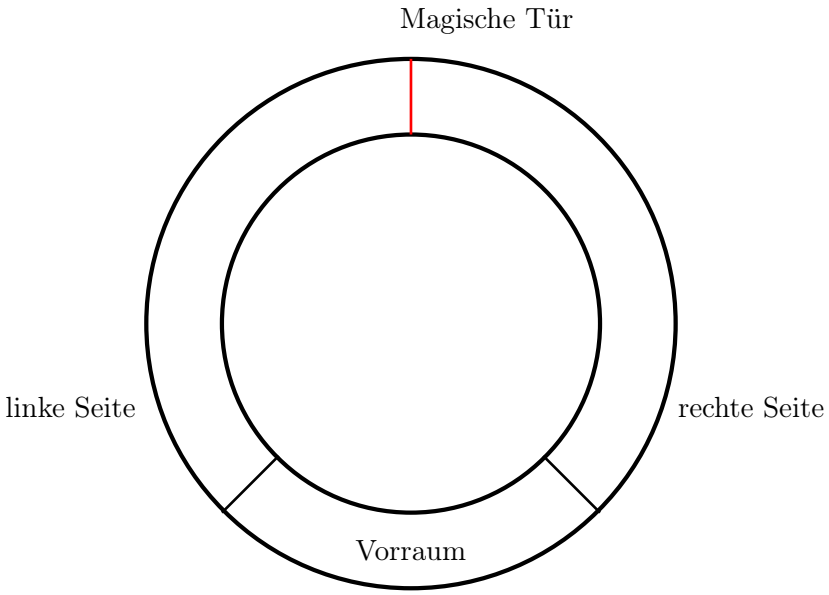

Abbildung 5.2: Schematische Darstellung der magischen Tür

dafür, dass er jedes Mal auf der richtigen Seite erscheint, beträgt allerdings nur noch 0,09765625%.

Im Allgemeinen kann, nach der Durchführung von t Protokollrunden, B sich mit einer Wahrscheinlichkeit von $1-(1/2)^t$ sicher sein, dass A das Geheimnis kennt – sofern A jedes mal die Aufgabe erfüllen konnte.

5.2.4.2 Fiat-Shamir-Protokoll

Das Fiat-Shamir-Protokoll wurde 1986 von Adi Shamir und Amos Fiat vorgestellt (siehe [Fiat und Shamir, 1986]). Das Protokoll basiert, ähnlich wie die vorgestellten asymmetrischen Algorithmen, auf einem mathematischen Problem: das Finden der modularen Quadratwurzel einer Zahl v. Wenn n keine Primzahl ist, dann ist es sehr schwer eine Zahl s zu finden, sodass

$$s^2 \mod n = v$$

gilt.

Bevor das eigentliche Protokoll durchgeführt werden kann, muss eine Schlüsselzentrale Schlüssel an die Benutzer austeilen. Hierzu wird zunächst die Zahl $n = p \cdot q$ berechnet, wobei p und q Primzahlen sind. Die Primzahlen müssen geheim bleiben, wohingegen n öffentlich gemacht wird. Zudem sollten p und q groß genug sein um einen Faktorisierungsangriff auf n unmöglich zu machen.

Anschließend berechnet die Schlüsselzentrale für alle Benutzer eine Zahl s und eine Zahl v mit $v = s^2 \ mod \ n$. Die Zahl s ist das Geheimnis des Benutzers, ähnlich dem Passwort für die magische Tür.

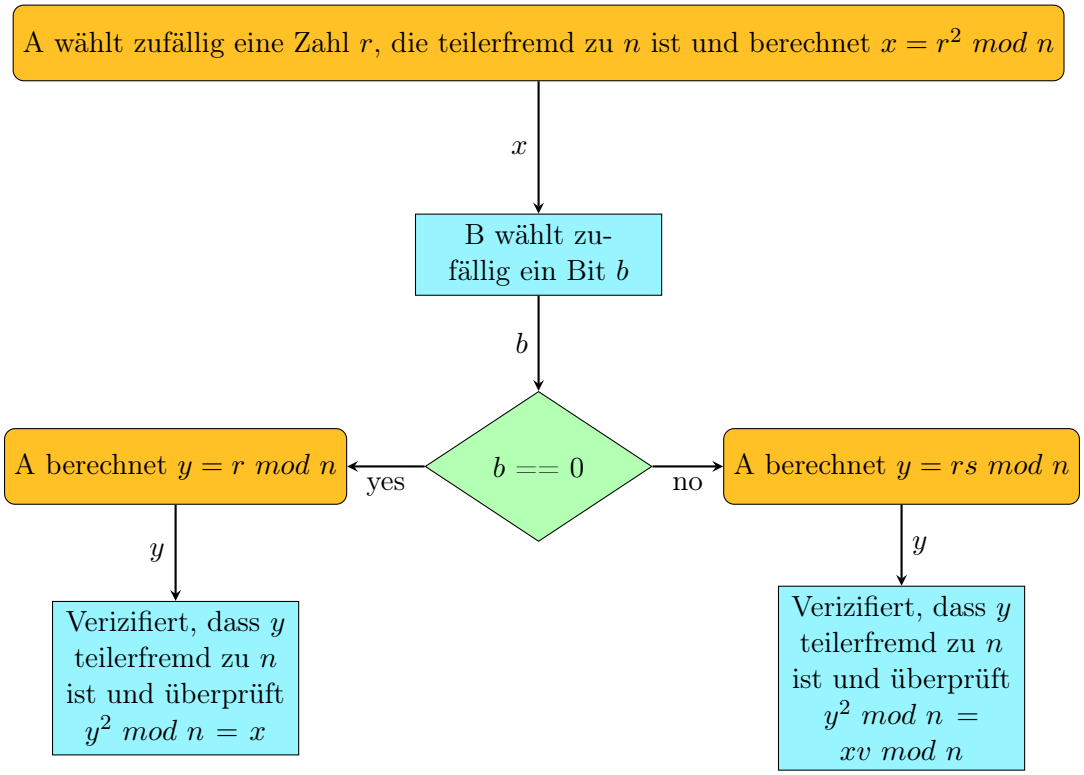

Abbildung 5.3: Ablauf einer Protokollrunde im Fiat-Shamir-Protokoll zwischen dem Beweiser A und dem Verifizierer B

Der Ablauf einer Protokollrunde ist in Abbildung 5.3 dargestellt. Es werden zunächst verschiedene Berechnungen durchgeführt, die am Ende von B verifiziert werden. In der Hälfte der Fälle kann der Wert von y nur richtig sein, wenn A das Geheimnis s kennt.

Im Folgenden wird ein beispielhafter Ablauf für eine Protokollrunde dargestellt.

Beispiel 5.2.1. *Die Schlüsselzentrale wählt* $p = 13$ *sowie* $q = 17$ *und berechnet*

$$n = p \cdot q = 13 \cdot 17 = 221.$$

Der Benutzer A bekommt die Werte

$$s = 53 \text{ und } v = 157$$

zugeteilt. Um sich zu verifizieren wählt A

$$r = 150$$

und schickt

$$x = r^2 \mod n = 179$$

an B. Daraufhin schickt B

$$b = 1$$

zurück an A. Nun berechnet A

$$y = 150 \cdot 53 \mod 221 = 215$$

und schickt y an B. Jetzt überprüft B ob

$$y^2 \mod n = xv \mod n$$

und sieht, dass

$$215^2 \mod 221 = 36 = 179 \cdot 157 \mod 221.$$

5.3 Authentifizierung mit Signaturen

Eine weitere Möglichkeit, um die Authentizität einer Nachricht zu gewährleisten, sind digitale Signaturen. Digitale Signaturen können als Gegenstück zur realen Unterschrift auf Briefen oder Postkarten verstanden werden.

Damit die digitale Signatur nicht zu fälschen ist, ist sie in der Regel abhängig von der konkreten Nachricht sowie einem spezifischen geheimen Schlüssel. Genauer gesagt, ist eine digitale Signatur zu einer Nachricht m, eine bestimmte Prüfsumme. Die Prüfsumme wird mit einem Signaturverfahren berechnet, welches als Eingabe die Nachricht selbst sowie den geheimen Schlüssel bekommt. Ein Verifizierer, in der Regel der Empfänger, der m sowie die zugehörige Signatur bekommt, kann nun mit einem öffentlichen Schlüssel aus der Signatur die ursprüngliche Nachricht m' berechnen. Falls diese Nachricht m' mit m übereinstimmt, kann der Verizifierer davon ausgehen, dass der Sender der ist für den er sich ausgibt.

An ein Verfahren, welches Signaturen erzeugt, werden folgende Voraussetzungen gestellt [Schmeh, 2013, S. 202]:

- Die Prüfsumme darf nicht zu fälschen sein
- Die Echtheit der Prüfsumme muss überprüfbar sein
- Die zugehörige Nachricht darf nicht unbemerkbar verändert werden
- Die Prüfsumme darf nicht von einer Nachricht zu einer Anderen übertragen werden

Ein Beispiel für ein Signaturverfahren wurde bereits mit dem RSA-Algorithmus vorgestellt. Um mit RSA Signaturen zu erzeugen, vertauscht man die Reihenfolge von Ver- und Entschlüsselung. Konkret berechnet der Absender die Signatur einer Nachricht, indem er auf die Nachricht die Entschlüsselungsfunktion von RSA unter Einbeziehung seines geheimen Schlüssel anwendet. Der Empfänger kann die Nachricht aus der Signatur rekonstruieren, indem er die Verschlüsselungsfunktion von RSA mit dem öffentlichen Schlüssel auf die Signatur anwendet.

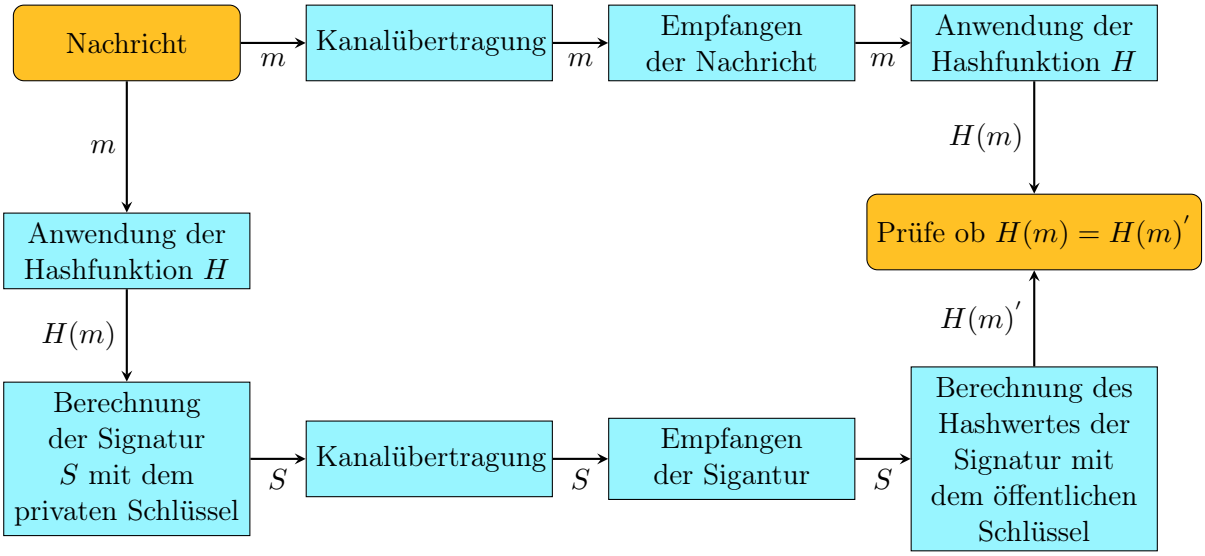

Abbildung 5.4: Schematische Darstellung von digitalen Signaturen unter Einbeziehung von Hashfunktionen

Neben RSA gibt es noch eine Menge von weiteren Signaturverfahren, auf die an dieser Stelle jedoch nicht weiter eingegangen werden soll.

Um den Rechenaufwand gering zu halten signiert man jedoch oft nicht die Nachricht selbst, sondern den zu einer Nachricht gehörigen Hashwert. In Abbildung 5.4 ist das Schema von Signaturen mit Hilfe von Hashfunktionen veranschaulicht. Der Sender berechnet zunächst den Hashwert der Nachricht m. Anschließend bildet er mit seinem privaten Schlüssel die Signatur von dem Hashwert. Daraufhin werden einerseits die Nachricht selbst, sowie die Signatur übertragen. Der Empfänger berechnet aus der Signatur mit dem öffentlichen Schlüssel den ursprünglichen Hashwert. Diesen vergleicht er mit dem Hashwert von m, denn er ebenfalls berechnen muss. Bei Übereinstimmung der Werte kann sich der Empfänger sein, dass die Nachricht von dem richtigen Sender stammt und nicht verändert wurde.

6 Steganografie

Der Begriff Steganografie stammt aus dem Griechischen und kann mit *verborgen schreiben* übersetzt werden. Im Grunde geht es darum, Nachrichten so zu übertragen, dass diese Übertragung nicht bemerkt wird. Dies geschieht in der Regel durch die Einbettung der Nachricht in ein Trägermedium, welches anschließend übertragen wird. Die ursprüngliche Nachricht wird also in einer zweiten Nachricht versteckt [Johnson et al., 2001, S. 1]. In diesem Kapitel sollen wichtige Begriffe sowie verschiedene Verfahren und Anwendungen der Steganografie vorgestellt werden. Zudem soll untersucht werden, inwieweit sich die Steganografie als Alternative zur Kryptografie eignet.

6.1 Grundlagen

Ein steganografisches System [Böhme, 2010, S. 12] ist ein Tupel $(\mathbb{M}, \mathbb{X}, \mathbb{K}, h, j)$ mit nichtleeren Mengen $\mathbb{M}, \mathbb{X}, \mathbb{K}$ und Abbildungen

$$h\colon \mathbb{M} \times \mathbb{K} \times \mathbb{X} \to \mathbb{X} \text{ und } j : \mathbb{X} \times \mathbb{K} \to \mathbb{M}$$

mit der Eigenschaft:

$$\forall k \ \in \mathbb{K} \ \exists k' \in \mathbb{K} : j(h(m,k,x),k') = m \ \forall m \in \mathbb{M}, \forall x \in \mathbb{X}.$$

Es heißen:

- $\mathbb{M}$ die Nachrichtenmenge
- $\mathbb{X}$ die Trägerdatenmenge
- $\mathbb{K}$ die Schlüsselmenge
- h Einbettungsfunktion
- j Extrahierungsfunktion

Als Trägermedium kommen die verschiedensten Objekte infrage. In der Vergangenheit waren dies beispielsweise Briefe, Gedichte, Musikstücke oder sogar Menschen. Bei der digitalen Steganografie kommen dagegen digitale Trägermedien, wie Bilder oder Audiodateien infrage [Schmeh, 2009].

In Abbildung 6.1 ist das Schema von steganografischen Systemen dargestellt. Zunächst wird eine Nachricht m abhängig von einem Schlüssel k in ein Trägermedium x eingebettet. Das resultierende Trägermedium x' wird über einen Kanal übertragen. Anschließend kann der Empfänger mit dem Schlüssel k' die Nachricht extrahieren. Ob k und k' gleich

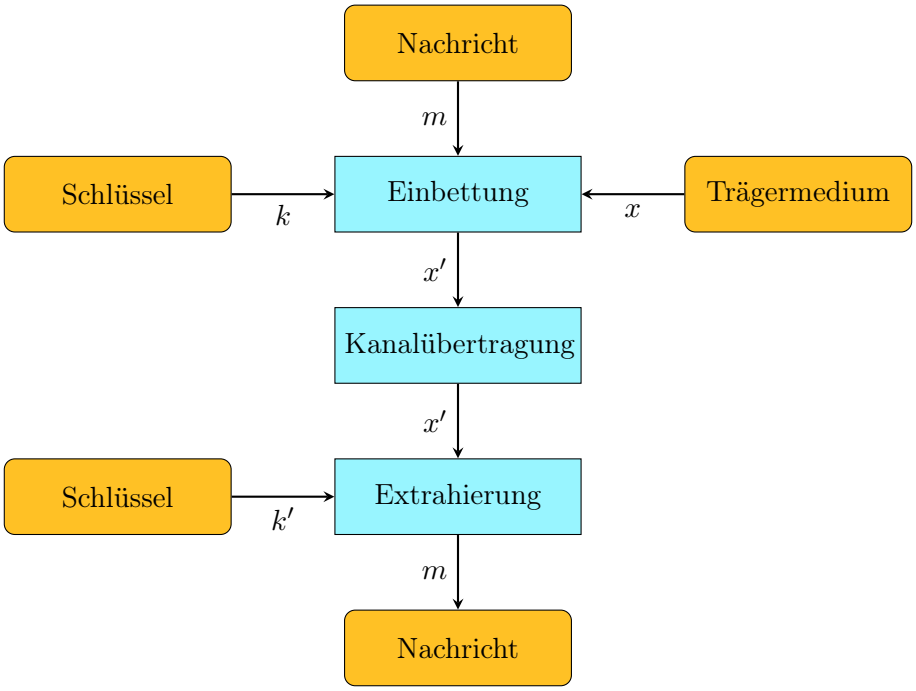

Abbildung 6.1: Schematische Darstellung von steganografischen Systemen

oder unterschiedlich sind, hängt davon ab ob es sich um ein symmetrisches oder ein asymmetrisches System handelt. Die Nachricht m die durch die Extrahierung gewonnen wird, ist in jedem Fall dieselbe, die ursprünglich eingebettet wurde.

Die Güte eines steganografischen Systems lässt sich anhand von drei Designzielen messen, die in der Regel jedoch nicht unabhängig sind und somit ausbalanciert werden müssen [Böhme, 2010, S. 17-21]:

- Kapazität
- Steganografische Sicherheit
- Robustheit

Die Kapazität eines steganografisches Systems gibt an wieviel Informationen in einem Trägermedium gespeichert werden können. Sie kann absolut oder relativ gemessen werden.

Die Steganografische Sicherheit dagegen gibt an wie sicher es ist, dass die geheime Nachricht nicht entdeckt wird. Je unwahrscheinlicher es ist, dass die Existenz einer geheimen Nachricht bemerkt werden kann, desto höher ist die Sicherheit des steganografischen Systems.

Die Robustheit wiederum gibt an, ob es möglich ist, dass ein Angreifer die geheime Nachricht ohne den Schlüssel extrahieren kann bzw. wieviel Aufwand damit verbunden ist. Um eine hohe Robustheit zu erreichen, sollte man das von der Kryptografie bekannte Kerckhoffs'sche Prinzip auch in der Steganografie berücksichtigen [Javidi, 2005, S. 123].

6.2 Altertümliche Methoden

Zunächst soll, anhand von einigen historischen Verfahren, ein Einblick in die Funktionsweise und den Möglichkeiten der Steganografie gewonnen werden.

6.2.1 Verwendung von Geheimtinte

Die Verwendung von Geheimtinte ist eine relativ bekannte Möglichkeit Nachrichten zu verstecken. Die älteste bekannte Anwendung geht auf das 2. Jahrhundert vor Christus zurück.

Das Prinzip der Geheimtinte ist recht simpel. Die Nachricht wird zunächst mit einer bestimmten Flüssigkeit auf Papier geschrieben. An die Flüssigkeit wird jedoch die Anforderung gestellt, dass unter normalen Umständen keine Spuren erkennbar sind. Erst unter Hinzunahme von Hitze oder durch eine chemische Reaktion darf die Nachricht sichtbar gemacht werden.

Die Anzahl der möglichen Flüssigkeiten hierfür ist beachtlich. Diese reichen von einfachen Pflanzensäften wie Zitronensaft oder Apfelsaft, über natürliche Flüssigkeiten wie Milch und Urin bis zu chemischen Stoffen wie beispielsweise einer Kaliumrhodanid-Lösung [Schmeh, 2009, S. 77-80].

6.2.2 Steganografie in historischen Konflikten

Die folgenden Verfahren stammen aus Geschichten des griechischen Geschichtsschreibers Herodot [Schmeh, 2009, S. 106-107].

In der ersten Geschichte geht es um Histiaios, einem enttäuschten griechischen Berater des Königs Darius I. von Persien. Um seinen Schwiegersohn Aristagoras zu einem Aufstand zu überreden, musste er sich für die Übertragung der Nachricht eine List ausdenken. Hierzu ließ er einem Sklaven, welchem er vertraute, die Haare abrasieren und tätowierte die Nachricht auf seinen Kopf. Nachdem die Haare nachgewachsen war, schickte er den Sklaven zu der Stadt von Aristagoras. Mit dieser Methode konnte die Nachricht unbemerkt überbracht werden.

In einer anderen Geschichte erzählt Herodot von dem Griechen Demaratos, welcher in Persien lebte und sein Heimatland vor einem bevorstehenden Angriff warnen wollte. Eine damals weit verbreitete Methode, um Nachrichten zu verschicken, war das Eingravieren von Nachrichten in Wachstafeln. Normalerweise wird die Nachricht in den Wachs gegossen. Demaratos jedoch ritzte die Nachricht in die unter dem Wachs liegende Holzschicht und tarnte Sie mit einem neuen Wachsüberguss. Somit konnte die Nachricht unbemerkt nach Griechenland gelangen, womit die Griechen gewarnt waren.

6.3 Linguistische Steganografie

Die Linguistische Steganografie befasst sich mit Methoden, um Nachrichten in Bildern, Texten, Symbolen und anderen Medien zu verstecken. Sie wird unterteilt in Semagramme und Open Codes.

6.3.1 Semagramme

Bei einem Semagramm wird eine Nachricht offen in das Trägermedium reincodiert. Dies kann zum Beispiel durch Details in einem Bild oder durch eine unauffällige Markierung bestimmter Buchstaben geschehen. Die Codierung sollte jedoch möglichst getarnt sein, um eine Erkennung durch Dritte zu verhindern.

Im zweiten Weltkrieg wurde beispielsweise ein Pullover als Semagramm benutzt. Deutsche Spione hatten Informationen über im Bau befindliche Schiffe in Form eines Pullovers nach Deutschland geschickt. Dort wurde der Pullover aufgezogen, worauf Wollfäden lauter Knoten zum Vorschein kamen. Die Abstände zwischen den Knoten kodierten die Buchstaben des Alphabets [Schmeh, 2009, S. 18-19].

6.3.2 Open Codes

Open Codes sind, anders als Semagramme, nicht ersichtlich im Trägermedium versteckt und sind somit schwerer zu erkennen. Sie beruhen oft auf einer Absprache zwischen Empfänger und Sender. Bei Open Codes unterscheidet man zwischen maskierten und verschleierten Geheimschriften.

6.3.2.1 Maskierte Geheimschriften

Unter einer maskierten Geheimschrift versteht man einem Code, bei dem harmlos klingende Begriffe eine andere Bedeutung erhalten. Sie werden auch als Jargon-Codes bezeichnet. Diese Codes können mit der Umgangssprache verglichen werden, in der das Wort Kohle beispielsweise ein Synonym für Geld geworden ist. Jargon-Codes verwenden in der Regel jedoch keine allgemeingültigen Begriffe, sondern zwischen Empfänger und Sender ausgemachte Wörter.

Einen Sonderfall bilden Jargon-Codes, welche nur aus einem Wort oder Satz bestehen. Sie können als Alarmsignal, Erkennungszeichen oder Quittung dienen. Ein Beispiel hierfür ist der amerikanische Militärmarsch *The Starts and Stripes Forever*. Dieser wird beispielsweise in Zirkussen gespielt, wenn das Personal auf eine drohende Gefahr aufmerksam gemacht werden soll, ohne sofort das Publikum zu informieren. Somit soll eine Panik verhindert werden [Schmeh, 2009, S.8-15].

6.3.2.2 Verschleierte Geheimschriften

Unter einer verschleierten Geheimschrift hingegen versteht man Methoden, mit denen man unauffällig Informationen in das Trägermedium reincodieren kann. Im Folgenden sollen zwei Beispiele für verschleierte Geheimschriften vorgestellt werden.

Eine einfache Methode besteht darin, jedem Wort eines Textes einen binären Wert zuzuordnen. Wörter deren Länge inklusive Satzzeichen gerade ist, bekommen den Wert 1 zugeordnet. Die restlichen Wörter entsprechend den Wert 0. Jeweils 8 Bit-Werte codieren nun ein Zeichen aus dem ASCII-Code. Das in Abbildung 6.2 skizzierte Beispiel codiert nach dieser Vorschrift das Wort SOS.

Liebe	Kolleginnen!	Wir	genießen	nun	endlich	unsere	Ferien
0	1	0	1	0	0	1	1
auf	dieser	Insel	vor	Spanien.	Wetter	gut,	Unterkunft
0	1	0	0	1	1	1	1
auch,	ebenso	das	Essen.	Toll!	Gruß,	M.	K.
0	1	0	1	0	0	1	1

Abbildung 6.2: Beispiel für eine einfache verschleierte Geheimschrift durch eine binäre Codierung [Nowka, 2004]

Um die Robustheit des Verfahrens zu verbessern, wäre es denkbar, mit einem Schlüssel eine Permutation des ASCII-Codes anzugeben. Dies würde Angriffe verhindern, die lediglich ausprobieren ob eine binäre Codierung des Textes eine sinnvollen Nachricht ergibt. Falls der Angreifer sich jedoch sicher ist, dass eine binäre Codierung verwendetet wurde, kann er versuchen mit einer Häufigkeitstabelle die richtige Permutation zu berechnen, ohne alle Möglichkeiten durchzuprobieren.

Eine weitere Methode könnte eine Nachricht verstecken, indem beispielsweise nur jeweils der erste Buchstabe eines Wortes beachtet wird. Solche Methoden werden auch als Würfel-Methoden bezeichnet, da die Nachricht im Trägermedium verwürfelt ist. In den folgenden Sätzen ist beispielsweise der Satz *Ich bin verliebt* versteckt worden.

Ich, Carl, habe Bier in Nordstadt versteckt.
Ein Richter liest im Einkaufsladen „Bier testen“.

Anstatt jeweils nur das erste Zeichen zu verwenden, könnte man auch jedes dritte Zeichen verwenden, beziehungsweise beliebige andere Kombinationen. Die Möglichkeiten sind hier sehr vielfältig

Steganografische Verfahren die auf Texten basieren weisen allerdings häufig nur eine geringe Kapazität auf. Geht man beispielsweise von einer durschnittlichen Wortlänge von 5 Zeichen pro Wort aus, wobei jedes Zeichen durch 8 Bit codiert ist, dann braucht das vorgestellte binäre Verfahren 40 Bit um ein Bit zu verstecken.

6.4 Steganografie mit digitalen Bildern

Digitale Bilder eignen sich sehr gut um Informationen zu verstecken, da sie aufgrund von Diskretisierungsprozessen oft verrauscht sind. Eine Digitalkamera die ein Bild aufnimmt kann beispielsweise die wirklichen Farben nicht zu 100% darstellen, sondern nähert die Farben nur an. Steganografische Algorithmen die Nachrichten in Bildern verstecken nutzen dies aus, indem sie beispielsweise die Farbtöne geringfügig auf eine Weise ändern, die der Mensch nicht wahrnehmen kann. Zwischen dem originalen und dem veränderten Bild ist somit für das menschliche Auge kein Unterschied zu sehen.

6.4.1 Grundlagen zu digitalen Bildern

Für die Darstellung von Bildern gibt es unzählige Dateiformate wie BMP, GIF oder PNG. Eine sehr ausführliche Liste von Bilddateiformaten findet sich unter Wikipedia [Wikipedia, 2015a].

Auch wenn die unterschiedlichen Formate diverse interne Unterschiede haben, gibt es in der Regel eine Gemeinsamkeit: ein Bild wird in der Regel durch eine Menge von Pixeln definiert, die in einer Matrix abgespeichert sind. Ein Pixel kann, abhängig vom Format, verschiedene Farben annehmen und entspricht einem Bildpunkt des Bildes.

Bei dem BMP Format wird beispielsweise jeder Pixel durch 3 Byte definiert. Jedes Byte codiert dabei eine der Farben rot, grün und blau. Ein Bild der Größe 128×128 hat demnach 16384 Pixel. Insgesamt sind also $16384 \cdot 24$ Bit nötig, um das Bild zu speichern.

6.4.2 Algorithmen

Im Folgenden sollen verschiedene Algorithmen vorgestellt werden, mit denen man Informationen in Bildern speichern kann, die auf einfachen Bildformaten wie BMP anwendbar sind.

6.4.2.1 LSB-Methode

Die LSB-Methode beruht auf der Tatsache, dass zwei Farben, die sich jeweils nur in den niederwertigsten Bits eines jeden Byte unterscheiden, für das menschliche Auge nicht zu unterscheiden sind. Zur Verdeutlichung sind in Abbildung 6.3 zwei Farben im RGB-Format gegenüber gestellt. Die linke Farbe ist durch (30, 144, 255), die rechte durch (31, 145, 254) definiert. Jedes Byte unterscheidet sich also in genau einem Wert, dennoch ist kein Unterschied zu erkennen.

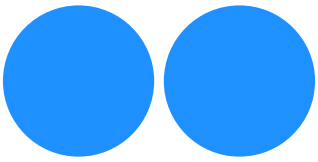

Abbildung 6.3: Vergleich von zwei minimal abweichenden Farben im RGB-Format – links die Farbe (30, 144, 255), rechts die Farbe (31, 145, 254)

Der LSB-Algorithmus nutzt diese Tatsache aus, indem er in jedem Pixel des Bildes, dass jeweils niedrigste Bit überschreibt [Böhme, 2010, S. 40] [Beck, 2014]. Wenn ein Pixel durch drei Byte codiert ist, können also in jedem Pixel 3 Bit versteckt werden. Insgesamt können, unabhängig davon wieviele Bytes ein Pixel codieren, 12,5% des Bildes mit verstecktem Inhalt überschrieben werden.

Beispiel 6.4.1. *Um das folgende Beispiel übersichtlich zu halten, soll davon ausgegangen werden, dass jedes Pixel nur durch ein Byte beschrieben wird. Die in Abbildung 6.4 beschriebene Matrizen, seien die Pixel-Beschreibungen zu einem* 3×3 *Bild. Die linke Matrix sei die Pixelbeschreibung zu dem ursprünglichen Bild. In der rechten Matrix ist mit*

der LSB-Methode die Nachricht 101 101 010 versteckt worden. Die Bits die sich durch das Einfügen der Nachricht geändert haben sind rot makiert. Die grün makierten Bits wurden überschrieben, haben jedoch denselben Wert erhalten.

01011101	11001011	11000101		01011101	11001010	11000101
11011000	10111001	00011011	→	11011001	10111000	00011011
11001101	10110011	11010101		11001100	10110011	11010100

Abbildung 6.4: Pixelbeschreibung eines 3×3 Bildes vor und nach dem Einfügen der Nachricht 101101010 mit dem LSB-Verfahren

Wie man in Abbildung 6.4 sieht, werden mit der LSB-Methode in der Regel nicht alle Bits überschrieben. Statistisch gesehen werden mit der LSB-Methode 50% der niedrigsten Bits verändert.

Zu bemerken ist noch, dass das Verfahren nicht besonders sicher ist. Ein Angreifer kann bei einem verdächtigen Bild die niedrigsten Bits einfach auslesen und auf sinnvollen Inhalt überprüfen.

6.4.2.2 LSB-Methode mit schlüsselgesteuerter Pixelauswahl

Die LSB-Methode mit schlüsselgesteuerter Pixelauswahl funktioniert prinzipiell wie die ursprüngliche LSB-Methode [Beck, 2014]. Allerdings werden nicht alle zur Verfügung stehenden Pixel genutzt, sondern nur eine ausgewählte Teilmenge. Welche Pixel genutzt werden, wird durch einen Schlüssel reguliert. Dieser muss zuvor zwischen Sender und Empfänger ausgetauscht werden. Die Schlüssel bestehen aus Zahlen, welche angeben wie viele Pixel nach dem aktuellen Pixel ausgelassen werden.

Beispiel 6.4.2. *Wir betrachten dasselbe Bild wie im vorherigen Beispiel. Es soll diesmal die Nachricht 00101 mit dem Schlüssel 00211 in dem Bild versteckt werden. Die linke Matrix zeigt wieder das ursprüngliche Bild, die rechte Matrix das modifizierte.*

Anmerkungen zum benutzten Schlüssel: Die erste 0 im Schlüssel gibt an, dass das erste Pixel nicht übersprungen wird, also beschrieben wird. Durch die zweite 0 im Schlüssel wird wieder kein Pixel übersprungen, es wird also das zweite Pixel überschrieben. Da jetzt eine zwei im Schlüssel steht, werden jetzt zwei Pixel ausgelassen. Somit wird erst das 5. Pixel wieder beschrieben.

01011101	11001011	11000101		01011100	11001010	11000101
11011000	10111001	00011011	→	11011000	10111001	00011011
11001101	10110011	11010101		11001100	10110011	11010101

Abbildung 6.5: Pixelbeschreibung eines 3×3 Bildes vor und nach dem Einfügen der Nachricht 00101 mit dem schlüsselgesteuertem LSB-Verfahren und dem Schlüssel 00211

6.4.2.3 LSB-Methode mit Paritätskodierung

Eine weitere Modizifierung der ursprünglichen LSB-Methode, ist die LSB-Methode mit Paritätskodierung. Hier werden jeweils n Pixel in Gruppen zusammengefasst, wobei jede Gruppe ein Geheimbit speichert. Das eingebettete Bit ergibt sich jeweils aus der Parität der niedrigsten Bits der n Gruppenpixel. Der Vorteil bei dieser Methode ist, dass maximal ein Bit pro n Pixel verändert werden muss. Die Änderungsrate wird also umso kleiner, je mehr Pixel zusammengefasst werden. Allerdings sinkt somit auch die Anzahl der Bits die effektiv versteckt werden können. Ein weiterer Vorteil ist, dass von den n Pixeln das jeweils günstigste verändert werden kann, beispielsweise indem man das Pixel ändert, welches sich am meisten von den anderen Pixeln unterscheidet.

Beispiel 6.4.3. *Wir betrachten erneut dasselbe Bild. Es soll diesmal die Nachricht 001 durch die LSB-Methode mit Paritätskodierung versteckt werden. Wir wählen $n = 3$. Die linke Matrix zeigt das ursprüngliche Bild, die rechte Matrix das Bild nach der Modizifierung.*

01011101	11001011	11000101		01011100	11001011	11000101
11011000	10111001	00011011	$\rightarrow$	11011000	10111001	00011011
11001101	10110011	11010101		11001101	10110011	11010101

Abbildung 6.6: Pixelbeschreibung eines 3×3 Bildes vor und nach dem Einfügen der Nachricht 001 mit dem LSB-Verfahren mit Paritätskodierung und dem Parameter $n = 3$

6.5 Digitale Wasserzeichen

Digitale Wasserzeichen können als Teilgebiet der Steganografie verstanden werden, allerdings unterscheidet sich die Zielsetzung von digitalen Wasserzeichen grundlegend mit denen der Steganografie. Die ursprüngliche Steganografie befasst sich hauptsächlich mit der versteckten Nachrichtenübertragung, das heißt es wird hauptsächlich der Aspekt der Vertraulichkeit behandelt. Wasserzeichen hingegen haben die Aufgabe die Echtheit, Originalität oder die Urheberschaft eines Mediums zu garantieren, es geht also darum Authentizität und Integrität von Medien zu garantieren [Dittmann, 2000, S. 16].

Ein sehr bekanntes Beispiel für Wasserzeichen sind die Sicherheitsmarkierungen auf Geldscheinen. Diese dienen dazu, die Echtheit eines Geldscheins zu beweisen und sollen verhindern, dass sich jeder sein Geld selbst druckt.

Digitale Wasserzeichen können für verschiedene Einsatzgebiete verwendet werden, so unterscheidet man Verfahren für die [Dittmann, 2000, S. 30]:

- Urheberidentifizierung
- Kundenidentifizierung
- Annotation von Datenmaterial

- Durchsetzung von Kopierschutz und der Übertragungskontrolle
- Nachweis der Unversehrtheit von Daten

Des Weiteren unterscheidet man in Bezug auf digitale Wasserzeichen unter anderem folgende Kriterien, deren Ausprägungen in der Regel abhängig vom jeweiligen Anwendungsgebiet sind [Dittmann, 2000, S. 31]:

- Wahrnehmungsaspekt
  - wahrnehmbare Wasserzeichen
  - nicht-wahrnehmbare Wasserzeichen
- Robustheit
  - robuste Wasserzeichen
  - fragile, zerbrechliche Wasserzeichen
- Verifizierbarkeit der Markierung
  - geheim
  - öffentlich
- Verwendung des Originals im Abfrageprozess
  - Original benötigt
  - Original nicht benötigt

Konkrete Verfahren werden beispielsweise in *Digital Watermarking* [Kim et al., 2010] und in *Information Hiding* [Filler et al., 2011] beschrieben.

6.6 Spreu-und-Weizen-Algorithmus

Der Spreu-und-Weizen-Algorithmus wurde 1998 von Ronald L. Rivest vorgestellt und stellt eine Alternative zur Kryptografie und zur Steganografie dar [Rivest, 1998]. Die zu übertragende Nachricht wird theoretisch als Klartext übertragen, allerdings wird die richtige Nachricht zwischen ähnlich aussehenden Nachrichten versteckt. Die Schwierigkeit für einen Angreifer besteht darin, die richtigen Nachrichten von den falschen Nachrichten zu unterscheiden. Um die ursprüngliche Nachricht zu erhalten, muss die Spreu vom Weizen getrennt werden.

Das Verfahren lässt sich in drei Bestandteile einteilen. Der erste Schritt besteht darin, die Nachricht in Pakete aufzuteilen, welche einzeln übertragen werden. Damit der Empfänger die Nachricht in der richtigen Reihenfolge wieder zusammen setzen kann, werden die Pakete fortlaufend nummeriert. Im zweiten Schritt werden die Pakete gekennzeichnet. Hierzu wird für jedes Paket eine Prüfsumme berechnet. Die Prüfsumme ist im Prinzip ein Hashwert aus der Seriennummer, dem Paketinhalt und einem zuvor zwischen Sender und Empfänger ausgetauschten Schlüssel. Als letztes werden falsche Nachrichten erzeugt.

Rivest empfiehlt für jede Seriennummer mindestens eine falsche Nachricht zu erzeugen. Diese sollte jeweils ähnlich zu der originalen sein, damit ein Angreifer die falsche Nachricht nicht direkt ausschließen kann. Damit der Empfänger nach der Übertragung die falschen Nachrichten aussortieren kann, erhält jedes Paket mit einer falschen Nachricht eine falsche Prüfsumme.

Beispiel 6.6.1. *Es soll die Nachricht „Kaufe sofort 100 Aktien von der Firma Nike.“ übertragen werden. Um das Beispiel einfach zu halten sei die Prüfsumme zu einem Paket, die Anzahl der Zeichen der Nachricht.*

Zunächst wird die Nachricht in folgende Pakete geteilt:

- *(1, Kaufe, 5)*
- *(2, sofort, 6)*
- *(3, 100 Aktien, 10)*
- *(4, von der Firma Nike, 18)*

Anschließend könnte beispielsweise folgende Spreu erzeugt werden:

- *(1, Verkaufe, 3)*
- *(2, am Ende des Monats, 20)*
- *(3, 2000 Aktien, 4)*
- *(4, von der Firma Adidas, 6)*

Insgesamt werden folgende Pakete übertragen:

- *(1, Kaufe, 5)*
- *(1, Verkaufe, 3)*
- *(2, sofort, 6)*
- *(2, am Ende des Monats, 20)*
- *(3, 100 Aktien, 10)*
- *(3, 2000 Aktien, 4)*
- *(4, von der Firma Nike, 18)*
- *(4, von der Firma Adidas, 6)*

Um mit dem Verfahren eine höhere Sicherheit zu erreichen, empfiehlt Rivest die Übertragung von einem Bit pro Paket. Das jeweils zugehörige falsche Paket, soll dann das andere Bit erhalten. Allerdings steigt bei diesem Ansatz die Anzahl der Pakete sehr stark an, sodass es nur für kleine Nachrichten anwendbar ist.

6.7 Vergleich mit der Kryptografie

Um die Vertraulichkeit einer Nachricht zu gewährleisten kann neben der Kryptografie auch die Steganografie verwendet werden. Dies bietet sich insbesondere dann an, wenn der Nachrichtenaustausch nicht bemerkt werden soll oder die Möglichkeiten der Kryptografie nicht zur Verfügung stehen. Insbesondere im Falle der Umsetzung des in der Politik diskutierten Umsetzung eines Kryptografie Verbots [DUB, 2015], stünde mit der Steganografie bereits eine wirkungsvolle Alternative bereit.

Für sehr sicherheitskritische Situationen bildet die Steganografie jedoch keine Alternative, sondern vielmehr eine *Ergänzung* zur Kryptografie. Die zu übertragenden Daten können erst mit einem modernen Verschlüsselungsalgorithmus verschlüsselt werden, um anschließend in ein Trägermedium eingebettet zu werden [BÖHME, 2010, S. 51]. Im besten Falle ist dieser Geheimtext nicht zu entdecken, da er sich nicht von zufälligen Rauschen unterscheidet. Im schlechtesten Fall muss ein Angreifer zwei Sicherheitsmechanismen knacken. Durch Kombination von Kryptografie und Steganografie kann die Sicherheit der Nachrichtenübertragung also stark verbessert werden. Zudem sollten zwei unabhängige Schlüssel ausgetauscht werden, womit die Sicherheit noch weiter erhöht wird.

7 Fazit

„Is cryptography dead?“ Mit dieser provokanten Frage beginnt das unter anderem von Daniel J. Bernstein stammende Buch *Post-Quantum Cryptography* [Bernstein et al., 2008]. Gemeint ist die Angst davor, dass in der Zukunft Quantencomputer jede Verschlüsselung brechen können. In der Tat geht er davon aus, dass viele wichtige Algorithmen wie RSA oder Diffie-Hellman durch Quantencomputer in der Zukunft gebrochen werden.

Diese Gefahr besteht allerdings hauptsächlich für asymmetrische Algorithmen. Bisher sind, in Bezug auf die Kryptografie, lediglich effiziente Quantenalgorithmen bekannt, die verschiedene mathematische Probleme lösen können. Der bekannteste Algorithmus ist der Shor-Algorithmus, welcher die Primfaktorzerlegung einer Zahl in Polynomialzeit berechnen kann [Shor, 1997] und somit eine große Gefahr für RSA darstellt. Von neuen komplexen Angriffen durch Quantencomputer auf symmetrische Verfahren wie AES, geht er jedoch nicht aus.

Weil das Problem des Schlüsselaustausches symmetrischer Verfahren jedoch in der Regel durch asymmetrische Verfahren gelöst wird, besteht hier ein erheblicher Forschungsbedarf, um die heutzutage in der Regel verwendeten Algorithmen wie RSA, durch Algorithmen die gegen Angriffe durch Quantencomputer sicher sind, abzulösen. Aktuelle Verfahren an denen derzeit geforscht werden sind *Code-based cryptography*, *Lattice-based cryptography* sowie *Multivariate-quadratic-equations cryptography* [Bernstein et al., 2008, S. 1].

Letztendlich bleibt zwar die Frage, ob die Quantencomputer jemals leistungsfähig genug sein werden, um die Verschlüsselung wirklich zu gefährden, dennoch empfiehlt Bernstein, sich jetzt auf die Suche nach adäquaten Ersatz zu machen, damit im Fall der Fälle sichere Kryptoalgorithmen bereit stehen.

Ein anderes großes Problem der Kryptografie sowie der Steganografie ist der Mensch an sich. Die Kryptografie und die Steganografie bieten viele verschiedene Verfahren, Protokolle und Algorithmen, die bei richtiger Verwendung, durchaus in der Lage sind, die verschiedenen Sicherheitsziele um Vertraulichkeit zu garantieren. Allerdings werden diese häufig nicht oder nicht vernünftig eingesetzt, sei es aus Bequemlichkeit oder weil man es nicht besser weiß, man denke nur an Abbildung 5.1 über Passwörter.

Hier besteht meiner Meinung nach ein erheblicher Bedarf an Aufklärung, damit die Menschen bewusster mit den Möglichkeiten der Kryptografie, wie beispielsweise der Passworterstellung, umgehen. Ein weiterer Ansatz könnte sein, die Usability kryptografischer Methoden zu erhöhen, um die Hürden der Verwendung von Kryptografie zu verringern, beispielsweise indem Emails automatisch verschlüsselt werden. Des Weiteren ist es denkbar vermehrt Constraints einzusetzen, die beispielsweise vorgeben, dass ein Passwort aus Zahlen sowie Klein- und Großbuchstaben bestehen. Hierdurch könnte immerhin die Verwendung von sehr schwachen Passwörtern verhindert werden.

Literaturverzeichnis

[Anderson et al., 1998] Anderson, Ross, E. Biham und L. Knudsen (1998). *Serpent: A Proposal for the Advanced Encryption Standard*.

[Arora, 2012] Arora, Mohit (2012). *How secure is AES against brute force attacks?*. `http://www.eetimes.com/document.asp?doc_id=1279619`, abgerufen am 23.08.15.

[Beck, 2014] Beck, Martin (2014). *Grundlagen der Steganographie*.

[Bernstein, 2005] Bernstein, Daniel J. (2005). *The Salsa20 family of stream ciphers*.

[Bernstein et al., 2008] Bernstein, Daniel J., J. Buchmann und E. Dahmen, Hrsg. (2008). *Post-Quantum Cryptography*. Springer-Verlag Berlin Heidelberg.

[Beutelspacher et al., 2010] Beutelspacher, Albrecht, H. Neumann und T. Schwarzpaul (2010). *Kryptografie in Theorie und Praxis*. Vieweg + Teubner.

[Beutelspacher et al., 2006] Beutelspacher, Albrecht, J. Schwenk und K.-D. Wolfenstetter (2006). *Moderne Verfahren der Kryptographie*. Vieweg + Teubner.

[Blizzard, 2015] Blizzard (2015). *Battle.net Authenticator*. `https://eu.battle.net/support/de/article/authenticator-faq`, abgerufen am 23.08.15.

[Bube, 2014] Bube, Lars (2014). *Die beliebtesten Passwörter 2013*. `http://www.crn.de/security/artikel-101738-2.html`, abgerufen am 23.08.2015.

[Buchmann, 2010] Buchmann, Johannes (2010). *Einführung in die Kryptographie*. Springer-Verlag Berlin Heidelberg.

[Bundesamt für Sicherheit in der Informationstechnik, 2015] Bundesamt für Sicherheit in der Informationstechnik (2015). *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*.

[Bundesministerium des Innern, 2015] Bundesministerium des Innern (2015). *Die elektronischen Funktionen des Personalausweises*. `http://www.personalausweisportal.de/DE/Buergerinnen-und-Buerger/Der-Personalausweis/Funktionen/funktionen_node.html`, abgerufen am 24.08.2015.

[Böhme, 2010] Böhme, Rainer (2010). *Advanced Statistical Steganalysis*. Springer-Verlag Berlin Heidelberg.

[Daemen und Rijmen, 2002] Daemen, Joan und V. Rijmen (2002). *The Design of Rijndael*. Springer-Verlag Berlin Heidelberg.

[Dittmann, 2000] Dittmann, Jana (2000). *Digitale Wasserzeichen*. Springer.

[dub, 2015] dub (2015). *EU-Innenminister diskutieren über ein Ende der vertraulichen Kommunikation*. `https://www.unwatched.org/20150121_EU-Innenminister_diskutieren_ueber_ein_Ende_der_vertraulichen_Kommunikation`, abgerufen am 24.08.15.

[Ertel, 2012] Ertel, Wolfgang (2012). *Angewandte Kryptographie*. Carl Hanser Verlag München.

[Fiat und Shamir, 1986] Fiat, Amos und A. Shamir (1986). *How To Prove Yourself: Practical Solutions to Identification and Signature Problems*.

[Filler et al., 2011] Filler, Tomáš, T. Pevný, S. Craver und A. Ker (2011). *Information Hiding*. Springer.

[Forster, 2015] Forster, Otto (2015). *Algorithmische Zahlentheorie*. Springer Spektrum.

[Freiermuth et al., 2014] Freiermuth, Karin, J. Hromkovič, L. Keller und B. Steffen (2014). *Einführung in die Kryptologie*. Springer Vieweg.

[Gabele et al., 1991] Gabele, Eduard, M. Kroll und W. Kreft (1991). *Kommunikation in Rechnernetzen*. Springer-Verlag Berlin Heidelberg.

[Hochschule Augsburg, 2015] Hochschule Augsburg (2015). *McEliece-Kryptosystem Beispieldurchlauf*. `http://glossar.hs-augsburg.de/McEliece-Kryptosystem_Beispieldurchlauf`, abgerufen am 23.08.15.

[Homeister, 2005] Homeister, Matthias (2005). *Quantum Computing verstehen: Grundlagen – Anwendungen – Perspektiven*. Vieweg + Teubner.

[Javidi, 2005] Javidi, Bahram, Hrsg. (2005). *Optical and Digital Techniques for Information Security*. Springer.

[Jochemsz, 2002] Jochemsz, Ellen (2002). *Goppa Codes & the McEliece Cryptosystem*. Doktorarbeit, Universiteit van Amsterdam.

[Johnson et al., 2001] Johnson, Neil F., Z. Duric und S. Jajodia (2001). *INFORMATION HIDING: Steganography and Watermarking – Attacks and Countermeasures*. Springer Science+Business Media New York.

[Kahn, 1967] Kahn, David (1967). *The Codebreakers – The Story of Secret Writing*. The Macmillan Company.

[Karapanos et al., 2015] Karapanos, Nikolaos, C. Marforio, C. Soriente und S. ˇ Capkun (2015). *Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound*.

[Karpfinger und Kiechle, 2009] Karpfinger, Christian und H. Kiechle (2009). *Kryptologie Algebraische Methoden und Algorithmen*. Vieweg + Teubner.

[Kerckhoffs, 1883] Kerckhoffs, Auguste (1883). *La cryptographie militaire*. Journal des sciences militaires.

[Kim et al., 2010] Kim, Hyoung-Joong, Y.-Q. Shi und M. Barni (2010). *Digital Watermarking*. Springer.

[Küsters und Wilke, 2011] Küsters, Ralf und T. Wilke (2011). *Moderne Kryptographie*. Vieweg + Teubner.

[McEliece, 1978] McEliece, Robert J. (1978). *A Public-Key Cryptosystem Based on Algebraic Coding Theory*.

[Niebuhr et al., 2012] Niebuhr, Robert, M. Meziani, S. Bulygin und J. Buchmann (2012). *Selecting parameters for secure McEliece-based cryptosystems*. International Journal of Information Security.

[Nowka, 2004] Nowka, Jörg (2004). *Steganographie – Das Verstecken von Informationen*.

[Rivest, 1998] Rivest, Ronald L. (1998). *Chaffing and Winnowing: Confidentiality without Encryption*.

[Rivest et al., 1978] Rivest, Ronald L., A. Shamir und L. Adleman (1978). *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*.

[Schmeh, 2009] Schmeh, Klaus (2009). *Versteckte Botschaften (TELEPOLIS): Die faszinierende Geschichte der Steganografie*. dpunkt Verlag.

[Schmeh, 2013] Schmeh, Klaus (2013). *Kryptografie : Verfahren, Protokolle, Infrastrukturen*. Heidelberg : dpunkt-Verl.

[Schneier et al., 1998] Schneier, Bruce, J. Kelsey, D. Whiting, D. Wagner, C. Hall und N. Ferguson (1998). *Twofish: A 128-Bit Block Cipher*.

[Schwenk, 2014] Schwenk, Jörg (2014). *Sicherheit und Kryptographie im Internet*. Springer Vieweg.

[Schönfeld et al., 2012] Schönfeld, Dagmar, H. Klimant und R. Piotraschke (2012). *Informations- und Kodierungstheorie*. Vieweg + Teubner.

[Shannon, 1949] Shannon, Claude (1949). *Communication Theory of Secrecy Systems*. Bell System Technical Journal.

[Shor, 1997] Shor, Peter W. (1997). *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*.

[Singh, 2004] Singh, Simon (2004). *Codes: Die Kunst der Verschlüsselung. Geschichte – Geheimnisse – Tricks*. Deutscher Taschenbuch Verlag (1. März 2004).

[Steger, 2007] Steger, Angelika (2007). *Diskrete Strukturen*. Springer.

[Swoboda et al., 2008] Swoboda, Joachim, S. Spitz und M. Pramateftakis (2008). *Kryptographie und IT-Sicherheit*. Vieweg + Teubner.

[Weck, 2014] Weck, Andreas (2014). *Jede Hürde zählt: So setzt du die Zwei-Faktor-Authentifizierung bei großen Webseiten ein*. `http://t3n.de/news/zwei-faktor-authentifizierung-google-facebook-dropbox-paypal-twitter-531483/`, abgerufen am 23.08.15.

[Werner, 2002] Werner, Annette (2002). *Elliptische Kurven in der Kryptographie*. Springer.

[Wikipedia, 2015a] Wikipedia (2015a). *Grafikformat*. `https://de.wikipedia.org/wiki/Grafikformat`, abgerufen am 23.08.15.

[Wikipedia, 2015b] Wikipedia (2015b). *SecurID*. `https://de.wikipedia.org/wiki/SecurID`, abgerufen am 23.08.15.