HUMBOLDT-UNIVERSITÄT ZU BERLIN

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Diplominformatiker

KRYPTOGRAFIE

unter begrenzten Ressourcen

AUF MIKROCONTROLLERN

✧

eingereicht von

ALEXANDER BORISOV

geboren am 28.09.1984 in Petrozavodsk, Russland

1. Gutachter: Prof. Dr. Johannes Köbler
2. Gutachter: Prof. Dr. Ernst-Günter Giessmann

eingereicht am 28. Januar 2013

Lehrstuhl für Komplexität und Kryptografie
Institut für Informatik
Mathematisch-Naturwissenschaftliche Fakultät II
Humboldt-Universität zu Berlin

Berlin
Januar 2013

Alexander Borisov
Sewanstr. 173
10319 Berlin

Geboren am 28. September 1984 in Petrozavodsk, Russland

E-Mail: mail@aborisov.de

Eingereicht am: 28. Januar 2013
Verteidigt am: 19. April 2013

Humboldt-Universität zu Berlin
Präsident der Universität: Prof. Dr. Jan-Hendrik Olbertz

Diese Arbeit wurde in LaTeX gesetzt.

Als LaTeX Template wurde die Vorlage von Uggedal [52] und Borisov [5] verwendet.

Hauptschrift: Linux Libertine von Libertine Open Fonts Projekt;
Mathematische Schrift: AMS Euler von Hermann Zapf;
Kegelgröße im Satzspiegel: 12 pt bei 0.3515 mm.

Kompilierungsmethode: pdfTeX – 1.40.12
Datum der letzten Kompilierung: 24. April 2013

Ulrich Tholuck
gewidmet

KURZFASSUNG

Der Einsatz von kryptografischen Methoden auf Mikrocontrollern erfordert eine genaue Untersuchung und Auswahl der einzusetzenden Verfahren. Dabei stehen zahlreiche technologische Beschränkungen der Mikrocontroller wie eine geringe Ausführungsgeschwindigkeit, sehr begrenzter Speicherplatz und oft knappe Energieressourcen im Widerspruch zu den hohen Sicherheitsanforderungen von bestimmten Anwendungen.

Das Ziel der Arbeit ist es, eine Auswahl an bekannten Verschlüsselungsverfahren zu analysieren und zu bestimmen, inwiefern diese für den Einsatz auf Mikrocontrollern geeignet sind. Dabei werden in einer empirischen Untersuchung mehrere Verschlüsselungsverfahren getestet und miteinander verglichen. Anschließend erfolgt eine beispielhafte Umsetzung der gewünschten Sicherheitsanforderungen. Die Untersuchungen werden an einem 8-Bit Mikrocontroller und an einem 32-Bit Mikrocontroller durchgeführt.

Stichwörter: Mikrocontroller, Blockalgorithmen, AES, DES, XTEA, PRESENT, RSA, federleichte Kryptografie.

ABSTRACT

The application of cryptographic methods on microcontrollers requires close examination and selection of the applied methods. Yet, the numerous technological constraints of the microcontrollers such as the long execution time, the limited memory capacity and the scarce energy resources come into conflict with the high security requirements of certain applications.

The aim of this thesis is to analyze some well-known encryption algorithms in order to define which are the most suitable for the application on the microcontrollers. In the empirical work, these encryption algorithms are tested and compared with each other. The selected algorithms are then implemented on the microcontrollers. The empirical experiments are conducted on a 8-bit and on a 32-bit microcontrollers.

Keywords: microcontrollers, block algorithms, AES, DES, XTEA, PRESENT, RSA, lightweight cryptography.

VORWORT

Diese Diplomarbeit wurde am Lehrstuhl für Komplexität und Kryptografie an der Humboldt-Universität zu Berlin und in Zusammenarbeit mit der iSAtech GmbH verfasst.

Ich möchte vor allem meinen zwei Gutachtern Prof. Dr. Johannes Köbler und Prof. Dr. Ernst-Günter Giessmann danken, die mich hervorragend während der Erstellung der Arbeit mit zahlreichen Anregungen unterstützt haben. Zudem gilt mein Dank meiner Frau für die sorgfältige Prüfung der Arbeit sowie für die wertvollen Hinweise und die kritischen Fragen. Ebenfalls möchte ich mich sehr herzlich bei dem ganzen Team der iSAtech GmbH und insbesondere bei Emanuel Mey und Pawel Sawicki für die ausgezeichnete Unterstützung und die Bereitstellung der für die Untersuchungen notwendigen Mikrocontrollern bedanken. Ich danke auch Ljuba Klassen, Anna Jakovleva, Regine Dugas, Alexander Decker und Karl Christ für das Korrekturlesen der Arbeit ganz herzlich.

Alexander Borisov
Berlin
Januar 2013

INHALTSVERZEICHNIS

EINLEITUNG

1

> »There are two types of encryption: one that will prevent your sister from reading your diary and one that will prevent your government.«
>
> Bruce Schneier, zit. nach [55, S. 261].

Die Kryptografie nimmt in der modernen Gesellschaft eine immer wichtiger werdende Stellung ein. Wurden noch vor einigen Jahren kryptografische Methoden und Protokolle vor allem von Militär, Regierungen oder speziellen Industriezweigen eingesetzt, so findet inzwischen eine stärkere Ausweitung der Kryptografie auch im privaten Bereich statt. Durch die Einführung und Verbreitung von beispielsweise Chipkarten, bargeldlosen Bezahlsystemen, Online-Banking, WLAN oder Bezahlterminals sind kryptografische Methoden allgegenwärtig geworden.

Durch die Entwicklung von effizienten und gleichzeitig starken Verschlüsselungsverfahren und der parallelen Erhöhung der Rechenleistung ist es nun möglich geworden, moderne kryptografische Verfahren auch auf kleinen Recheneinheiten, auf Mikrocontrollern und auf Chipkarten einzusetzen. Die Anzahl der möglichen Szenarien für den Einsatz von kryptografischen Methoden ist dadurch abermals gestiegen. Die vorliegende Diplomarbeit befasst sich mit der Verwendung von kryptografischen Methoden auf Mikrocontrollern unter begrenzten Ressourcen. Im Folgenden wird die Zielsetzung der Arbeit formuliert und ihr Aufbau vorgestellt.

1.1 ZIELSETZUNG DER ARBEIT

Der Einsatz von kryptografischen Algorithmen auf Mikrocontrollern bedarf einer genauen Auswahl der einzusetzenden Verfahren. Primär sind die beschränkten Ressourcen, insbesondere eine geringe

Ausführungsgeschwindigkeit, kleine Größe des Speichers und keine permanente Energieversorgung, die wichtigsten Randbedingungen für den Einsatz von Algorithmen. Das Ziel der Arbeit ist zu prüfen, inwiefern gängige Verschlüsselungsverfahren für den Einsatz auf Mikrocontrollern geeignet sind. Um dieses Ziel zu erreichen werden die ausgewählten Verschlüsselungsverfahren vorgestellt, analysiert und miteinander verglichen. Dabei wird ein besonderer Fokus auf Blockalgorithmen gelegt. Ein Schwerpunkt der Arbeit ist die Durchführung einer empirischen Untersuchung an zwei unterschiedlichen Mikrocontrollern.

Durch die empirische Untersuchung und die anschließende Analyse werden die Unterschiede zwischen den Algorithmen sichtbar. Dies erlaubt in einem weiteren Schritt diejenigen Algorithmen auszuwählen, die für den Einsatz unter gegebenen beschränkten Ressourcen die aussichtsreichsten Resultate gezeigt haben. Es wird eine Empfehlung für den Einsatz von bestimmten Algorithmen gegeben. In einem weiteren Schritt werden die gewünschten Sicherheitsanforderungen mit den empfohlenen Algorithmen beispielhaft umgesetzt.

1.2 AUFBAU DER ARBEIT

Die vorliegende Diplomarbeit besteht aus fünf Kapiteln. Nach einer Einleitung findet im zweiten Kapitel eine begriffliche Einordnung sowie ein kurzer geschichtlicher Rückblick statt. Es werden die zentralen Begriffe der Kryptologie vorgestellt, die in den folgenden Kapiteln Verwendung finden. Die Beschreibung der am häufigsten verwendeten Konstruktionen für den Aufbau der Blockalgorithmen liefert wichtige Erkenntnisse für die darauf folgende Darstellung der Algorithmen und die empirische Untersuchung.

Im dritten Kapitel findet zuerst die Darstellung der ausgewählten Plattform statt. Dabei werden die zwei bei den empirischen Untersuchungen verwendeten 8-Bit und 32-Bit Mikrocontroller vorgestellt. Anschließend folgt eine ausführliche Darstellung der ausgewählten Algorithmen. Dabei werden sowohl ältere Algorithmen als auch Algorithmen, die den aktuellen Wissenstand repräsentieren, untersucht. Im Anschluss wird der Versuchsaufbau für die empirische Untersuchung beschrieben. In einem folgenden Schritt werden die Resultate

der Untersuchung vorgestellt und visualisiert. Das Kapitel endet mit einer Zusammenfassung der wichtigsten Ergebnisse.

Im vierten Kapitel wird zunächst auf die Beschreibung der ausgewählten Sicherheitsanforderungen eingegangen. Anschließend folgt die Umsetzung dieser Anforderungen in einer beispielhaften Implementierung. Die Umsetzung von den zentralen kryptografischen Anforderungen wie Verschlüsselung, Prüfung der Integrität und Authentifizierung wird unter der Verwendung von analysierten Algorithmen umgesetzt.

Das fünfte Kapitel gibt einen Ausblick auf Themen, die im Rahmen von weiteren Forschungsaktivitäten untersucht werden können. Die Arbeit endet mit einer Zusammenfassung der wichtigsten Erkenntnisse und Schlussfolgerungen.

EINFÜHRUNG IN DIE KRYPTOGRAFISCHEN PRIMITIVE

2

»Cryptography is about communication in the presence of an adversary.«

Ronald L. Rivest [41].

Um eine genaue Untersuchung der im nächsten Kapitel vorgestellten Algorithmen und Konzepte zu ermöglichen, werden im Folgenden die wichtigsten Ziele und Möglichkeiten der modernen Kryptografie vorgestellt. Zuerst wird in dem Kapitel ein grober Überblick über die Kryptologie gegeben. Anschließend findet ein geschichtlicher Rückblick statt. Nach der Betrachtung der Angriffsszenarien werden die wichtigsten Ideen und Konzepte der symmetrischen und der asymmetrischen Kryptografie dargelegt. Das Ziel dieses Kapitels ist primär, die in den folgenden Kapiteln der Arbeit verwendeten Begriffe darzustellen und einzuordnen.

2.1 KURZE EINFÜHRUNG IN DIE WELT DER KRYPTOLOGIE

Die Kryptologie als moderne Wissenschaft besteht aus zwei großen Bereichen: der *Kryptografie* und der *Kryptoanalyse*.[1] Die Kryptografie befasst sich mit der Entwicklung von neuen und der Verbesserung von bestehenden kryptografischen Algorithmen. Die Kryptoanalyse beschäftigt sich dagegen mit den Schwachstellen der verwendeten kryptografischen Algorithmen und den Ansätzen wie diese ausgenutzt werden können. Beide Bereiche sind eng miteinander verknüpft,

1. Diese Unterteilung wurde zum ersten Mal vom amerikanischen Kryptologen William Friedman (1891 – 1969) vorgeschlagen; vgl. Salomon [42, S. 90].

da für die Entwicklung von neuen kryptografischen Algorithmen die Techniken der Kryptoanalyse angewendet werden müssen. Umgekehrt gilt für die Kryptoanalyse, dass beispielsweise das Wissen, wie die Algorithmen intern aufgebaut sind, für die Auswahl der in Frage kommenden Techniken der Kryptoanalyse verwendet wird. Im Folgenden werden die wichtigsten Begriffe der Kryptologie kurz vorgestellt.

2.1.1 *Klärung der Begriffe*

Ein kryptografischer Prozess ist eine Interaktion zwischen zwei oder mehreren Parteien,[2] die mit Hilfe von speziellen kryptografischen Algorithmen bestimmte Sicherheitsanforderungen zu erfüllen versuchen. Dabei sind die Sicherheitsanforderungen in der Regel durch die folgenden vier Ziele definiert:[3]

2. Streng gesehen kann auch nur eine Partei beteiligt sein, dann gibt es eine zeitliche Differenz zwischen den Aktionen.

3. Vgl. Paar u. Pelzl [37, S. 263 f.].

1. **Vertraulichkeit**: Alle nicht autorisierten Teilnehmer erhalten keine Informationen über den Inhalt der ausgetauschten Daten.
2. **Integrität**: Eine Modifikation der ausgetauschten Daten darf nicht unbemerkt durchführbar sein.
3. **Authentifizierung**: Der Sender der ausgetauschten Daten muss eindeutig und fälschungssicher bestimmbar sein.
4. **Verbindlichkeit**: Der Sender kann seine Autorenschaft nach getätigter Kommunikation nicht mehr abstreiten.

Zu den oberen vier zentralen Zielen werden in komplexeren kryptografischen Systemen und Prozessen zusätzlich folgende Ziele verfolgt:[4]

4. Nicht alle der folgenden Zielen werden ausschließlich durch kryptografische Mittel gelöst. Die Zutrittskontrolle kann zum Beispiel durch gesicherte Türen oder Ähnliches realisiert werden.

1. **Zutrittskontrolle**: Es muss eine eindeutige Identifikation der Benutzer von bestimmten Ressourcen realisiert werden.
2. **Verfügbarkeit**: Die Sicherstellung einer unterbrechungsfreien Verfügbarkeit von kritischen Ressourcen.
3. **Protokollierung**: Jegliche Veränderungen in kritischen Systemen müssen fälschungssicher protokolliert werden.
4. **Anonymität**: Bei einigen Anwendungen, beispielsweise elektronischen Wahlen, darf die Identität der Benutzer nicht offenbart werden.

Mathematisch gesehen ist ein kryptografisches System ein Tupel, bestehend aus den folgenden fünf Elementen: $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, wobei diese wie folgt definiert werden: [5]

5. Vgl. Stinson [49, S. 1 f.].

$\mathcal{P}$ — Menge der möglichen Klartexte;

$\mathcal{C}$ — Menge der möglichen Kryptotexte;

$\mathcal{K}$ — Menge der möglichen Schlüssel;

$\mathcal{E}$ — Menge der möglichen Verschlüsselungsfunktionen;

$\mathcal{D}$ — Menge der möglichen Entschlüsselungsfunktionen.

Zudem gilt:

$$\forall\, k \in \mathcal{K} \text{ und } \forall\, e_k \in \mathcal{E} \text{ gilt } e_k : \mathcal{P} \to \mathcal{C} \tag{2.1}$$

$$\forall\, k' \in \mathcal{K} \text{ und } \forall\, d_{k'} \in \mathcal{D} \text{ gilt } d_{k'} : \mathcal{C} \to \mathcal{P} \tag{2.2}$$

$$\text{und } \forall\, p \in \mathcal{P} \text{ gilt } d_{k'}(e_k(p)) = p \tag{2.3}$$

Die Funktion (2.1) heißt *Verschlüsselung* und erzeugt aus einem Klartext den Kryptotext. Dagegen überführt die Funktion (2.2) Kryptotexte wieder zurück in die Klartexte und heißt demnach – *Entschlüsselung*. Die Bedingung (2.3) ist dabei besonders wichtig und besagt, dass für alle Klartexte die Verschlüsselung und die anschließende Entschlüsselung wieder den Klartext liefert. Dabei gilt für symmetrische Kryptosysteme, dass $k' = k$ ist. Zudem gilt, dass wenn $\mathcal{P} = \mathcal{C}$, dann gilt, dass $\forall\, e_k \in \mathcal{K}$ eine Permutation ist.

Ein *kryptografisches Primitiv* ist eine grundlegende kryptografische Funktion, die ein Element eines komplexeren kryptografischen Protokolls bzw. Prozesses sein kann. Beispiele für derartige Funktionen sind Blockalgorithmen, Zufallsgeneratoren oder Hashfunktionen. Gewöhnlich werden mehrere Primitive Bestandteile eines Protokolls. Um die komplexen Prozesse in der Kryptografie beschreiben zu können, werden die jeweiligen Rollen in den Prozessen bzw. Protokollen entsprechend bezeichnet. In der vorliegenden Arbeit werden die für die jeweiligen Rollen typischen Bezeichnung vorgenommen.[6]

6. Typische Bezeichnungen der Akteure bei den kryptografischen Protokollen:
Alice – Initiator des Prozesses;
Bob – Partner von Alice;
Eve – passiver Angreifer;
Oscar – aktiver Angreifer;
Trent – vertrauenswürdige Instanz.

2.1.2 Geschichtlicher Rückblick

Der Begriff Kryptografie, aus dem Griechischen von *kryptós* – verborgen und *gráphein* – schreiben, wurde im 17. Jahrhundert von John Wil-

kins[7] eingeführt. Schon lange zuvor, etwa seit der Zeit als Menschen anfingen Informationen schriftlich festzuhalten und auszutauschen, gab es den Wunsch bestimmte Informationen nur einem speziellen Personenkreis zugängig zu machen und somit vor anderen zu verbergen. Erste Funde kryptografisch verschlüsselter Texte wurden in Ägypten gemacht und sind mit etwa 2.000 Jahren v. u. Z. datiert. Als Verschlüsselungstechnik wurden dabei primär spezielle neu erfundene Zeichen für die allgemein verwendete Hieroglyphenschrift benutzt.[8] Es kann daher davon ausgegangen werden, dass die Geschichte der Kryptografie mindestens 4000 Jahre alt ist. Im 5. Jahrhundert v. u. Z. erwähnt der griechische Historiker Herodot erstmals geheime Texte und Methoden ihrer Übermittlung in militärischem Zusammenhang. Den Sieg über die Perser verdankten die Griechen laut Herodot zu einem Großteil der Kryptografie.[9]

7. John Wilkins (1614 – 1672) – Philosoph und Gründungsmitglied der Royal Society.

8. Vgl. Kahn [21, S. 71 ff.].

9. Vgl. Singh [48, S. 18].

Grundsätzlich kann die Geschichte der Kryptografie zeitlich grob in drei Bereiche unterteilt werden:

1. antike und mittelalterliche Kryptografie;
2. Mechanisierung der Kryptografie im frühen 20. Jahrhundert;
3. moderne digitale Kryptografie.

In der antiken und mittelalterlichen Kryptografie erfolgte die Ver- und Entschlüsselung von Texten ausschließlich manuell und erforderte somit einen hohen personellen sowie administrativen Aufwand. Die Verwendung von Kryptografie war deshalb einem kleinen Kreis von Herrschern, Militärangehörigen und Vertretern der Kirche vorbehalten. Zu den ältesten kryptografischen Verfahren gehörten einfache Substitutionen und Transpositionen der Texte. Bei der Substitution werden Buchstaben der Nachricht durch jeweils andere Buchstaben oder Zeichen ersetzt. Bei der Transposition werden die Buchstaben der Nachricht anders angeordnet und vertauscht. Eine der ersten Formen der Transposition stellt die seit dem 5. Jahrhundert verwendete *Skytale* dar. Die Skytale ist ein Holzstab von einem festgelegten Durchmesser. Ein Papierstreifen wird um den Stab gewickelt und die Nachricht wird entlang des Stabes aufgeschrieben. Wird nun der Stab entfernt, scheint der Papierstreifen nur eine sinnlose Buchstabenfolge zu enthalten. Der Schlüssel ist in dem Fall der Durchmesser des Stabes.[10]

10. Vgl. Singh [48, S. 23 f.].

Eine sehr bekannte Substitution ist die nach Julius Caesar benannte Caesar-Verschlüsselung. Dabei wird jedes Zeichen eines Klartextes mit einem anderen ersetzt, welches dem Zeichen um drei Zeichen im Alphabet folgt.[11] Diese antiken Verschlüsselungstechniken bieten aus heutiger Sicht keine zufriedenstellende Sicherheit und sind mit einfachen Methoden der Kryptoanalyse zu entschlüsseln.

Jedoch wurden bereits im antiken Griechenland Techniken entwickelt, die zum Teil auch in der modernen Kryptografie Verwendung finden. Der Geschichtsschreiber Polybius entwickelte beispielsweise ein Umrechnungssystem, welches es ermöglichte, Buchstaben in Paare von Ziffern zu überführen.[12] Die Buchstaben des Klartextalphabets werden dabei in einer Matrix angeordnet, wobei die Zeilen und Spalten jeweils durchnummeriert werden.[13] Ein Buchstabe wird demnach durch die jeweilige Zeilen- und Spaltennummer der Matrix repräsentiert. Der Buchstabe m ist folglich definiert durch $m = 32$ und beispielsweise x durch $x = 53$.

Im frühen 20 Jahrhundert wurde mit der Einführung von mechanischen Verschlüsselungsmaschinen wie Kryha, Hagelin und später Enigma und Lorenz eine neue Epoche angestoßen.[14] Sowohl die Geschwindigkeit als auch die Fehlertoleranz erhöhten sich dadurch stark. Da der eigentliche kryptografische Algorithmus von der Maschine realisiert wurde, war für ihre Bedienung keine spezielle Ausbildung mehr nötig, wie sie die früheren, manuellen Verfahren noch erforderlich machten. Eine weitere große Veränderung erlebte die Kryptografie mit der Einführung von Computern und der digitalen Informationsverarbeitung. Die Geschwindigkeit der Verarbeitung sowie die allgemeine kryptografische Sicherheit der verwendeten Algorithmen stieg erneut extrem und förderte die Verbreitung der kryptografischen Methoden in breitere Bevölkerungsschichten. Die Entwicklung neuer Standards und Methoden wie die asymmetrische Kryptografie führten letztendlich dazu, dass die Kryptografie zu einem eigenständigen Forschungsgebiet wurde.

Wie bereits gesehen, wurden in den letzten Jahrhunderten zahlreiche Techniken der Kryptografie entwickelt. Grundsätzlich wird dabei unter Verwendung eines geheimen Schlüssels ein Algorithmus auf die Nachricht angewendet, sodass es Unbefugten im Optimalfall nicht möglich ist, die Nachricht zu lesen. Der Sinn der Nachricht ist somit verborgen und die eigentliche Übermittlung der Nachricht kann über

11. Caesar-Verschlüsselung; vgl. Kahn [21, S. 83 ff.]:
`a` → `d`
`b` → `e`
`c` → `f`
`caesar` → `fdhvdu`.

12. Vgl. Kahn [21, S. 83].

13. Polybius-Matrix:

	1	2	3	4	5
1	a	b	c	d	e
2	f	g	h	ij	k
3	l	m	n	o	p
4	q	r	s	t	u
5	v	w	x	y	z

14. Genaue Informationen zu der Geschichte der Kryptografie und den jeweiligen Maschinen sind bei Kahn [21] und Singh [48] zu finden.

einen offenen Kanal erfolgen.

Neben dem Verschlüsseln vom Inhalt der Nachricht kann auch die Tatsache der Übermittlung der Nachricht versteckt werden. Damit befasst sich ein Teilgebiet der Kryptografie die *Steganographie*.[15] Die Steganographie befasst sich in erster Linie damit, die Tatsache, dass bestimmte Informationen übertragen oder gespeichert worden sind, zu verbergen. Bekannte historische Beispiele sind, die Nachricht auf die rasierte Kopfhaut eines Sklaven zu schreiben und sie damit unter den nachwachsenden Haaren zu verbergen oder statt Tinte Milch oder Zitronensaft zu verwenden.[16] Möglich und vor allem seit der Verwendung von Computern beliebt ist das Verstecken einer Nachricht in einer Bild- oder Audiodatei. Dabei werden oft bestimmte Bereiche eines Bildes als Informationsträger verwendet, die nicht ohne Weiteres von einem Betrachter als solche identifiziert werden können. Die Abbildung 2.1 zeigt ein Beispiel hierfür.

15. Manchmal wird Steganographie auch als ein eigenständiges Forschungsgebiet betrachtet.

16. Vgl. Bauer [1, S. 9 ff.].

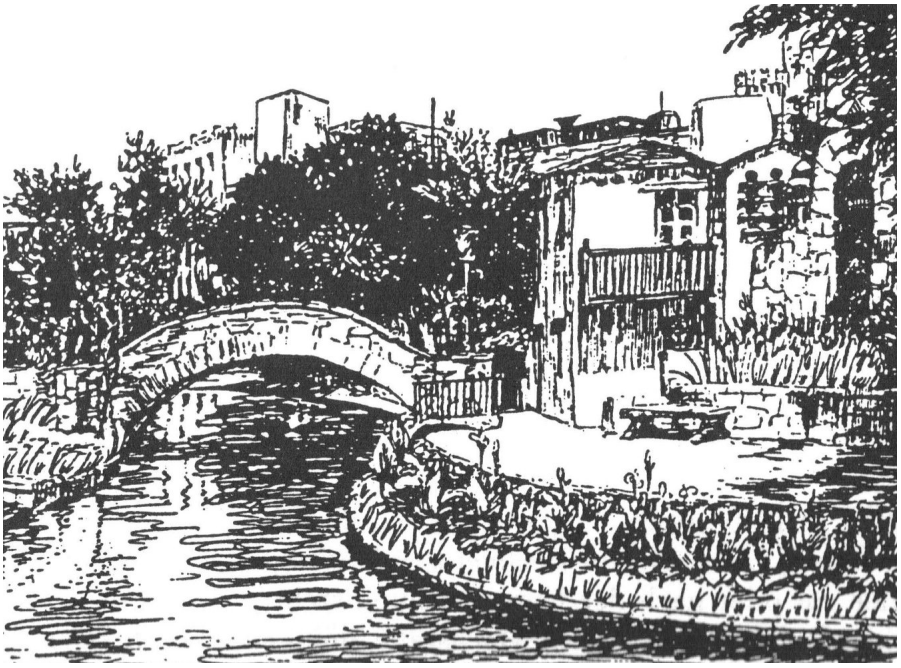


Abbildung 2.1: Beispiel für Steganographie in einem Bild.[17]

17. Die geheime Nachricht ist in Morsecode geschrieben, der aus kurzen und langen Grashalmen besteht, und lautet: »Compliments of CPSA MA to our chief Col. Harold R. Shaw on his visit to San Antonio May 11th 1945«.

Eine weitere, speziell in der Vergangenheit häufig anzutreffende Methode Texte zu verschlüsseln, besteht darin, speziell definierte Begriffe, Zeichenketten oder Wörter, genannt *Codes*,[18] als Substitute für die zu verschlüsselnde Wörter oder ganze Textpassagen zu verwenden. Ein Code operiert somit auf größeren lexikalischen Einheiten als ein Chiffre. Durch eine derartige Konstruktion entstehen Codebücher,

18. Eine gelegentliche Verwendung von englischen Begriffen ist in dieser Arbeit nicht zu vermeiden, da diese in der wissenschaftlichen Welt primär Verwendung finden.

die häufig aus mehreren Tausend Einträgen bestehen. Ein Nachteil ist zudem, dass die nicht im Codebuch erfassten Begriffe nicht chiffriert werden können. Ein bekanntes Beispiel ist ein Codebuch der deutschen Luftwaffe aus dem Jahr 1945.[19] Die Verwendung von Codes ermöglicht in der Regel neben der Geheimhaltung der Textinhalte auch eine Komprimierung der Größe der zu übermittelnden Nachricht.

19. Beispiel aus dem Codebuch der Luftwaffe; vgl. Kahn [21, S. 462]:
`sLk - Befehl`
`snb - Suchbefehl`
`xdm - Schlüssel vernichtet.`

2.1.3 Perfekte Sicherheit

Im Jahr 1945 entwickelte Claude Shannon[20] in einer zuerst unter Verschluss gehaltenen Arbeit die Voraussetzungen für einen Algorithmus mit einer *beweisbaren perfekten Sicherheit*. Die Arbeit wurde zum ersten Mal im Jahr 1949 publiziert. Laut Shannon sind folgende Voraussetzungen für einen Verschlüsselungsalgorithmus mit einer perfekten Sicherheit erforderlich:[21]

20. Claude Elwood Shannon (1916 – 2001), amerikanischer Mathematiker und Vorreiter auf dem Gebiet der Kryptografie.

21. Vgl. Shannon [46, S. 679 ff.].

- der Schlüssel muss genau so lang sein wie die Nachricht;
- der Schlüssel muss zufällig gewählt werden;
- der Schlüssel darf nur ein Mal verwendet werden.

Mit einer perfekten Sicherheit ist gemeint, dass der Angreifer keine Informationen über den Klartext bekommt, wenn nur der Kryptotext beobachtet wird. Seien $p \in \mathcal{P}$ ein Klartext und $c \in \mathcal{C}$ ein Kryptotext, dann gilt bei einer perfekten Sicherheit für alle p und c: [22]

22. Vgl. Stinson [49, S.48 ff.] für eine vollständige Herleitung.

$$P[p|c] = P[p],$$

die a posteriori Wahrscheinlichkeit, dass der verschlüsselte Klartext p ist, wenn ein Kryptotext c beobachtet wurde, ist genauso groß, wie die a priori Wahrscheinlichkeit, dass der Klartext p ist. Die zwei Ereignisse sind folglich stochastisch unabhängig. Anders ausgedrückt, hat ein kryptografisches System $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ eine perfekte Sicherheit, wenn:

$$\begin{aligned} &\forall\, p_1, p_2 \in \mathcal{P}: \quad |p_1| = |p_2|, \text{[23]} \quad \text{gilt} \\ &\forall\, c \in \mathcal{C}: \quad P\left[e_k(p_1) = c\right] = P\left[e_k(p_2) = c\right], \end{aligned}$$

23. p_1 und p_2 sind gleichlang.

wobei k gleichverteilt in $\mathcal{K}$ ist. Der einzige Verschlüsselungsalgorithmus, der diese Kriterien erfühlt, ist die im Jahr 1917 vorgestellte Vernam-Chiffre.[24] Bei der Vernam-Chiffre, auch One-Time-Pad genannt, wird jedes Klartextzeichen p_i mit dem jeweiligen Schlüsselzeichen k_i Modulo der Größe des Alphabets addiert. Bei einer Binärkodierung wird die Addition mittels einer XOR-Verknüpfung realisiert:

24. Benannt nach dem amerikanischen Kryptologen und dem Erfinder von Stromchiffren Gilbert Vernam (1890 – 1960).

$$c = p \oplus k, \quad \text{wobei } |p| = |k| = |c|.$$

Diese Art der Verschlüsselung ist sehr effizient und vergleichsweise einfach auch manuell durchzuführen. Wichtig ist, dass die Länge des Schlüssels genauso lang sein muss wie die Länge des Klartextes und dass der Schlüssel jeweils nur ein Mal für die Kommunikation verwendet wird. Wird der Schlüssel auch nur ein weiteres Mal mehr verwendet,[25] ist die Sicherheit nicht mehr gewährleistet. Denn, seien $c_1 = p_1 \oplus k$ und $c_2 = p_2 \oplus k$, dann gilt:

25. Dieser Fall wird *Two-Time-Pad* genannt.

$$c_1 \oplus c_2 = p_1 \oplus p_2,$$

und folglich können aufgrund der Redundanz z. B. der natürlichen Sprache die ursprünglichen Nachrichten p_1 und p_2 leicht ermittelt werden. Denkbar sind Analysen, die die Häufigkeit des Vorkommens von einzelnen Buchstaben der Sprache untersuchen und ausnutzen.

Diese Bedingungen implizieren auch die größte Schwierigkeit bei der Anwendung der Vernam-Chiffre. Würden die Benutzer eine Möglichkeit haben jederzeit pro Übermittlung der Nachricht über einen sicheren Kanal einen Schlüssel zu transportieren, der genauso lang ist wie die Daten selbst, dann würde es auch möglich sein direkt über diesen Kanal die eigentlichen Daten zu transportieren. Zudem ist die Generierung und die Verwaltung von solch langen Schlüsseln eine große Herausforderung.[26] Aus diesen Gründen ist die Verwendung der Vernam-Chiffre nur auf sehr spezielle Anwendungen beschränkt. Denkbar sind Anwendungen, bei denen die Verschlüsselung und Entschlüsselung manuell durchgeführt werden und zudem höchste Sicherheitsanforderungen erfüllt sein müssen.[27] Da die Anwendung der Vernam-Chiffre nicht effizient genug ist, musste nach neuen Algorithmen und Verfahren geforscht werden, um die es in den folgenden Abschnitten gehen wird.

26. Mehr zu der Erzeugung von großen Mengen von Zufallszahlen s. Stinson [49, S. 323 ff.].

27. Während des kalten Kriegs nutzten sowjetische Spione die Vernam-Chiffre für Ihre Kommunikation; vgl. Kahn [21, S. 662 ff.].

2.1.4 Kerckhoffs Prinzipien

> »Natürlich glaubt niemand, daß der CIA dem Mossad regelmäßig über die neuesten kryptographischen Algorithmen Bericht erstattet. Aber der Mossad würde sie über kurz oder lang wohl sowieso aufdecken.«
>
> Bruce Schneier [44, S. 6].

Im Jahr 1883 formulierte Auguste Kerckhoffs[28] eine zentrale Forderung, wie sichere kryptografische Systeme aufgebaut sein müssen. Die Forderung besagt, dass ein System auch dann sicher sein muss, wenn alles bis auf den geheimen Schlüssel über das System dem Angreifer bekannt ist: [29]

28. Jean Guillaume Auguste Victor François Hubert Kerckhoffs (1835 – 1903) – ein niederländischer Linguist und Kryptologe.

29. Vgl. Originalartikel von Kerckhoffs [23].

»A cryptosystem should be secure even if the attacker (Oscar) knows all details about the system, with the exception of the secret key. In particular, the system should be secure when the attacker knows the encryption and decryption algorithms.« [30]

30. Paar u. Pelzl [37, S. 11].

Insbesondere darf dem Angreifer der Algorithmus selbst, seine Arbeitsweise und die verwendeten Parameter bekannt sein. Für diese These spricht die Erfahrung, dass zahlreiche bisher bekannte und als sicher geltende Algorithmen, die unter *security by obscurity* [31] entwickelt worden sind, geknackt wurden, sobald die Arbeitsweise der Algorithmen öffentlich bekannt geworden ist.[32]

31. Deutsch: Sicherheit durch Obskurität.

32. Bekannte Beispiele sind die GSM Algorithmen A5/1 und A5/2.

Der US Geheimdienst NSA (National Security Agency) veröffentlicht dagegen nicht alle verwendeten Algorithmen. Die öffentlich verfügbare, so genannte *Suite B*, liefert eine Zusammenstellung der Algorithmen, die bis zum Level »secret«[33] verwendet werden können. Dagegen existiert für die höchste Geheimhaltungsstufe »top secret«, die so genannte *Suite A*. Die Algorithmen dieser Klasse sind geheim, werden nicht veröffentlicht und widersprechen somit dem Kerckhoffs Prinzip.[34] Bruce Schneier vermutet, dass dies auf der Tatsache beruht, dass die NSA die weltweit besten Kryptographen beschäftigt, die untereinander die erarbeiteten und verwendeten Algorithmen diskutieren und auf Schwachstellen prüfen [44, S. 8].

33. In den USA wird zwischen den folgenden Geheimhaltungsstufen unterschieden: »confidential«, »secret«, »top secret«; vgl. [50].

34. »Another suite of NSA cryptography, Suite A, contains some classified algorithms that will not be released. Suite A will be used for the protection of some categories of especially sensitive information.«, s. National Security Agency [35].

Die These Kerckhoffs wird aktuell in zahlreichen Verschlüsselungsverfahren umgesetzt, indem die eigentlichen Algorithmen offengelegt

und standardisiert werden. Es existieren eigene Standards für symmetrische (DES und später AES) und asymmetrische Verschlüsselung (RSA, ECC), für Hashfunktionen (MD5, SHA) sowie für den Schlüsselaustausch und digitale Signaturen.

2.1.5 Angriffsszenarien

Der Schutz vor Angriffen auf kryptografische Algorithmen und Protokolle kann nur dann wirksam sein, wenn die Angriffsmethoden und Techniken genau analysiert und auf die jeweiligen Algorithmen und Protokolle angewendet werden. Im Folgenden werden einige wichtige Angriffsszenarien beschrieben.[35]

35. Für weitere Informationen s. z. B. Paar u. Pelzl [37, S. 9 ff.] sowie Schneier [44, S. 5 ff.].

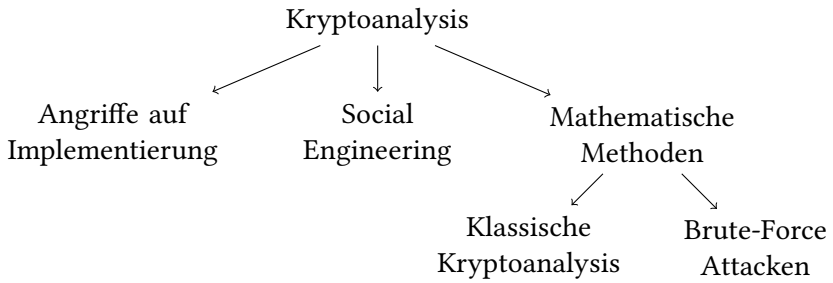


Abbildung 2.2: Überblick über die Angriffsmethoden in der Kryptoanalyse.[36]

36. Angelehnt an Paar u. Pelzl [37, S. 10].

Die Abbildung 2.2 zeigt grob die Übersicht über die möglichen Angriffsszenarien. Die Angriffe lassen sich dabei in drei Kategorien unterteilen: Angriffe auf die jeweilige Implementierung, die *social engineering Attacken*[37] und Angriffe auf die mathematische Logik der Algorithmen. Die ersten beiden Varianten sind von der Qualität des Algorithmus oder Protokolls unabhängig und können auch bei den als sicher geltenden Algorithmen erfolgreich angewandt werden.

37. Deutsch: Angriffe durch soziale Manipulation.

Bei den Angriffen auf die jeweilige Implementierung der Algorithmen wird von der Tatsache ausgegangen, dass entweder Programmierfehler bei der Implementierung begangen wurden, die ausgenutzt werden können, oder es werden so genannte *side channel attacks*[38] durchgeführt. Bei der Ausführung bestimmter kryptografischer Berechnungen gibt es charakteristische Größen, etwa den Energieverbrauch, die Ausführungsdauer oder Eigenschaften der emittierten elektromagnetischen Strahlung. Diese Effekte können je nach zu verschlüsselnder Nachricht unterschiedlich ausfallen. Die Unterschiede können dann

38. Deutsch: Seitenkanalangriffe.

mit spezieller Soft- und Hardware gemessen und analysiert werden. Aus diesen Daten können unter Umständen Rückschlüsse auf den zugrunde liegenden Klartext oder Schlüssel gezogen werden. Insbesondere bei den Algorithmen, die auf eingebetteten Systemen bzw. auf Chipkarten zum Einsatz kommen, ist diese Art von Angriff von großer Bedeutung.

Die social engineering Angriffe zielen darauf ab durch eine geschickte Manipulation und Täuschung die legalen Teilnehmer davon zu überzeugen, Teile oder die gesamten Klartexte bzw. Schlüssel freiwillig dem Angreifer zur Verfügung zu stellen. Die Angriffsmethoden von social engineering reichen in die Gebiete von Soziologie und Psychologie hinein.[39] Insbesondere sind die social engineering Angriffe für eine Mehrzahl der Angriffe über das Internet verantwortlich.

39. Vgl. Paar u. Pelzl [37, S. 11].

Die dritte Gruppe der Angriffe stellen die mathematischen Angriffe auf den zugrunde liegenden Algorithmus bzw. auf das Protokoll dar. Innerhalb dieser Gruppe wird oft zwischen den Brute-Force-Attacken[40] und klassischen kryptoanalytischen Verfahren unterschieden. Im Folgenden werden die wichtigsten Klassen der Angriffe kurz vorgestellt:

40. Deutsch: Angriff mit roher Gewalt.

Brute-Force Attack: Bei einem Brute-Force-Angriff werden alle möglichen Schlüsselkombinationen durchprobiert, bis ein erfolgreicher Treffer erzielt wird. Bei den modernen Verschlüsselungsverfahren und bei einer korrekten Anwendung dieser, ist ein solcher Angriff jedoch wenig aussichtsreich, da der zu durchsuchende Suchraum etwa gleich der Hälfte des möglichen Schlüsselraums ist. Bei z. B. AES-128 liegt der Bereich somit bei 2^{127}.

Ciphertext-only Attack: Bei dieser Angriffsmethode werden unterschiedliche Kryptotexte analysiert mit dem Ziel, Informationen über Klartexte oder sogar den Schlüssel zu erhalten. Dieser Angriffsmethode sind gewöhnlich alle Algorithmen ausgesetzt, da der Angreifer in der Regel über zahlreiche verschlüsselte Nachrichten verfügt, die frei übertragen wurden.

Known-plaintext Attack: Dem Angreifer sind nicht nur die verschlüsselten Texte, sondern auch die jeweiligen Klartexte der Nachrichten bekannt. Dadurch ergeben sich unter Umständen zusätzliche Möglichkeiten der Analyse mit dem Ziel, den geheimen Schlüssel zu erhalten.

Chosen-plaintext Attack: Der Angreifer kann den zu verschlüsselnden Text selbst frei festlegen. Daraufhin wird der Text von Alice mit dem jeweiligen Algorithmus verschlüsselt. Das Ziel ist den geheimen Schlüssel zu finden bzw. weitere geheime Nachrichten zu entschlüsseln.

Chosen-ciphertext Attack: Der Angreifer hat die Möglichkeit verschiedene verschlüsselte Texte zu entschlüsseln, ohne dabei den Schlüssel zu kennen. Praktisch kann ein solches System als eine Maschine vorgestellt werden, die eine Entschlüsselung der Texte durchführt, ohne dagegen dem Angreifer den Schlüssel zu verraten. Das Ziel des Angreifers ist es, den Schlüssel zu bestimmen.

Chosen-key Attack: Der Angreifer hat keine Kenntnisse über den gesamten Schlüssel, besitzt jedoch das Wissen über bestimmte Teile des Schlüssels bzw. über den Aufbau und hat gegebenenfalls die Möglichkeit, Teile des Schlüssels festzulegen.

Neben den unterschiedlichen Arten von Angriffen, sollte ein Angreifer stets beachten, wie aufwendig ein Angriff tatsächlich sein wird. So kann es unter Umständen sein, dass die »Kosten« für einen Angriff über dem eigentlichen »Gewinn« eines erfolgreichen Angriffs liegen. In diesem Fall kann ein System auch dann noch »sicher« sein, wenn die verwendeten Algorithmen theoretisch nicht mehr als sicher einzustufen sind. Ein Angriff kann unter kommerziellen Gesichtspunkten trotzdem nicht erfolgreich durchgeführt werden und wird daher wahrscheinlich nicht zustande kommen. Die Komplexität eines Angriffs wird oft unter den folgenden drei Aspekten bewertet:[41]

41. Vgl. Schneier [44, S. 9 f.].

1. **Datenkomplexität**: Hiermit wird die Menge der Daten gemessen, die zur Durchführung eines Angriffs nötig sind. Bei bestimmten Arten von Angriffen kann beispielsweise die erforderliche Anzahl der unterschiedlichen Klartextnachrichten bei 2^n liegen, wobei n die Größe des Schlüssels ist.

2. **Berechnungskomplexität**: Hiermit wird die Zeit gemessen, die für einen erfolgreichen Angriff nötig ist. Bei den Brute-Force-Angriffen beispielsweise, liegt die Berechnungskomplexität für einen erfolgreichen Angriff bei etwa 2^{n-1}.

3. **Speicherverbrauch**: Hiermit wird der Speicherplatz gemessen, der für einen erfolgreichen Angriff benötigt wird. Häufig stellt dieses Kriterium eine starke Einschränkung bzgl. der praktischen Umsetzung von bestimmten Angriffsszenarien dar.

Moderne Methoden der Kryptografie werden häufig in zwei Bereiche aufgeteilt: *symmetrische* und *asymmetrische* Kryptografie. Die Verfahren der symmetrischen Kryptografie werden wiederum häufig in Stromalgorithmen und Blockalgorithmen untergliedert.[42]

42. S. Abbildung 2.3 und Paar u. Pelzl [37, S. 3 f.].

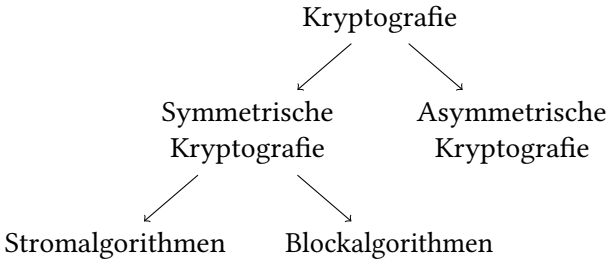


Abbildung 2.3: Überblick über die Verschlüsselungsmethoden der Kryptographie.[43]

43. Vgl. Schneier [44, S. 4 ff.].

In den folgenden Abschnitten werden jeweils einige wichtige Definitionen zur symmetrischen und asymmetrischen Kryptografie gegeben.

2.2 SYMMETRISCHE KRYPTOGRAFIE

Bei der symmetrischen Kryptografie besitzt das zugrunde liegende Kryptosystem nur einen einzigen Schlüssel k, der sowohl für die Verschlüsselung, als auch für die Entschlüsselung des jeweiligen Klartextes genutzt wird.[44] Es gilt demnach:

44. Weitere Informationen zu den symmetrischen Algorithmen sind z. B. bei Paar u. Pelzl [37, S. 29 ff.] zu finden.

$$\forall\, p \in \mathcal{P}, \forall\, k \in \mathcal{K}: \quad d_k\left(e_k(p)\right) = p$$

In der Abbildung 2.4 wird ein Beispiel für eine gewöhnliche Kommunikation bei einer symmetrischen Verschlüsselung dargestellt. Alice als Initiator des Prozesses verschlüsselt mit der Verschlüsselungsfunktion $e_k(p)$ den Klartext p mit dem Schlüssel k. Anschließend werden die verschlüsselten Daten, der Kryptotext c, über einen offenen Kanal zu Bob gesendet. Während der Übertragung können die Daten von einem passiven Angreifer Eve betrachtet und von einem aktiven Angreifer Oscar modifiziert werden. Der Schlüssel k muss für die Entschlüsselung über einen separaten, gesicherten Kanal, zu dem Oscar und

Eve keinen Zugang haben, an Bob übertragen werden.[45] Nach Erhalt der Daten wendet Bob die Entschlüsselungsfunktion $d_k(c)$ mit dem separat erhaltenen Schlüssel k auf den Kryptotext c an und erhält den Klartext p wieder. Ohne die Kenntnis des Schlüssels k können weder Oscar noch Eve den Kryptotext entschlüsseln. Für die krypto-

45. In der Praxis stellt diese Anforderungen häufig eine nicht zu unterschätzende Schwierigkeit dar.

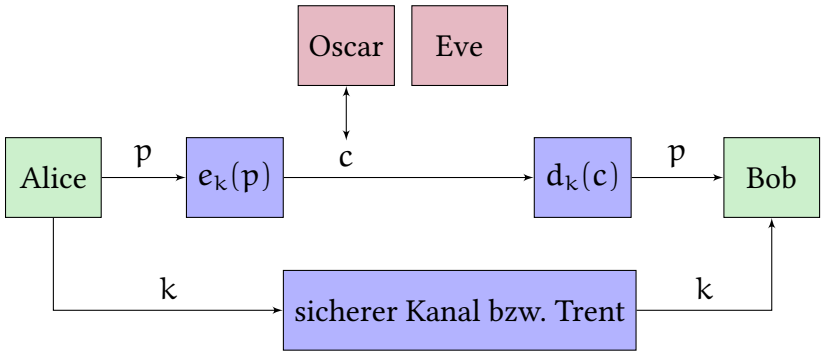


Abbildung 2.4: Kommunikation bei einem symmetrischen Verfahren.

grafische Sicherheit der Algorithmen ist neben einer ausreichenden Schlüssellänge insbesondere das Design der Algorithmen entscheidend. Die modernen symmetrischen Algorithmen bestehen häufig aus einer Verkettung von primitiven mathematischen Operationen wie Multiplikationen oder Additionen.

Wie in der Abbildung 2.3 bereits gesehen, werden die symmetrischen Verfahren in Stromalgorithmen und Blockalgorithmen unterteilt. Die Stromalgorithmen verschlüsseln jedes Bit des Klartextes separat. Dabei wird aus dem vom Benutzer definierten Schlüssel ein so genannter *Schlüsselstrom* erzeugt, der wiederum jeweils Bit für Bit mit dem Klartext verknüpft wird.[46]

46. Mehr zu Stromalgorithmen bei Paar u. Pelzl [37, S. 29 ff.].

Die Blockalgorithmen operieren dagegen auf fest definierten Blöcken von Daten. Die typischen Blockgrößen sind 64 bzw. 128 Bit. Die interne Struktur von Blockalgorithmen besteht aus einer Routine für die Erzeugung von den so genannten *Rundenschlüsseln* und mehreren Runden, die anschließend durchgeführt werden. Die typische Anzahl von Runden reicht von 10 bis 64. In diesen Runden wird die eigentliche Modifikation des Klartextes vorgenommen und die Verknüpfung mit dem jeweiligen Rundenschlüssel durchgeführt. Die zwei häufigsten Konstruktionen für moderne Blockalgorithmen ist das Substitutions-Permutations-Netzwerk und das Feistel-Netzwerk, die in den folgenden Abschnitten vorgestellt werden.

Die Blockalgorithmen gelten allgemein als etwas besser erforscht und finden aktuell häufiger Verwendung, insbesondere bei den Kommunikationsprotokollen für das Internet.[47] Zudem existieren zahlreiche frei verfügbare Blockalgorithmen, die auch unter eingeschränkten Ressourcen gute Resultate zeigen. Aus diesen Gründen wird in der vorliegenden Arbeit der primäre Fokus auf die Untersuchung von Blockalgorithmen gelegt.

47. Vgl. Paar u. Pelzl [37, S. 31].

2.2.1 Substitutions-Permutations-Netzwerk

Die folgende Konstruktion stellt ein Grundgerüst für zahlreiche moderne Algorithmen und Verfahren der Kryptografie dar. Wie der Name schon andeutet, besteht eine solche Konstruktion aus mehreren Substitutions- und Permutationsroutinen, die in einem System miteinander verknüpft sind.

Bei einer Substitution c wird jedes Zeichen aus der Menge der möglichen Klartexte $\mathcal{P}$ durch ein oder mehrere Zeichen aus der Menge der möglichen Kryptotexte $\mathcal{C}$ substituiert. Es gilt folglich:

$$\mathsf{c} : \mathcal{P} \rightarrow \mathcal{C}$$

Bekannte Beispiele für eine Substitution sind die Geheimzeichen Karls des Großen (s. Abbildung 2.5), die Caesarverschlüsselung sowie die so genannte Freimaurerchiffre.[48] Ist die Substitution eindeutig und

48. Mehr zu den jeweiligen Verfahren s. Bauer [1, S. 45 ff.].

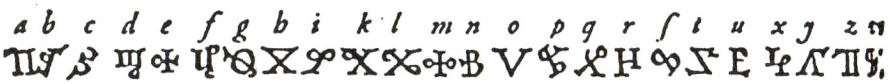


Abbildung 2.5: Geheimzeichen Karls des Großen

umkehrbar, wird von einem umgeordneten Alphabet gesprochen.[49] Für den Spezialfall, dass die Menge der Klartexte zudem gleich der Menge der Kryptotexte ist, also $\mathcal{P} \equiv \mathcal{C}$ und folglich eine Substitution der Form $\mathcal{P} \leftrightarrow \mathcal{P}$ stattfindet, wird von einer Permutation gesprochen. Bei einer Permutation wird demnach nur die Reihenfolge der Elemente der Klartextmenge vertauscht. Es wird gelegentlich auch von einer Transposition gesprochen. Eine Unterscheidung in Substitutions- und Permutationschiffren wurde zum ersten Mal von Giambattista Della Porta[50] im Jahr 1563 vorgenommen.

49. Vgl. Bauer [1, S. 47].

50. Giambattista Della Porta (1535 – 1615) neapolitanischer Universalgelehrter; vgl. Stinson [49, S. 19].

In seiner fundamentalen Arbeit [46] aus dem Jahr 1949 hat Claude Shannon die theoretischen Grundlagen sowie Verfahren der symmetrischen Kryptografie definiert. Eine zentrale Erkenntnis war, dass nur durch eine Kombination beider Verfahren, nämlich der Substitution und Permutation, eine sichere Verschlüsselung realisiert werden kann.[51] Eine weitere wichtige Überlegung war die Verbindung von mehreren Kryptosystemen zu einem einzigen Kryptosystem. Dabei wird in der Regel ein Produkt aus den jeweiligen Kryptosystemen gebildet.

51. Vgl. Paar u. Pelzl [37, S. 57 f.].

Seien S_1 und S_2 zwei Kryptosysteme mit zwei Schlüsseln k_1 und k_2 und sei $\mathcal{P} = \mathcal{C}$, so ist das Produkt von zwei Kryptosystemen gleich $S_1 \times S_2 = (\mathcal{P}, \mathcal{P}, \mathcal{K}_1 \times \mathcal{K}_2, \mathcal{E}, \mathcal{D})$. Dabei ist die kombinierte Verschlüsselungsfunktion e in $S_1 \times S_2$ auf einem Klartext p definiert als:

$$e_{(k_1,k_2)}(p) = e_{k_2}\left(e_{k_1}(p)\right),$$

die Entschlüsselungsfunktion d ist definiert als:

$$d_{(k_1,k_2)}(c) = d_{k_1}\left(d_{k_2}(c)\right),$$

wobei zu beachten ist, dass bei der Entschlüsselung die Schlüssel im Vergleich zur Verschlüsselung in umgekehrter Reihenfolge verwendet werden. Diese Überlegungen und Vorgehensweisen bilden die Grundlage für Algorithmen, die auf eine Kombination aus Substitutionen und Permutationen basieren und spielen somit eine große Rolle bei zahlreichen modernen Algorithmen wie beispielsweise AES.[52]

52. Für weitere Informationen sowie der Herleitung der Formeln s. Stinson [49, S. 67 ff.].

Bei einem Substitutions-Permutations-Netzwerk werden mehrere Kryptosysteme mit jeweils unterschiedlichen Schlüsseln miteinander verbunden, wobei häufig mehrere *Runden* von Substitutionen bzw. Permutationen implementiert werden. Die Abbildung 2.6 zeigt den typischen Aufbau eines Substitutions-Permutations-Netzwerkes. Abwechselnd erfolgen Verknüpfungen mit dem aktuellen Rundenschlüssel, Anwendung von Substitutions-Boxen und Permutations-Routinen. Bei einem Substitutions-Permutations-Netzwerk werden *alle* Bits pro Runde verändert.[53]

53. Bei einem Feistel-Netzwerk wird meistens nur die Hälfte aller Bits pro Runde verändert; vgl. Abschnitt 2.2.2.

Die Entschlüsselung im Substitutions-Permutations-Netzwerk erfolgt durch eine Anwendung der inversen Operationen in umgekehrter Reihenfolge. Aus diesem Grund müssen alle in dem Netzwerk verwen-

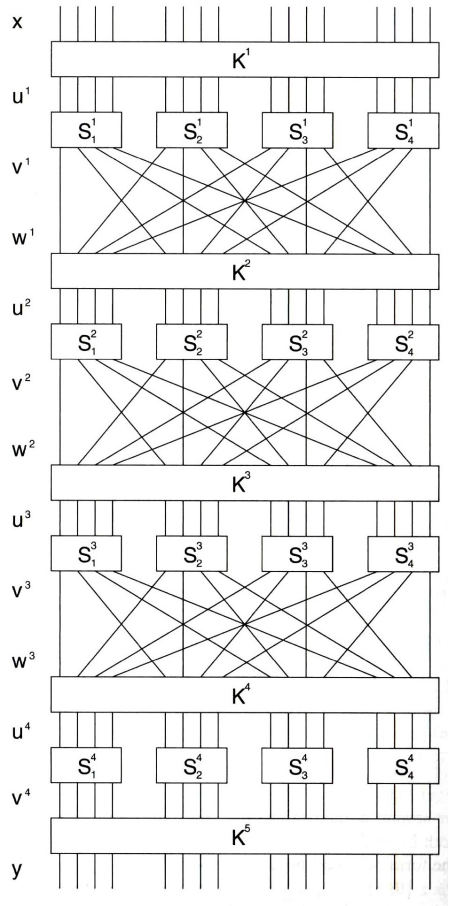


Abbildung 2.6: Aufbau eines Substitutions-Permutations-Netzwerks.

deten Funktionen eine Inverse haben. Zudem sollte die Komplexität der Ausführung von den inversen Funktionen nicht stark von der Komplexität der eigentlichen Funktion abweichen. Ansonsten würde beispielsweise die Dauer der Entschlüsselung in dem Netzwerk anders ausfallen als die Dauer der Verschlüsselung. Diese Eigenschaft muss insbesondere beim Design von Algorithmen bedacht werden.

2.2.2 *Feistel-Netzwerk*

Ein Feistel-Netzwerk, benannt nach Horst Feistel,[54] ist die zweite wichtige Konstruktion für Blockalgorithmen. Ein Feistel-Netzwerk besteht, wie auch ein Substitutions-Permutations-Netzwerk, aus mehreren Runden, wobei in jeder Runde ein neuer Rundenschlüssel angewendet wird. Ein wesentlicher Unterschied jedoch ist die Tatsache, dass pro Runde nur ein Teil der gesamten Bits verändert wird. Dazu wird bei einem Feistel-Netzwerk der Block in meistens zwei gleichgroße

54. Horst Feistel (1915 – 1990), deutsch-amerikanischer Kryptologe, Entwickler der Verschlüsselung *Lucifer*.

Teile (L_i) und (R_i) aufgeteilt. In den jeweiligen Runden werden die Teilblöcke nach dem folgenden Prinzip berechnet:[55]

$$L_{i+1} \leftarrow R_i \tag{2.4}$$

$$R_{i+1} \leftarrow L_i \oplus F(R_i, k_i)\,. \tag{2.5}$$

55. Solche Feistel-Netzwerke werden als *ausgeglichen* bezeichnet. Es existieren jedoch auch unausgeglichene Feistel-Netzwerke z. B. der Algorithmus *Skipjack*; s. [34].

Dabei wird der alte rechte Teilblock (2.4) ohne Veränderung zu dem neuen linken Teilblock transferiert. Der neue rechte Teilblock (2.5) wird aus einer XOR-Verknüpfung von dem linken Teilblock und dem in der Funktion F modifizierten rechten Teilblock berechnet. In die Funktion F wird in jeder Runde zusätzlich ein Rundenschlüssel k_i als Parameter übergeben.[56] Die Abbildung 2.7 verdeutlicht den Aufbau einer Feistel-Runde graphisch.

56. Vgl. Mao [29, S.218 ff.].

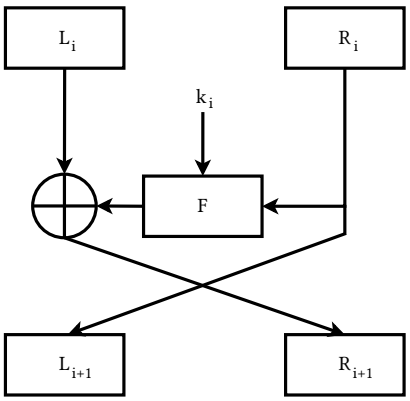


Abbildung 2.7: Aufbau eines Feistel-Netzwerks.

Die Entschlüsselung in einem solchen Netzwerk funktioniert durch die Anwendung der Rundenfunktionen F_1 bis F_n in umgekehrter Reihenfolge.[57] Dadurch ergibt sich ein wesentlicher Vorteil von Feistel-Netzwerken gegenüber Substitutions-Permutations-Netzwerken: Es muss keine explizite Entschlüsselungsfunktion definiert sein und die Rundenfunktion F muss demnach nicht effizient invertierbar sein. Daraus folgt eine gegebenenfalls leichtere Umsetzung in Soft- und vor allem in Hardware.[58] Der wohl bekannteste Beispiel für ein Feistel-Netzwerk ist das frühere Verschlüsselungsstandard DES.[59]

57. Wurde bei der Verschlüsselung $F_1 \ldots F_n$ angewendet, so wird bei der Entschlüsselung $F_n \ldots F_1$ angewendet.

58. Vgl. Mao [29, S. 221].

59. Mehr zu DES im Abschnitt 3.2.1.

2.2.3 Operationsmodi von Blockchiffren

Die zuvor vorgestellten Herangehensweisen an den Aufbau von Blockalgorithmen beschreiben in erster Linie die Verarbeitung von nur einem

Block von Daten. Gewöhnlich ist aber die Länge der zu verschlüsselnden Daten größer als die Blockgröße des Algorithmus.[60] Neben den unterschiedlichen Varianten, wie Blockalgorithmen intern aufgebaut werden können, sollen daher auch die unterschiedliche Arten wie eine Verarbeitung von mehreren Blöcken durchgeführt werden kann, beleuchtet werden. Diese unterschiedlichen Arten der Verarbeitung werden *modes of operation* genannt.[61] Die Betriebsmodi sind unabhängig von dem verwendeten Blockalgorithmus. Im Folgenden werden die wichtigsten Betriebsmodi vorgestellt:[62]

60. Blockalgorithmen operieren selten auf Blocklängen, die größer als 128 Bit sind.

61. Deutsch: Betriebsmodi von Blockalgorithmen.

62. In der Fachliteratur haben sich primär die englischen Bezeichnungen durchgesetzt.

Electronic Code Book: Die Electronic Code Book Modus ist die einfachste Variante wie Verschlüsselungen mit einem Blockalgorithmus durchgeführt werden können. Dabei werden die Eingabeblöcke jeweils separat, Block für Block, verschlüsselt. Es findet keine Verknüpfung zwischen den einzelnen Blöcken statt. Vorteil dieses Modus ist die einfache Arbeitsweise. Ein großer Nachteil ist, dass die Muster im Kryptotext, die Muster im Klartext widerspiegeln. Die Verschlüsselung desselben Blocks des Klartextes ergibt immer denselben Block im Kryptotext. Bei zahlreichen Anwendungen kann dies ein Sicherheitsproblem darstellen.[63]

63. Fängt der Angreifer zwei Kryptotexte ab, die im ECB-Modus verschlüsselt wurden, kann er ohne Kenntnis der eigentlichen Nachricht erkennen, ob die zwei Klartexte gleich sind oder nicht.

Cipher Block Chaining: Im Cipher Block Chaining Modus existiert eine Verknüpfung zwischen den jeweiligen Blöcken. Zuerst wird der erste Block des Klartextes mit einer Bitfolge, genannt *Initialisierungsvektor*[64] mittels XOR verknüpft. Anschließend erfolgt eine Verschlüsselung, wobei der resultierende Kryptotext als neuer Initialisierungsvektor für den nächsten Block verwendet wird. Der Initialisierungsvektor wird dabei zusammen mit den eigentlichen verschlüsselten Daten übermittelt und wird entsprechend für die Entschlüsselung benutzt. Der CBC-Modus wird häufig für die Integritätsprüfung benutzt.[65]

64. Initialisierungsvektor (IV) ist meistens genauso lang wie ein Block. Alle Modi bis auf den ECB-Modus arbeiten mit Initialisierungsvektoren.

65. Mehr zum CBC-Modus vgl. Abschnitt 4.2.2.

Output Feedback Mode: Die Verwendung des Output Feedback Mode ermöglicht aus einem Blockalgorithmus einen Stromalgorithmus zu erzeugen. Der Initialisierungsvektor wird dabei wie ein Klartextblock behandelt und verschlüsselt. Nach der Verschlüsselung entsteht folglich eine Pseudozufallsfolge, die anschließend mittels XOR mit dem Klartext verknüpft wird. Diese

Pseudozufallsfolge wird zudem neuer Initialisierungsvektor und wird im nächsten Schritt verwendet. Der Modus bietet zwei große Vorteile. Zum einen kann die Pseudozufallsfolge im Voraus, unabhängig von dem eigentlichen Klartext berechnet werden. Zum anderen wird keine eigene Entschlüsselungsfunktion benötigt. Die Entschlüsselung funktioniert analog, indem anstatt des Klartextes der Kryptotext mit der Pseudozufallsfolge mittels XOR verknüpft wird.

Counter Mode: Beim Counter Mode entsteht nach der Verschlüsselung ebenfalls eine Pseudozufallsfolge. Somit kann eine Blockverschlüsselung in diesem Modus ebenfalls als Stromverschlüsselung benutzt werden. Zwischen den einzelnen Blöcken existiert allerdings keine Verknüpfung. Die Besonderheit des Counter Modus ist die Wahl des Initialisierungsvektors. Die Hälfte des Initialisierungsvektors stellt eine Zufallszahl dar und die andere Hälfte beinhaltet einen Zähler.[66] Dieser Zähler wird pro Block um eins erhöht und ergibt dadurch pro Verschlüsselung eines Blocks jeweils einen anderen Initialisierungsvektor. Es gelten die gleichen Eigenschaften, wie bei dem OFM-Modus,[67] zusätzlich gibt es den Vorteil, dass die Berechnungen parallelisiert werden können.

66. Daher die Bezeichnung Counter-Mode.

67. Vorausberechnung der Zufallsfolge und der Verzicht auf eine Entschlüsselungsfunktion.

Der Initialisierungsvektor muss nicht geheim übertragen werden, sondern kann an den übertragenen Kryptotext angehängt werden. Da die typischen Klartexte wesentlich länger als eine Blockgröße sind, stellt diese zusätzliche Speicherung von einem Initialisierungsvektor keine besondere Einschränkung dar.[68] Häufige Anforderung bei allen Modi ist, dass der IV pro verschlüsseltem Klartext immer unterschiedlich sein muss. Es existieren zahlreiche weitere Betriebsmodi von Blockalgorithmen auf die in der vorliegenden Arbeit nicht näher eingegangen wird. Weitere Informationen zu den unterschiedlichen Betriebsarten sind beispielsweise bei Stinson [49, S. 109 ff.] und Paar u. Pelzl [37, S. 124 ff.] zu finden.

68. Bei kleinen Datenmengen kann die zusätzliche Speicherung und Übertragung vom IV jedoch eine Herausforderung darstellen.

2.2.4 *Federleichte Kryptografie*

Mit *federleichter Kryptografie*[69] wird ein Gebiet innerhalb der Kryptografie bezeichnet, der sich mit der Entwicklung von speziellen Al-

69. Englische Bezeichnung: lightweight cryptography.

gorithmen für den Einsatz unter sehr eingeschränkten Ressourcen beschäftigt. Die zentralen Kriterien bei der Entwicklung von derartigen Algorithmen sind: [70]

70. Vgl. Leander [27].

- Effizienz der Implementierung auf kleinen Mikrocontrollern in Soft- und Hardware;
- eine relativ geringe Block- und Schlüsselgröße;
- zahlreiche Runden mit möglichst einfachen Operationen innerhalb einer Runde.

Typische Blockgröße ist 64 Bit und die typische Schlüsselgröße liegt bei 80 bis 128 Bit. Die Anzahl von Runden beträgt bei manchen Algorithmen etwa 100. Im Abschnitt 3.2.4 wird mit PRESENT ein Algorithmus aus dem Bereich der federleichten Kryptografie vorgestellt. Weitere Informationen und insbesondere zahlreiche Forschungsarbeiten sind unter [38] zu finden.

2.3 Asymmetrische Kryptografie

Im Gegensatz zu der symmetrischen Kryptografie ist die asymmetrische Kryptografie ein relativ modernes Gebiet der Wissenschaft und entwickelte sich erst in den 1970er Jahren des 20. Jahrhunderts. Maßgeblich bei der Entwicklung beteiligt war der britische Nachrichtendienst GCHQ [71], der unter anderem für die Entwicklung neuer kryptografischen Verfahren zuständig ist. Den ersten öffentlich publizierten Algorithmus lieferten Diffie, Hellman und Merkle im Jahr 1976.[72]

71. GCHQ – Government Communications Headquarters.

72. Vgl. Paar u. Pelzl [37, S. 149].

Das Konzept der asymmetrischen Verschlüsselung entstand primär aus dem Wunsch, große Mengen von Schlüsseln effizient verteilen zu können. Bei einer symmetrischen Verschlüsselung muss vor der eigentlichen Kommunikation ein geheimer Schlüssel zwischen den beteiligten Parteien ausgetauscht werden. Ist die Menge der Parteien jedoch groß und soll pro Verbindung jeweils ein eigener Schlüssel benutzt werden, so steigt die Anzahl der benötigten Schlüsseln schnell an. Bei n Parteien liegt die Anzahl an benötigten Schlüsseln bei:[73]

73. Vgl. Paar u. Pelzl [37, S. 151].

$$\frac{n \cdot (n-1)}{2}$$

Die Verteilung von derartigen Mengen an Schlüsseln stellt ein Problem dar, da für jeden Austausch eine gesicherte Verbindung hergestellt werden muss.[74]

74. Vgl. die Abbildung 2.4.

Eine asymmetrische Verschlüsselung ist strukturell anders aufgebaut als eine symmetrische. Es existieren zwei unterschiedliche Schlüssel: ein öffentlicher Schlüssel k, der für die Verschlüsselung von Daten p benutzt wird und ein privater Schlüssel k', der für die Entschlüsselung verwendet wird. Aus der Kenntnis des öffentlichen Schlüssels kann der private Schlüssel nicht effizient ermittelt werden. Die kryptografische Sicherheit beruht dabei auf der Idee spezielle Funktionen, die sogenannten *Einwegfunktionen* einzusetzen. Eine Einwegfunktion ist eine Funktion f, die einfach zu berechnen ist, wobei das Inverse f^{-1} der Funktion sehr schwer zu berechnen ist: [75]

75. Mehr zu den Einwegfunktionen z. B. bei Stinson [49, S. 161 ff.].

$$\begin{aligned} y &= f(x) && \text{einfach zu berechnen;} \\ x &= f^{-1}(y) && \text{schwer zu berechnen.} \end{aligned}$$

Ist jedoch ein geheimer Schlüssel k' bekannt, kann das Inverse der Funktion wiederum einfach berechnet werden. Ein bekanntes asymmetrisches Kryptosystem RSA[76] basiert auf der Schwierigkeit große Zahlen zu faktorisieren. Dies bildet die zugrunde liegende Einwegfunktion.

76. Vgl. Abschnitt 3.2.5.

Der öffentliche Schlüssel kann, wie der Name schon andeutet, öffentlich, z. B. auf einem Server, publiziert werden. Für die initiale verschlüsselte Kommunikation ist somit kein vorhergehender Austausch der Schlüssel über einen gesicherten Kanal erforderlich. Die Abbildung 2.8 verdeutlicht den Aufbau der Kommunikation bei einem asymmetrischen Verschlüsselungsverfahren. Für die Verschlüsselung

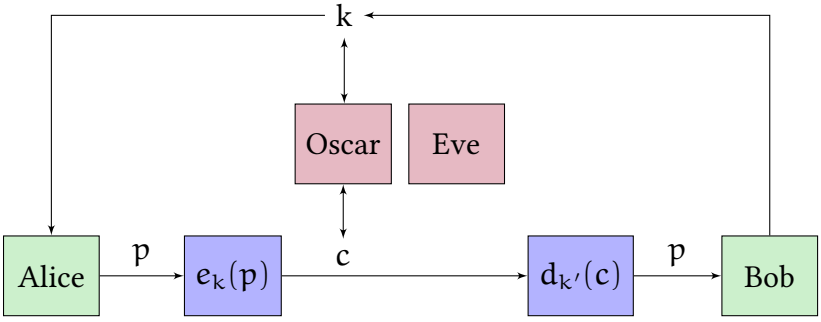


Abbildung 2.8: Kommunikation bei einem asymmetrischen Verfahren.

von Daten p benutzt Alice den von Bob an einem öffentlichen Server abgelegten Schlüssel k. Die verschlüsselten Daten c sendet Alice über einen offenen Kanal an Bob, der den geheimen Schlüssel k' für die Entschlüsselungsfunktion benutzt. Der aktive Angreifer Oscar kann den öffentlichen Schlüssel k von Bob einsehen und gegebenenfalls auch modifizieren bzw. mit einem eigenen ersetzen. Diese Art von Angriff wird ein sogenannter *Man-in-the-middle-Angriff* genannt.[77] Oskar kann folglich den von Bob bereitgestellten öffentlichen Schlüssel mit seinem eigenen ersetzen. Bleibt dieser Austausch von Alice unbemerkt, wird sie die Nachrichten folglich nicht mehr mit dem öffentlichen Schlüssel von Bob, sondern mit dem öffentlichen Schlüssel Oskars verschlüsseln. Oskar kann im nächsten Schritt die von Alice verschickten Nachrichten c mit dem eigenen privaten Schlüssel entschlüsseln, gegebenenfalls modifizieren und weiter an Bob, nunmehr verschlüsselt mit seinem öffentlichen Schlüssel weiterleiten. Demnach ist für die Verhinderung eines Angriffs, eine Signierung von öffentlichen Schlüsseln notwendig. Eine derartige digitale Signatur kann ebenfalls mit asymmetrischen Methoden durchgeführt werden. Hierfür wird der private Schlüssel für die Erzeugung einer Signatur verwendet und der öffentliche Schlüssel für die Überprüfung der Signatur benutzt. Es existiert eine Reihe von Ansätzen und Protokollen, wie eine derartige Signierung durchgeführt werden soll. Gängige Ansätze sind die Verwendung von speziellen Zertifizierungsstellen und das Konzept von *Web of Trust*.[78]

77. Deutsch: Mittelsmannangriff.

78. Mehr zu den asymmetrischen Kryptosystemen und Verfahren ist bei Paar u. Pelzl [37, S. 154 ff.] zu finden.

In den folgenden Kapiteln werden zahlreiche symmetrische Algorithmen sowie ein asymmetrischer Algorithmus untersucht und bewertet. Es wird jeweils ein spezieller Fokus auf den Einsatz auf Mikrocontroller gelegt.

SICHERHEITSINFRASTRUKTUR UNTER BEGRENZTEN RESSOURCEN

3

> »If you think technology can solve your problems you don't understand technology and you don't understand your problems.«
>
> Bruce Schneier, zit. nach [7, S. 182].

In diesem Kapitel werden zuerst die zwei untersuchten Mikrocontroller vorgestellt sowie die technologischen Beschränkungen aufgezeigt. Anschließend findet eine Darstellung der analysierten Algorithmen statt. Dabei werden vier symmetrische und ein asymmetrisches Verfahren vorgestellt und miteinander verglichen. Im Anschluss findet eine empirische Untersuchung der Algorithmen auf zwei Mikrocontrollern statt. Neben der benötigten Speichergröße und Dauer der Verschlüsselung wird der Datendurchsatz gemessen sowie einige datenabhängige Berechnungen durchgeführt.

3.1 DARSTELLUNG DER AUSGEWÄHLTEN PLATTFORM

In der vorliegenden Arbeit werden kryptografische Verfahren mit dem Ziel, diese auf Mikrocontrollern einzusetzen, untersucht. Ein Mikrocontroller besteht in der Regel aus einem Prozessor, einem Speicher für den Programmcode, genannt Flash-Speicher, einem Arbeitsspeicher, genannt RAM, einem überschreibbaren Speicher, genannt EEPROM und einer Reihe von Ein- und Ausgängen für die Kommunikation mit den externen Anwendungen. Die Programmierung auf den Mikrocon-

trollern erfolgt in der Regel in Assembler bzw. in der Programmiersprache C.[79]

79. Es existieren auch speziell entwickelte Sprachen z. B. JAL.

Bei einem Einsatz von kryptografischen Methoden auf Mikrocontrollern treten eine Reihe von Besonderheiten und speziellen Herausforderungen auf, die im Folgenden dargestellt werden:

Sehr beschränkte Speicherkapazität: Insbesondere die Speicherkapazität zur Laufzeit des Programms im RAM ist sehr eingeschränkt und liegt bei etwa 1 KByte.

Taktrate der Prozessoren: Die Taktrate der Prozessoren ist mit einigen wenigen MHz bis etwa 20 MHz ebenfalls eher gering. Trotzdem darf die Ausführung von kryptografischen Algorithmen die Ausführung der eigentlichen Geschäftslogik nur unwesentlich verlangsamen.

Beschränkte Energieressourcen: Die Energieressourcen sind häufig beschränkt, da auf eine permanente Stromversorgung oft verzichtet werden muss.

Klare Größe der ausgetauschten Daten: Bei vielen Anwendungen ist die Größe der ausgetauschten Daten gering und im Vorfeld der Kommunikation fest definiert.

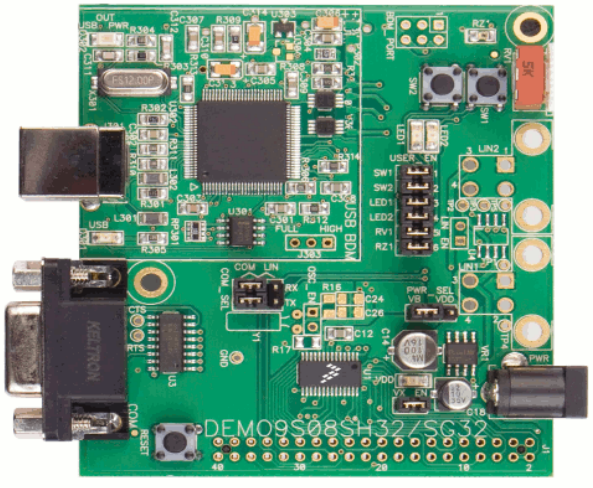


Abbildung 3.1: Darstellung eines development boards mit einem 8-Bit Mikrocontroller des Herstellers Freescale Semiconductor.

Die gegebene Infrastruktur besteht aus zwei unterschiedlichen Mikrocontrollern zwischen denen eine Kommunikation über einen Kommu-

nikationskanal mit einer sehr beschränkten Kapazität stattfindet. Es existiert keine permanente Stromversorgung, sodass die Dauer der Berechnung auf den Prozessoren gering gehalten werden soll, um Energie zu sparen.

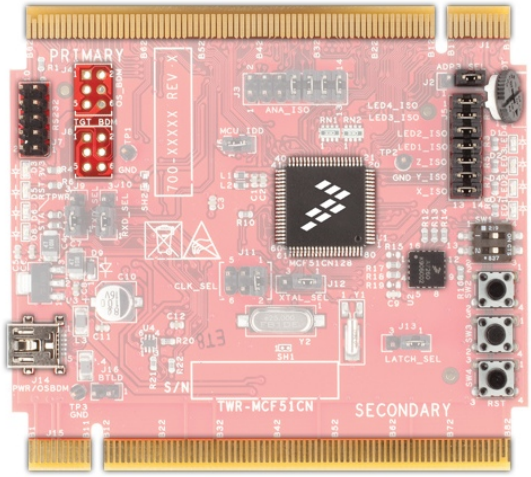


Abbildung 3.2: Darstellung eines development boards mit einem 32-Bit Mikrocontroller des Herstellers Freescale Semiconductor.

Der erste Mikrocontroller ist ein 8-Bit Mikrocontroller des Herstellers Freescale Semiconductor vom Typ MC9S08SH32.[80] Die Abbildung 3.1 zeigt ein Foto vom Mikrocontroller auf einer Entwicklerplatine. Auf dieser Entwicklerplatine sind neben dem eigentlichen Controller auch noch weitere Komponenten für die Anbindung z. B. an einen PC vorgesehen. Bei dem 32-Bit Mikrocontroller handelt es sich um einen Mikrocontroller des Typs MCF51CN128 ColdFire, ebenfalls von der Firma Freescale Semiconductor.[81] In der Abbildung 3.2 ist dieser Typ des Mikrocontrollers ebenfalls auf einer Entwicklerplatine abgebildet. Die Tabelle 3.1 gibt einen Überblick über die wichtigsten Hardwareparameter der zwei Mikrocontroller.

80. Genaue Details sind der Spezifikation [13] zu entnehmen.

81. Die Spezifikation ist unter [14] zu finden.

Typ	Prozessor	FLASH	RAM	EEPROM
8-Bit MC	8 MHz	32 KByte	1 KByte	521 KByte
32-Bit MC	20 MHz	128 KByte	24 KByte	1 MByte

Tabelle 3.1: Vergleich der zwei untersuchten Mikrocontroller.

3.2 Darstellung der Algorithmen

In diesem Abschnitt werden die ausgewählten Algorithmen dargestellt und miteinander verglichen. Im darauf folgenden Abschnitt findet eine empirische Untersuchung der vorgestellten Algorithmen statt. Bei der Auswahl der in Frage kommenden Algorithmen wurden insbesondere folgende Aspekte berücksichtigt:[82]

82. Es wurde versucht mit den aufgestellten Kriterien möglichst alle Varianten der Blockalgorithmen abzudecken.

- es sollten sowohl ein Substitutions-Permutations-Netzwerk als auch ein Feistel-Netzwerk untersucht werden;
- es sollten primär symmetrische Algorithmen untersucht werden, jedoch sollte zum Vergleich auch ein asymmetrisches Verfahren untersucht werden;
- es sollten möglichst weit verbreitete Algorithmen ausgewählt werden, die bereits eine tiefer gehende Analyse und Überprüfung erfahren haben;
- es sollte ein Algorithmus aus dem Bereich der »federleichten« Kryptografie untersucht werden;[83]
- ein aktueller Standard für Verschlüsselungen ist ebenfalls auszuwählen;
- ein »älterer« Algorithmus sollte zusätzlich untersucht werden.

83. Vgl. den Abschnitt 2.2.4.

Unter Betrachtung dieser Kriterien wurden für die Gruppe der symmetrischen Kryptografie, die Algorithmen AES, DES bzw. 3DES ausgewählt. Die Gruppe der federleichten Algorithmen wird durch die symmetrischen Chiffren TEA bzw. XTEA sowie PRESENT repräsentiert. Aufgrund seiner starken Verbreitung wurde RSA für die Gruppe der asymmetrischen Verfahren ausgewählt. AES und PRESENT sind zudem Substitutions-Permutations-Netzwerke, wobei DES und TEA/XTEA als Feistel-Netzwerke aufgebaut sind. Mit DES ist ein Algorithmus aus den 70er Jahren des 20. Jahrhunderts vertreten und mit AES ein aktueller Standard für Blockverschlüsselungen.[84]

84. Die Algorithmen aus dem Gebiet der elliptischen Kryptografie sowie Stromalgorithmen wurden nicht in die Untersuchung miteinbezogen, weil dies den Rahmen der vorliegenden Diplomarbeit sprengen würde.

Mit der Darstellung der Algorithmen wird das Ziel verfolgt, einen Überblick über die jeweiligen zugrundeliegenden Operationen zu geben. Damit soll beleuchtet werden, wie sich das Zusammenspiel von den jeweiligen Operationen und Konstruktionsideen auf die Dauer

der Ausführung der Operationen, die Speicherbelegung und weitere Leistungsmerkmale auswirken. Die exakten Beschreibungen der Algorithmen mit weiteren Erläuterungen und Kommentaren sind in den angegebenen Referenzen, in den Codebeispielen bei den folgenden Abschnitten sowie auf der zur Arbeit beiliegenden CD zu finden.

3.2.1 *DES und 3DES*

> »DES trained a generation of cryptographers.«
>
> Knudsen u. Robshaw [24, S. 13].

DES, der Data Encryption Standard, war der erste öffentlich publizierte und standardisierte Algorithmus für Verschlüsselungen und markiert damit eine neue Ära in der Kryptografie. Federführend bei der Einführung des Standards im Jahr 1977 war die Vorgängerorganisation der heutigen NIST in den USA, die NBS.[85] Da die Einführung des DES einen Meilenstein in der Geschichte der Kryptografie darstellt, wird im Folgenden detailliert auf den Prozess der Einführung und den eigentlichen Algorithmus eingegangen.

85. NBS – National Bureau of Standards wurde im Jahr 1988 in National Institute of Standards and Technology (NIST) umbenannt.

Obwohl einige Unternehmen und Regierungen schon damals viel Wissen und praktische Erfahrung in der Entwicklung von Verschlüsselungsalgorithmen gesammelt hatten, blieben diese Erkenntnisse weitgehend unter Verschluss und waren nur geheimen Regierungsbehörden oder den Unternehmen selbst zugänglich.[86] Mitte der 1970er Jahre wurde allerdings der Bedarf an kryptografischen Algorithmen auch für kommerzielle Anwendungen derart groß, dass die Initiative ergriffen wurde, einen öffentlichen Standard für Verschlüsselungen zu etablieren.

86. Selbst die Existenz der für die kryptografische Forschung zuständigen US Behörde NSA (National Security Agency) wurde zu der damaligen Zeit geheim gehalten; vgl. Paar u. Pelzl [37, S. 56].

Die erste Ausschreibung des NBS im Jahr 1973 brachte keine brauchbaren Ergebnisse. Lediglich während der erneuten Ausschreibung im Jahr 1974 wurde ein vom Unternehmen IBM eingebrachter Vorschlag für eine Blockverschlüsselung als aussichtsreich bewertet.[87] Der Vorschlag beruhte auf dem Algorithmus *Lucifer*, einer früheren internen Entwicklung von IBM unter Horst Feistel.[88] Im Jahr 1977 wurde der neue Algorithmus von NBS schließlich verabschiedet und galt mehr als 20 Jahre als Standard für die Verschlüsselung von nicht geheimer Information in den Vereinigten Staaten.

87. Vgl. Knudsen u. Robshaw [24, S. 13 f.].

88. Mehr zum Aufbau von Lucifer im Originalartikel von Feistel [9].

DES ist ein Feistel-Netzwerk mit einer Blockgröße von 64 Bit und einer Schlüsselgröße von 56 Bit. Die Anzahl der Runden beträgt 16, wobei in jeder Runde ein eigener, neuer Rundenschlüssel benutzt wird.[89] Als erstes wird bei DES eine initiale Permutation (IP) vorgenommen:

89. Die exakte Spezifikation von DES ist in der NIST Publikation unter [32] zu finden.

$$(L_o, R_o) \leftarrow IP(\text{Input}).$$

Diese initiale Permutation ist fest vorgegeben und nicht von den Eingabeparametern abhängig. Somit liefert diese Permutation keine Sicherheit im kryptografischen Sinn. Die Idee hinter der initialen Permutation ist, die Implementierungen von DES in Hardware zu erleichtern. Dazu werden bei der IP die Bits, die bei den darauf folgenden Operationen benutzt werden, hintereinander als 8-Bit Blöcke platziert.[90] Dadurch kann die Komplexität der Verdrahtungen bei einem Hardware-Design verringert werden und die Chipgröße dadurch möglichst klein gehalten werden. Häufig wird auf die initiale Permutation bei Software Implementierungen von DES verzichtet. Eine derartige Implementierung verstößt jedoch gegen die DES Spezifikation und sollte daher auch nicht DES genannt werden.[91]

90. Zur Zeit der Entwicklung von DES waren primär 8-Bit Prozessoren in Gebrauch.

91. Vgl. Schneier [44, S. 317].

Da DES ein ausgeglichenes Feistel-Netzwerk ist, wird ein Block in zwei gleichgroße Teile (L_o) und (R_o) von je 32 Bit aufgeteilt. Anschließend werden 16 Runden nach dem für Feistel-Netzwerke typischen Muster berechnet, indem pro Runde durch die Anwendung der Funktion F und des Rundenschlüssels k_i nur die Hälfte der Bits verändert wird.[92] Nach 16 durchgeführten Runden wird die initiale Permutation wieder rückgängig gemacht:

92. Vgl. Abschnitt 2.2.2.

$$\text{Output} \leftarrow IP^{-1}(L_{16}, R_{16}).$$

Diese Operationen sind sowohl für die Verschlüsselung als auch für die Entschlüsselung gleich. Der einzige Unterschied ist die Reihenfolge der Anwendung von Rundenschlüsseln $k_1 \ldots k_{16}$. Werden bei der Verschlüsselung die Rundenschlüsseln in der Reihenfolge $k_1, k_2, k_3 \ldots k_{16}$ verwendet, so muss bei der Entschlüsselung die umgekehrte Reihenfolge $k_{16}, k_{15}, k_{14} \ldots k_1$ angewendet werden.

Die eigentliche kryptografische Sicherheit von DES wird durch die Funktion F hergestellt. Diese besteht aus vier nacheinander ausgeführten Operationen:[93]

93. Die Abbildung 3.3 zeigt schematisch die Arbeitsweise der Funktion F bei DES.

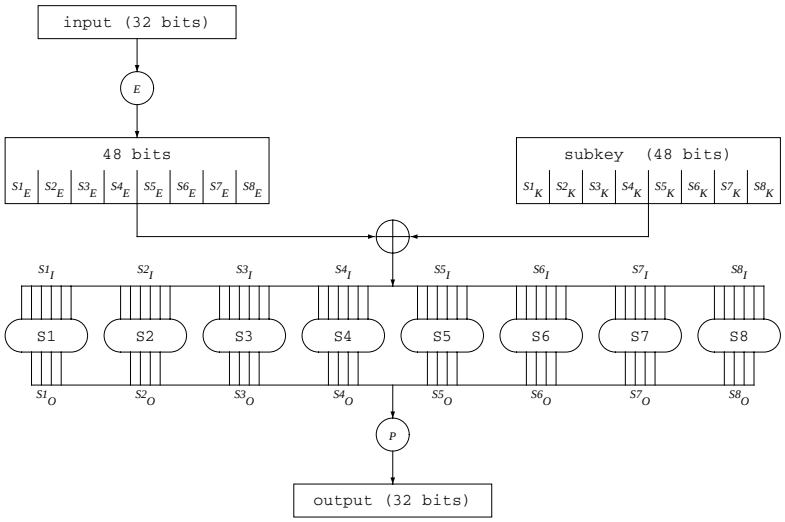


Abbildung 3.3: Darstellung der Arbeitsweise der Funktion F bei DES.

1. Zuerst wird die 32 Bit große rechte Hälfte des Blocks auf 48 Bit expandiert. Dabei werden einige Bits verschoben und andere dupliziert. Diese Operation wird als Expansionspermutation (E-Box) bezeichnet.[94]

   $$E : \{0,1\}^{32} \rightarrow \{0,1\}^{48}$$

   Ziel dieser Expansion ist die Erzeugung des so genannen *Lawineneffekts*. Damit werden massive Veränderungen bei den Ausgabebits bezeichnet, wenn sich nur einige der Eingabebits verändern.

2. Als nächstes werden die 48 Ausgabebits der E-Box mit dem jeweiligen, ebenfalls 48 Bit großen Rundenschlüssel mittels XOR verknüpft.

3. Anschließend werden die resultierenden Bits in acht jeweils 6 Bit große Blöcke aufgeteilt. Diese Blöcke bilden die Eingangsbits für die 8 verschiedene Substitutions-Boxen (S-Box).[95] Als Ausgabe liefern die S-Boxen jeweils 4 Bit und verkürzen somit die Ausgabe um 2 Bit:

   $$S_i : \{0,1\}^6 \rightarrow \{0,1\}^4$$

   Die Substitutions-Boxen sind für die kryptografische Sicherheit

94. Eine genaue Beschreibung der E-Box ist z. B. bei Schneier [44, S. 319 f.] zu finden.

95. Die Verwendung von acht unterschiedlichen Substitutions-Boxen ist eine Besonderheit von DES.

von DES ganz wesentlich, da diese nicht linear sind und so die differentielle Kryptoanalyse wesentlich erschweren.[96]

4. Als letzte Operation innerhalb der Funktion F wird eine Permutation durchgeführt:

$$P : \{0,1\}^{32} \rightarrow \{0,1\}^{32}$$

Nach dieser Operation ist die Funktion vollständig abgearbeitet und es kann mit der neuen Runde begonnen werden.

96. Differentielle Kryptoanalyse ist eine Technik, bei der die Differenzen in Klartexten und die Auswirkung auf die Kryptotexte untersucht werden. Mehr dazu s. Biham u. Shamir [2].

Die Rundenschlüssel für die 16 Runden von DES werden durch die folgende Funktion gebildet: Zuerst erfolgt eine Aufteilung des 56 Bit großen Schlüssels in zwei gleichgroße Hälften von je 28 Bit. Anschließend wird, abhängig von der aktuellen Rundenzahl, eine Verschiebung der Bits in diesen beiden Hälften um 1 oder 2 Bit durchgeführt. Durch eine anschließende Kompressionsfunktion werden 48 Bit ausgewählt und bilden den aktuellen Rundenschlüssel. Jedes Bit des gesamten 56 Bit großen Schlüssels wird dabei in etwa 14 der 16 Rundenschlüsseln verwendet.[97]

97. Vgl. Schneier [44, S. 318] für die genaue Definition der Kompressionsfunktion.

Insbesondere bei der Größe der Schlüssel wird die Rolle der NSA bei der Entwicklung des Standards stark kritisiert. Die von IBM bei der Ausschreibung eingereichte Version des Algorithmus hatte ursprünglich einen 128 Bit großen Schlüssel.[98] Durch die Verringerung der Schlüsselgröße auf 56 Bit ist der Algorithmus wesentlich anfälliger für Brute-Force-Angriffe geworden. Im Jahr 1998 existierten bereits spezielle Maschinen für etwa 250.000 US Dollar, die einen Brute-Force-Angriff auf DES in unter 56 Stunden durchführten.[99]

98. Lucifer hatte ursprünglich 128 Bit große Schlüssel und eine ebenfalls 128 Bit große Blockgröße; vgl. Paar u. Pelzl [37, S. 56].

99. Vgl. Mao [29, S. 222].

Ein zweiter Kritikpunkt war die Gestaltung und vor allem die Kriterien bei der Auswahl der Substitutions-Boxen. Die beiden Behörden NSA und IBM legten sehr lange die Designkriterien, die bei der Implementierung von S-Boxen verwendet wurden, nicht offen. Es wurde befürchtet, dass die NSA oder IBM eine Hintertür beim Design von S-Boxen eingebaut hatten, mit der es möglich sein sollte, den Algorithmus schneller als mit einem Brute-Force-Angriff erfolgreich zu attackieren. Nach neueren Erkenntnissen gelten diese Anschuldigungen jedoch als widerlegt.[100]

100. Vgl. Schneier [44, S. 341 f.].

Zur Verbesserung der Widerstandsfähigkeit von DES gegen Brute-Force-Angriffe wurde im Jahr 1981 von Ralph Merkle und Martin

Hellman eine Dreifachverschlüsselung auf der Grundlage von DES vorgeschlagen. Eine doppelte Verschlüsselung bringt die erhoffte Erhöhung der Sicherheit nicht, da ein einfacher *meet-in-the-middle-Angriff* durchgeführt werden kann.[101] Die dreifache Anwendung von DES wird 3DES oder auch Tripple-DES genannt und wendet häufig die folgende Struktur an:

101. Vgl. Originalartikel Merkle u. Hellman [30].

$$C = DES_{k_3}\left(DES^{-1}_{k_2}\left(DES_{k_1}(P)\right)\right)$$

Dabei wird der Klartext P zuerst mit DES und dem Schlüssel k_1 verschlüsselt, dann mit dem Schlüssel k_2 entschlüsselt und schließlich mit dem dritten Schlüssel k_3 erneut verschlüsselt. Es sind auch Konstruktionen mit jeweils nur zwei unabhängigen Schlüsseln möglich, diese bieten jedoch nicht die gleiche Sicherheit.[102]

102. S. Knudsen u. Robshaw [24, S. 30 ff.] für weitere Details.

DES war mit Abstand der am meisten verwendete Algorithmus für symmetrische Kryptografie in den Jahren 1980 bis 2000. Auch heute wird DES und insbesondere 3DES häufig z. B. in Bankensoftware eingesetzt.[103] Im Zusammenhang mit der Verschlüsselung auf Mikrocontrollern wird DES bzw. 3DES heutzutage jedoch kaum in neuen Produkten verwendet. Zum einen bietet DES nicht mehr eine ausreichende Sicherheit gegen Brute-Force-Angriffe. Und zum anderen ist der Datendurchsatz bei der Verwendung von 3DES im Vergleich zu beispielsweise AES auf kleinen Mikrocontroller geringer.[104]

103. Vgl. Paar u. Pelzl [37, S. 55].

104. Vgl. den Abschnitt 3.3.2.

3.2.2 AES

Im Jahr 1997 startete NIST [105] die Initiative, einen Nachfolger für DES, den bisherigen Standard für Verschlüsselungen, zu suchen. Die wichtigste Begründung für den Schritt war die Tatsache, dass DES nicht mehr die modernen Anforderungen an eine sichere Kommunikation erfühlt hatte. Der Hauptkritikpunkt war die geringe Größe des Schlüsselraums bei DES von lediglich 56 Bits. Die häufig verwendete verbesserte Variante des Algorithmus, die 3DES, erfüllte ebenfalls nicht alle gewünschten Kriterien und war insbesondere auf Software Implementierungen nicht effizient genug. Zudem war die Blockgröße von 64 Bits nicht ausreichend groß, um beispielsweise Hashfunktionen aufbauend auf dem Algorithmus konstruieren zu können.[106]

105. NIST – National Institute of Standards and Technology, eine für Standardisierungen zuständige Behörde in den USA.

106. Vgl. Paar u. Pelzl [37, S. 87 ff.].

Daher wurde noch im selben Jahr ein Wettbewerb ausgerufen mit

dem Ziel, ein neues Verschlüsselungsverfahren, genannt Advanced Encryption Standard (AES), zu entwickeln und zu standardisieren. Der Wettbewerb an sich sowie das Vorgehen gelten bis heute als ein Musterbeispiel bei der Entwicklung von neuen kryptografischen Algorithmen und Standards. Der Algorithmus musste über die folgenden zentralen Eigenschaften verfügen:[107]

107. Vgl. Paar u. Pelzl [37, S. 88].

- als symmetrische Blockchiffre implementiert sein;
- über eine Blockgröße von mindestens 128 Bit verfügen;
- die Größe des Schlüsselraums von 128, 192 und 256 Bit haben;
- eine effiziente Implementierung sowohl in Hardware als auch in Software ermöglichen;
- gute Resultate, sowohl auf kleinen Mikrocontrollern als auch auf großen Rechenanlagen zeigen;
- resistent gegen alle bisher bekannten Angriffe sein;
- frei von jeglichen Patenten sein.

Im Jahr 2000 wurde schließlich Rijndael [108] unter den fünf Kandidaten in der letzten Runde als neuer symmetrischer Verschlüsselungsstandard ausgewählt. Gründe für die Bevorzugung waren ein elegantes Design von Rijndael sowie eine hohe Effizienz der Implementierung in Soft- und Hardware. Seitdem wurde AES in zahlreichen kryptografischen Protokollen implementiert, unter anderem in IPsec, IEEE 802.11i und TLS. AES ist heute der am meisten verwendete symmetrische Verschlüsselungsalgorithmus weltweit.[109]

108. Benannt nach den belgischen Entwicklern Vincent Rijmen und Joan Daemen.

109. Vgl. Paar u. Pelzl [37, S. 87].

AES ist ein Substitutions-Permutations-Netzwerk mit einer Blockgröße von 128 Bit und einer Schlüsselgröße von wahlweise 128 Bit, 192 Bit bzw. 256 Bit. Abhängig von der Schlüsselgröße verändert sich die Anzahl der erforderlichen Runden des Substitutions-Permutations-Netzwerks (s. Tabelle 3.2). Intern werden die 128 Bit der Blockgröße jeweils in 16 Feldern der Länge 8 Bit zusammengefasst, wobei die Daten in Form einer Matrix von 4×4 Feldern angeordnet sind.[110] Für Substitutionen werden spezielle Substitutions-Boxen verwendet, die gewöhnlich als Konstanten im Algorithmus definiert werden. In jeder AES Runde werden folgende Operationen durchgeführt:

Schlüssellänge	Runden
128 Bit	10
192 Bit	12
256 Bit	14

Tabelle 3.2: Anzahl der Runden bei unterschiedlichen Schlüsselgrößen.

110. Die 4×4 Matrix von AES:

m_0	m_4	m_8	m_{12}
m_1	m_5	m_9	m_{13}
m_2	m_6	m_{10}	m_{14}
m_3	m_7	m_{11}	m_{15}

SubBytes: Jedes Byte des Arrays wird durch ein anderes Byte aus der Substitutionen-Box ersetzt.[111]

111. Die S-Box kann entweder als ein Array aus Konstanten beschrieben werden oder auch algebraisch definiert werden, s. dazu Stinson [49, S. 104 ff.].

ShiftRows: Jede Zelle der Matrix wird nach links rotiert, wobei folgende Regel angewandt wird: Für alle Zellen der Zeile i werden die Zellen nach links jeweils um i Stellen rotiert, wobei für i gilt: $0 \leqslant i \leqslant 3$.

MixColumns: Jedes Byte $(m_1 ... m_3)$ der ursprünglichen Spalte wird mit einer konstanten Matrix P multipliziert:[112]

112. Für weitere Details s. z. B. Mao [29, S. 226 ff.].

$$C = P \cdot M = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \end{pmatrix}$$

AddKey: In jeder Runde wird die aktuelle Zelle der Matrix m_i mit der jeweiligen Zelle des aktuellen Rundenschlüssels k_i verknüpft, wobei die XOR Operation angewendet wird: $c_i = k_i \oplus m_i$.

Die Rundenschlüssel von AES werden in einer separaten Funktion erzeugt. Ein vom Anwender definierter und übergebener Schlüssel K wird für die Erzeugung der eigentlichen Rundenschlüssel $k_1 ... k_n$ verwendet. Für jede einzelne Runde von AES und zusätzlich für die allererste Runde[113] wird ein separater Rundenschlüssel erzeugt. Da der Schlüssel bitweise mit dem aktuellen Zustand mittels XOR verknüpft wird, ist jeder Rundenschlüssel genauso lang wie der Block selbst, nämlich 128 Bit. Die Rundenschlüssel werden in einem Array gespeichert, wobei die Größe des gesamten Arrays von der gewählten Variante von AES abhängig ist. Unterschiedliche Größen des Arrays sind in Tabelle 3.3 dargestellt.

113. Oft *pre-whitening* genannt.

AES	Größe in Bit
128	$11 \times 128 = 1.408$
192	$13 \times 128 = 1.664$
256	$15 \times 128 = 1.920$

Tabelle 3.3: Größe des Arrays in Bit bei den unterschiedlichen Varianten von AES.

Abbildung 3.4 zeigt schematisch die Arbeitsweise der Funktion für die Erzeugung von Rundenschlüsseln bei AES-128. Jeder Rundenschlüssel wird intern in vier Blöcke zu je 32 Bit aufgeteilt. Für die erste Runde wird der vom Anwender bereitgestellte Schlüssel für die vier Ausgangsblöcke $k_0 ... k_4$ verwendet. Anschließend werden die ersten drei Blöcke durch XOR miteinander verknüpft, wobei der vierte Block mit dem ersten verknüpft wird. Der vierte Block wird außerdem modifiziert, indem sowohl eine Permutation als auch eine Substitution

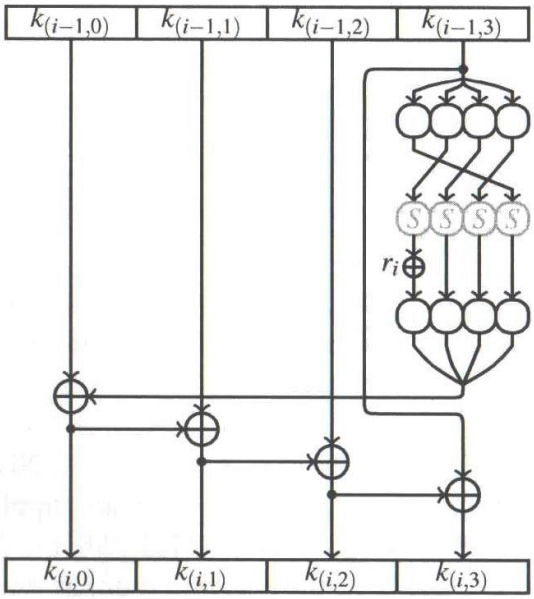


Abbildung 3.4: Schematische Darstellung der Funktion für die Erzeugung von Rundenschlüsseln bei AES-128.

angewendet wird.[114] Es findet zudem eine Addition der Konstanten r_i nach der durchgeführten Substitution statt. Für die AES Variante mit 192 und 256 Bit großen Schlüsseln gelten leicht abgewandelte Varianten der Funktion.[115] Im Folgenden wird die komplette Reihenfolge der Operationen von AES in einem Pseudocode dargestellt:

114. Für die Substitution wird dieselbe Substitutions-Box verwendet wie bereits für die Verschlüsselung.

115. Eine ausführliche Beschreibung der Funktion ist bei Knudsen u. Robshaw [24, S. 45 ff.] zu finden.

```
1  /* Encrypt a single block of 16 bytes */
2  aes_encrypt(plain, key, cipher) {
3    add_round_key(); //Pre round
4    //AES-Rounds (max - 1), 13 for AES-256
5    for (round = 1; round < max-1; round++) {
6      sub_bytes();
7      shift_rows();
8      mix_columns();
9      add_round_key();
10   }
11   //Final round
12   sub_bytes();
13   shift_rows();
14   add_round_key();
15 }
```

Quellcode 3.1: Implementierung der Verschlüsselungsfunktion in AES.

Da jede Operation in AES eine inverse Operation beinhaltet, ist die Entschlüsselung recht einfach zu implementieren. Für die Substitutions-Box wird eine entsprechende inverse Substitutions-Box verwenden. Bei den Verschiebungen von Zeilen wird eine Verschiebung nach rechts, statt nach links durchgeführt. Und für die Vertauschung von Spalten wird die Multiplikation mit der inversen Matrix M^{-1} durchgeführt. Für eine exakte Beschreibung der Arbeitsweise von AES sowie für zahlreiche Hintergrundinformationen ist in erster Linie das Buch der Entwickler von AES, Daemen u. Rijmen [6], zu empfehlen. Weitere Informationen sind bei Knudsen u. Robshaw [24, S. 35 ff.] sowie in der Publikation des NIST [31] zu finden.

3.2.3 *TEA und XTEA*

> »one of the most secure cipher algorithms ever devised ... and certainly the simplest!«
>
> Simon Shepherd [47].

Ein weiterer Algorithmus, der untersucht wird, ist der Tiny Encryption Algorithm (TEA) bzw. seine verbesserte Version der Extended Tiny Encryption Algorithm (XTEA).[116]

116. Beide Algorithmen wurden von britischen Wissenschaftlern Roger Needham (1935 – 2003) und David Wheeler (1927 – 2004) entwickelt.

TEA wurde zum ersten Mal im Jahr 1994 vorgestellt und beruht auf einem Feistel-Netzwerk mit einer Blockgröße von 64 Bit. Ziel der Entwicklung des Algorithmus war, ein sehr einfaches Verschlüsselungsverfahren zu finden, das sogar auf kleinsten Mikrocontrollern und Smart-Karten sehr gute Resultate in Bezug auf die Geschwindigkeit und die Größe des benötigten Speichers liefert. Trotzdem musste der Algorithmus die hohen Sicherheitsanforderungen einhalten und gegen die meisten bekannten Angriffe resistent sein. Die Idee der Konstruktion beruhte auf dem Prinzip vieler Iterationen eines Feistel-Netzwerkes zu realisieren, wobei jede einzelne Iteration für sich besonders einfach und effizient sein sollte. Daher werden bei TEA in jeder Runde nur die folgenden Operationen durchgeführt: XOR, die Addition modulo 2^{32} und die bitweise Verschiebung.[117]

117. Die genaue Definition von TEA ist in dem von den Autoren des Algorithmus verfassten Artikel Wheeler u. Needham [54] zu finden.

Der Algorithmus besteht aus 32 internen Runden, wobei jede Runde zwei Runden des Feistel-Netzwerks implementiert. Die gesamte Anzahl der durchgeführten Feistel-Runden liegt daher bei 64. Die Schlüsselgröße beträgt 128 Bit. In der Abbildung 3.5 wird die Arbeits-

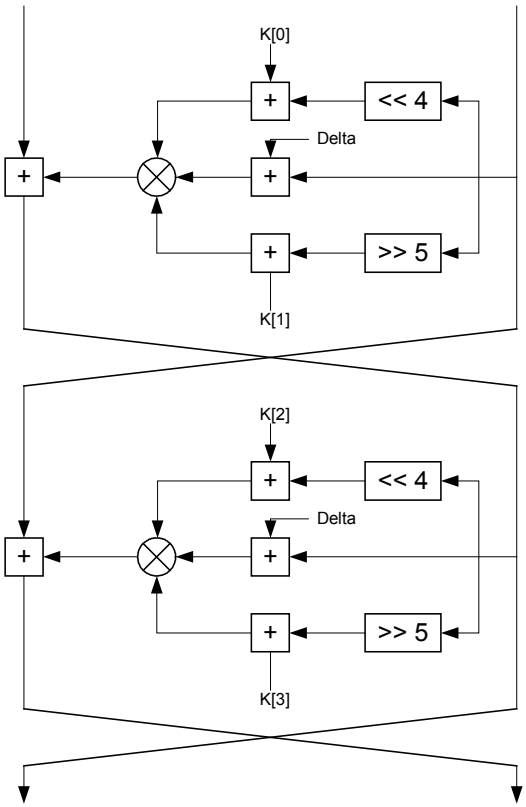


Abbildung 3.5: Darstellung der Arbeitsweise von einer Runde in TEA.

weise von einer TEA-Runde dargestellt. Wie in den meisten Feistel-Netzwerken wird der Klartext in zwei gleichgroße Teile aufgeteilt.[118] Zudem wird eine Konstante *Delta* – δ, wie folgt definiert:

$$\delta = (\sqrt{5} - 1) \cdot 2^{31} \approx 2654435769$$

Die Festlegung des Wertes für die Konstante ist aus der Definition des goldenen Schnittes abgeleitet.[119] Die Konstante δ wird in jeder Runde des Algorithmus nach dem folgenden Schema erweitert:

$$\mathrm{sum}_i = \mathrm{sum}_{i-1} + \delta$$

wobei bei der ersten Feistel-Runde gilt, dass $\mathrm{sum}_0 = 0$ ist.

In jeder Runde wird zuallererst eine bitweise Verschiebung des rechten Blocks des Feistel-Netzwerks um 4 Bits nach links und um 5 Bits nach rechts durchgeführt.[120] Beide modifizierten Blöcke werden zuerst unabhängig voneinander zwischengespeichert. Anschließend erfolgt eine Addition modulo 2^{32} mit dem Rundenschlüssel zu den modifizierten Blöcken. Als letzte Aktion erfolgt eine Verknüp-

118. Es existieren Feistel-Netzwerke bei denen die Klartexte beispielsweise in vier Blöcke aufgeteilt werden, z. B. Twofish [45].

119. Goldener Schnitt: $\frac{1}{\varphi} = \frac{\sqrt{5}-1}{2}$, vgl. Wheeler u. Needham [54, S. 2].

120. Die bitweise Verschiebung wird mit << bzw. >> angegeben.

fung der beiden Blöcke mittels XOR miteinander und der Konstante δ. Das Resultat der Berechnung wird zu dem linken Block des Feistel-Netzwerkes addiert. Der rechte Block des Feistel-Netzwerks wird ohne Veränderung an die nächste Feistel-Runde übergeben.

Da die Blockgröße des Algorithmus bei 64 Bit liegt, ist die Größe eines Blocks in der Feistel-Runde gleich 32 Bit. Der addierte Rundenschlüssel ist ebenfalls 32 Bit groß und wird durch eine einfache Aufspaltung des gesamten Schlüssels K definiert. Die ungeraden Runden benutzen die Schlüssel $[K_0]$ und $[K_1]$ als Rundenschlüssel und die geraden $[K_2]$ und $[K_3]$.[121]

121. Die genaue Aufspaltung und Addition der Rundenschlüssel $[K_0]$...$[K_3]$ ist in der Abbildung 3.5 sichtbar.

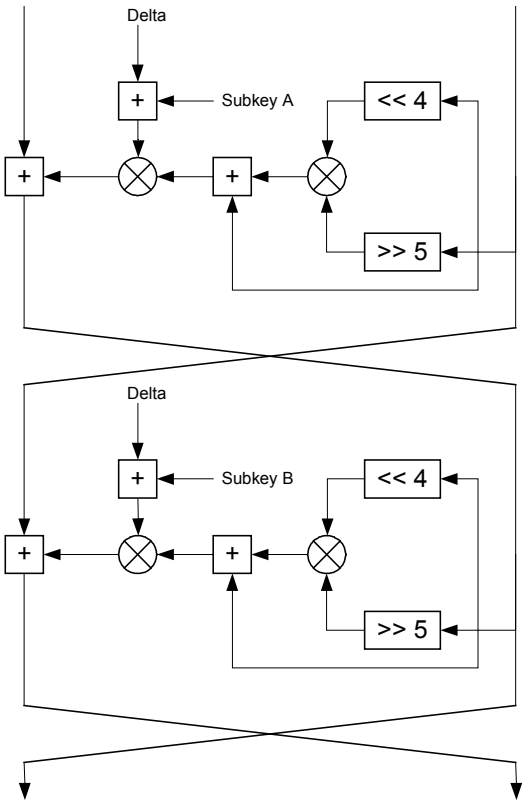


Abbildung 3.6: Darstellung der Arbeitsweise von einer Runde in XTEA.

Im Jahr 1997 entwickelten die beiden Autoren von TEA eine leicht verbesserte Variante des Verfahrens, den Extended Tiny Encryption Algorithm (XTEA). Der Grund für die Weiterentwicklung war eine Anfälligkeit von TEA gegenüber den so genannten *related-key-attacks*.[122] Die wesentlichen Veränderungen bei XTEA betreffen die Erzeugung von Rundenschlüsseln. Im Gegensatz zu der ursprünglichen Variante ist die Funktion bei XTEA etwas komplexer und liefert eine von dem aktuellen Wert von δ abhängige Verteilung der Schlüssel.[123] Die kom-

122. Die Beschreibung der Attacke ist bei Kelsey u. a. [22] zu finden.

123. Vgl. die Zeilen 16, 18, 24 und 26 im Quellcode 3.2.

plette Arbeitsweise einer Runde von XTEA ist in der Abbildung 3.6 zu finden.[124]

124. Eine ausführliche Beschreibung von XTEA ist bei Wheeler u. Needham [53] zu finden.

Sowohl bei TEA als auch bei XTEA wurde auf den Einsatz von Substitutions-Boxen und speziellen Permutations-Tabellen komplett verzichtet. Dadurch ist eine besonders kompakte Darstellung in nahezu allen Programmiersprachen und auf allen Plattformen möglich.[125] Die beeindruckende Einfachheit und Eleganz des Designs von TEA und XTEA wird insbesondere dadurch deutlich, dass der gesamte Quellcode sowohl für die Verschlüsselung als auch für die Entschlüsselung in der Programmiersprache C weniger als 20 Zeilen einnimmt.

125. Vgl. Wheeler u. Needham [54].

```
/* v gives the plain text of 2 words,
k gives the key of 4 words,
N gives the number of cycles, 32 are recommended,
if negative causes decoding, N must be the same as for
    coding,
if zero causes no coding or decoding.*/
tean(long *v, long *k, long N) {
  unsigned long y = v[0], z = v[1], DELTA = 0x9e3779b9;
  if (N > 0) {
    /* coding */
    unsigned long limit = DELTA * N, sum = 0;
    while (sum != limit)
      y += (z << 4 ^ z >> 5) + z ^ sum + k[sum & 3],
      sum += DELTA,
      z += (y << 4 ^ y >> 5) + y ^ sum + k[sum >> 11 & 3];
  }
  else {
    /* decoding */
    unsigned long sum = DELTA * (-N);
    while (sum)
      z -= (y << 4 ^ y >> 5) + y ^ sum + k[sum >> 11 & 3],
      sum -= DELTA,
      y -= (z << 4 ^ z >> 5) + z ^ sum + k[sum & 3];
  }
  v[0] = y, v[1] = z;
  return;
}
```

Quellcode 3.2: Implementierung von XTEA in C.

Der Quellcode 3.2 zeigt eine Referenzimplementierung von XTEA, die von den beiden Autoren implementiert wurde.[126] Als Eingabeparameter (Zeile 6) werden ein Block des Klartextes v, der Schlüssel k sowie die Anzahl der gewünschten Runden N der Funktion übergeben.[127] Der Array für den Klartext gilt gleichzeitig als Rückgabewert, sodass in dieser Variable nach der erfolgten Operation die Ergebnisse gespeichert werden (Zeile 24). Ist der übergebene Parameter N positiv, erfolgt eine Verschlüsselung der übergebenen Daten im v mit dem Schlüssel k für jeweils N Runden (Zeilen 10 bis 15). Ist N negativ, werden die Daten in v entschlüsselt (Zeilen 18 bis 25).

126. Vgl. die Implementierung in Wheeler u. Needham [53, S. 2].

127. Die Größe der Daten beträgt:
v – 64 Bit;
k – 128 Bit;
N – es wird eine Anzahl von 32 Runden empfohlen.

Die Algorithmenfamilie TEA und XTEA ist somit ein sehr gutes Beispiel für einen einfachen und kleinen Algorithmus, der dennoch sehr hohen Standards der Sicherheit genügt. Aufgrund der stärkeren Verbreitung und der besseren Resistenz gegen *related key Angriffe*, wird für die empirische Untersuchung in den folgenden Kapiteln XTEA als Algorithmus betrachtet.

3.2.4 PRESENT

Ein relativ neuer Algorithmus ist der im Jahr 2007 von einem Team aus Wissenschaftlern der Ruhr-Universität Bochum, der Technical University Denmark und der Forschungsabteilung der France Telecom vorgestellte Algorithmus mit dem Namen PRESENT.[128] Das Hauptziel bei der Entwicklung dieser neuen Blockverschlüsselung war der Wunsch, eine möglichst effiziente Implementierung in Hardware zu erreichen. Zudem wurde davon ausgegangen, dass der Algorithmus eher auf kleinen Recheneinheiten Verwendung findet und somit die Handhabung großer Datenbandbreiten nicht zu erwarten ist.[129]

128. Der Name soll eine Ähnlichkeit mit dem Finalisten der AES-Ausschreibung Serpent suggerieren; vgl. Bogdanov u. a. [4, S. 2].

PRESENT ist ein Substitutions-Permutations-Netzwerk mit einer Blocklänge von 64 Bit und einer Rundenzahl von 31. Es werden mit 80 und 128 Bit zwei unterschiedliche Größen von Schlüsseln unterstützt. In jeder Runde werden folgende Operationen durchgeführt: XOR-Verknüpfung mit dem Rundenschlüssel, eine Substitution mit einer Substitutions-Box und eine Permutation. Als letzter Schritt wird der letzte Rundenschlüssel zusätzlich mit dem entstandenen Block mittels XOR verknüpft. Die Abbildung 3.7 liefert eine Darstellung der Arbeitsweise von PRESENT.[130]

129. Eine vollständige Liste der Designkriterien von PRESENT ist bei Bogdanov u. a. [4, S. 5 f.] zu finden.

130. Für eine genaue Beschreibung vgl. Bogdanov u. a. [4].

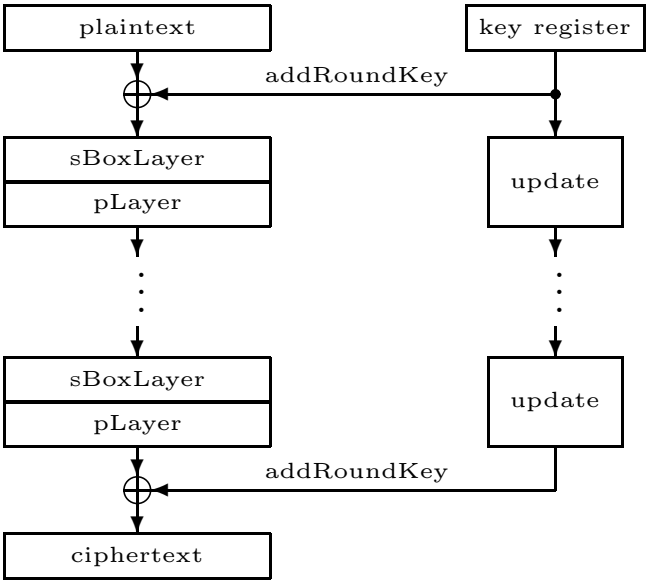


Abbildung 3.7: Darstellung der Arbeitsweise von PRESENT.

Die Erzeugung der Rundenschlüssel geschieht nach dem folgenden Schema in drei Schritten:[131]

131. Es wird die Erzeugung von 80-Bit Rundenschlüsseln gezeigt.

1. Der Schlüssel wird um 61 Positionen nach links bitweise verschoben: $k_i = k_{i-1} << 61$;

2. Die letzten 4 Bits werden durch die Substitutions-Box modifiziert: $k_i[76, 77, 78, 79] = S\,(k_{i-1}[76, 77, 78, 79])$;

3. Einige Stellen im Schlüssel werden mit dem aktuellen Rundenzähler i verknüpft: $k_i[15, 16, 17, 18, 19] = k_{i-1}[15, 16, 17, 18, 19] \oplus i$.

Die Erzeugung von 128 Bit großen Rundenschlüsseln erfolgt nach einem ähnlichen Prinzip.[132]

132. Vgl. Bogdanov u. a. [4, S. Anhang II].

Es gibt nur eine einzige Substitutions-Box in PRESENT und diese transformiert einen 4 Bit großen Eingangswert in einen ebenfalls 4 Bit großen Ausgangswert.[133] Die Substitutionen-Box wird in jeder Runde 16 Mal angewendet. Dabei wird der aktuelle Zustand $b_0 \ldots b_{63}$ der Daten in 16 jeweils 4 Bit große Blöcke $w_0 \ldots w_{15}$ aufgeteilt. Dabei gilt:

133. Eine interne Konstruktion mit nur 4 Bit großen Substitutions-Boxen erlaubt eine effizientere Implementierung in Hardware; vgl. Paar u. Pelzl [37, S. 79].

$$w_i = b_{4 \cdot i+3} \parallel b_{4 \cdot i+2} \parallel b_{4 \cdot i+1} \parallel b_{4 \cdot i}, \quad 0 \leqslant i \leqslant 15$$

Die Werte w_i, die diese Berechnung liefert, werden anschließend

durch die folgenden Werte in der Tabelle 3.4 ersetzt.

w_i	0	1	2	3	4	5	6	7
$S[w_i]$	C	5	6	B	9	0	A	D
w_i	8	9	A	B	C	D	E	F
$S[w_i]$	3	E	F	8	4	7	1	2

Tabelle 3.4: Darstellung der Substitutions-Box bei PRESENT.

Die Permutation der Bits des aktuellen Zustands erfolgt nach dem folgenden Prinzip: Sei i die Position eines Bits im aktuellen Zustand. Dann wird durch die Operation der Bit i an die Position $P(i)$ bewegt, wobei für $P(i)$ gilt:[134]

134. Vgl. Paar u. Pelzl [37, S. 80].

$$P(i) = \begin{cases} (i \cdot 16) \bmod 63 & i \in 0, ..., 62; \\ 63 & i = 63. \end{cases}$$

Diese Konstruktion kann auch als eine Permutationstabelle angegeben werden. Eine Realisierung als Tabelle bzw. feste Verdrahtung kann bei einer Implementierung in Hardware von Vorteil sein.[135]

135. Vgl. Bogdanov u. a. [4, S 4 f.].

Da bei PRESENT die größtmögliche Priorität auf eine schnelle Implementierung in Hardware gelegt wurde, wird der Algorithmus häufig ohne die Operationen für Entschlüsselungen definiert. Die Idee dabei ist, den Algorithmus nur für Anwendungen zu verwenden, die keine explizite Entschlüsselung benötigen. Dazu zählen in erster Linie Authentifizierung sowie Verschlüsselungen im Counter-Modus bzw. im Output Feedback Modus.[136] Die Einfachheit des Designs, die konsequente Verfolgung einer ressourcenschonenden Umsetzung und der Verzicht auf eine Entschlüsselungslogik erlauben es PRESENT in der Version mit 80 Bit großen Schlüsseln mit 1570 GE und bei 128 Bit großen Schlüsseln mit 1.886 GE zu implementieren.[137]

136. Vgl. Abschnitt 2.2.3.

137. GE steht für *gate equivalent*. Das ist eine Methode für die Messung der Hardwarekomplexität von Algorithmen. 1 GE entspricht in etwa dem Aufwand einen NAND-Gatter zu bauen.

Zum Vergleich liegt die Komplexität von DES bei etwa 2.309 GE und von AES-128 bei etwa 3.400 GE. Laut Bogdanov u. a. [4] gibt es aktuell keine Blockverschlüsselung, die kompakter implementiert werden kann als PRESENT. Bei den Stromverschlüsselungen liefert der Algorithmus Grain mit 1294 GE das beste Resultat.[138]

138. S. Bogdanov u. a. [4, S. 13].

3.2.5 RSA

»The Magic Words are Squeamish Ossifrage.«

Entwickler von RSA.[139]

139. Dieser Satz war die verschlüsselte Nachricht, die im Jahr 1977 bei der Publikation von RSA in *Scientific American* als Rätsel von den Entwicklern gestellt wurde.

RSA war der erste öffentlich publizierte Algorithmus, der auf den Ideen der asymmetrischen Kryptografie beruhte. Entwickelt wurde er von den drei Mathematikern Ronald L. Rivest, Adi Shamir und Leonard Adleman im Jahr 1977 am MIT.[140] Der Name des Algorithmus ist von den Anfangsbuchstaben der drei Entwickler abgeleitet. Heute ist RSA der weltweit am häufigsten verwendete asymmetrische Algorithmus in der Kryptografie.[141]

140. MIT – Massachusetts Institute of Technology; vgl. Mao [29, S. 257].

141. Vgl. Schneier [44, S. 532].

Da RSA ein asymmetrischer Algorithmus ist, existieren zwei Schlüssel: ein öffentlicher Schlüssel für die Verschlüsselung und ein privater Schlüssel für die Entschlüsselung. Es gibt im Gegensatz zu den Blockalgorithmen keine Blockgröße und auch keine wiederholte Ausführung von bestimmten Routinen im Algorithmus selbst. Die gesamte Konstruktion basiert auf Einwegfunktionen, in dem Fall auf der Schwierigkeit das Produkt zweier großer Primzahlen zu faktorisieren.[142]

142. Vgl. Originalartikel von den Entwicklern, Rivest u. a. [40].

Der Algorithmus besteht aus zwei wesentlichen Teilen: Aus der Erzeugung der beiden Schlüssel und aus der eigentlichen Verschlüsselung. Es wird zuerst die Erzeugung der Schlüssel vorgestellt. Dabei wird davon ausgegangen, dass Alice und Bob von Anfang an über einen offenen Kanal kommunizieren.[143]

143. Vgl. Mao [29, S. 258].

1. Zuerst wählt Alice zwei zufällige, große Primzahlen p und q, sodass die beiden Primzahlen in etwa gleich groß sind. Die Größenordnung sollte bei modernen Algorithmen bei etwa 1024 Bit oder höher sein.

2. Alice berechnet $N = p \cdot q$;

3. und anschließend $\phi(N) = (p-1) \cdot (q-1)$.

4. Als nächstes wählt Alice eine zufällige Zahl $e < \phi(N)$, sodass $ggT(e, \phi(N)) = 1$ ist.[144]

5. Unter Verwendung des erweiterten Euklidischen Algorithmus wird eine Zahl d gefunden, sodass: $e \cdot d \equiv 1 \bmod \phi(N)$.

144. ggT(a,b) – größter gemeinsamer Teiler von a und b.

6. Nun ist (N, e) der öffentliche Schlüssel von Alice, d ist ihr privater Schlüssel. p, q und $\phi(N)$ werden nicht mehr gebraucht und sollten vernichtet werden.

Die Erzeugung von Schlüsseln ist hiermit abgeschlossen und Bob ist es nun möglich, die Daten m mit dem öffentlichen Schlüssel (N, e) von Alice zu verschlüsseln und die verschlüsselten Daten c an Alice zu senden. Dabei reicht es die folgende Funktion auszuführen:

$$c \leftarrow m^e \bmod N, \quad \text{für } m < N.$$

Bob kann folglich alle Nachrichten bis zur Größe N an Alice versenden. Alice wiederum kann die Nachrichten von Bob mit ihrem privaten Schlüssel d entschlüsseln:

$$m \leftarrow c^d \bmod N.$$

Es ist offensichtlich, dass das Potenzieren von größeren Zahlen, also längeren Nachrichten, aufwendig sein kann und zahlreiche Multiplikationen erfordert. Es gibt allerdings eine Reihe von Algorithmen, z. B. die binäre Exponentiation, die dieses Problem etwas entschärfen.[145] Durch eine geschickte Wahl der Zahl e, kann die Geschwindigkeit der Ausführung der Verschlüsselung nochmals gesteigert werden. Dafür existieren spezielle Werte, die von den Standardisierungsorganisationen für den Einsatz empfohlen werden.[146] Trotzdem ist RSA sowohl als Software- als auch als Hardwareimplementierung etwa um den Faktor 100 bis 1000 langsamer als die meisten symmetrischen Verfahren und eignet sich daher nur bedingt für die Verschlüsselung von größeren Datenmengen.[147]

145. Mehr dazu bei Rivest u. a. [40, S. 8] und Knuth [25, S. 513 ff.].

146. Oft wird 65537 genommen, da die Zahl in der Binärdarstellung nur zwei »1« enthält und so leichteres Potenzieren ermöglicht.

147. Vgl. Schneier [44, S. 535].

3.2.6 Zwischenbilanz

In den vorhergehenden Abschnitten wurden fünf unterschiedliche Verschlüsselungsalgorithmen vorgestellt. Bevor im nächsten Abschnitt eine empirische Untersuchung der Algorithmen auf den ausgewählten Mikrocontrollern durchgeführt und analysiert wird, wird in diesem Abschnitt eine kurze Zusammenfassung gegeben sowie ein Vergleich der formellen Eigenschaften der Algorithmen unternommen.

Bei der Darstellung der unterschiedlichen kryptografischen Algo-

rithmen in den vorhergehenden Abschnitten konnte gezeigt werden, dass diese häufig eine ähnliche Struktur aufweisen und ähnliche zugrunde liegende mathematische Modelle benutzen. Alle untersuchten Blockalgorithmen basieren auf einer wiederholten Ausführung von einfachen Operationen wie Substitutions-Boxen, Permutations-Boxen oder Expansions-Boxen. Der vom Benutzer bereitgestellte Schlüssel wird in einer speziellen Funktion zu mehreren Rundenschlüsseln expandiert, die wiederum mittels XOR mit dem aktuellen Zustand der Daten verknüpft werden.

An dieser Stelle ist es wichtig festzuhalten, wofür die benutzten Operationen verwendet wurden und welche Ziele damit erreicht werden.[148] Die Verwendung von Substitutions-Boxen dient in erster Linie der Erreichung der Nichtlinearität. Diese Eigenschaft ist entscheidend bei der Abwehr von Angriffen, die auf der differentiellen Analyse basieren. Die Verwendung von Permutations-Boxen dient in erster Linie dazu, die Entropie der Klartexte zu erhöhen.[149] Gewöhnlich besteht ein Klartext aus einer Reihe von Redundanzen und bildet nur eine kleine Untermenge der gesamten Menge aller möglichen Klartexte. Durch die Permutations-Boxen wird versucht die Verteilung der gegebenen Klartexte im gesamten Bereich der prinzipiell möglichen Klartexte zu verbessern. Die Verwendung eines geheimen Schlüssels gewährleistet die notwendige geheime Verteilung im Datenblock.

148. S. Mao [29, S. 229 f.] für weitere Details.

149. Entropie ist ein Maß für den mittleren Informationsgehalt eines Zeichens.

Die vorgestellten Algorithmen werden nun unter den folgenden Kriterien miteinander verglichen und bewertet:

Jahr der Publikation: Die Kryptografie durchlebt eine rasante Entwicklung und neue Algorithmen berücksichtigen eher die neueren Erkenntnisse. Auf der anderen Seite sind ältere, schon länger bekannte Algorithmen zum Teil besser und gründlicher von der wissenschaftlichen Gemeinschaft untersucht worden.

Patentierung: Einige Realisierungen von Algorithmen unterliegen Patenten und sind in kommerziellen Anwendungen nur unter der Zahlung einer Lizenz bzw. nur mit Einschränkungen einzusetzen.

Standardisierung: Einige bekannte Algorithmen werden von Standardisierungsbehörden untersucht und in spezielle kryptografische Standards übernommen.

Schlüsselgröße: Die Schlüsselgröße ist ein entscheidender Faktor bei der Bewertung von Brute-Force-Angriffen.

Blockgröße: Bei der Verwendung der Algorithmen als Hashfunktionen bzw. zum Zwecke der Authentifizierung ist die Blockgröße eine wichtiges Designkriterium.

Anzahl der Runden: Damit wird die Anzahl der Runden in der Standardimplementierung angegeben.

Typ des Netzwerks: Je nach Implementierung ist der Blockalgorithmus entweder als ein Feistel-Netzwerk oder als ein Substitutions-Permutations-Netzwerk konstruiert.

Die Tabelle 3.5 bietet eine Übersicht über die vorgestellten Algorithmen. Es wurden mit DES und RSA zwei vergleichsweise ältere Algorithmen, mit TEA und AES zwei Algorithmen aus den 90er Jahren und mit PRESENT ein neuerer Algorithmus untersucht. Alle Algorithmen sind frei von Patenten und können in kommerziellen Anwendungen ohne Einschränkungen eingesetzt werden. Das Patent für den Algorithmus RSA ist im Jahr 2000 ausgelaufen. Bis auf TEA sind zudem alle Algorithmen ein Teil von bestimmten Standards und sind demnach in zahlreichen Produkten implementiert.

Algorithmen:	*DES*	*AES*	*TEA*	*PRESENT*	*RSA*
Publikation:	1977	2000	1994	2007	1977
Patentierung:	nein	nein	nein	nein	nein[150]
Standards:	NBS	NIST	nein	ISO	ISO[151]
Schlüsselgröße:	58	128, 192, 256	128	80, 128	>1024[152]
Blockgröße:	64	128	64	64	—
Rundenzahl:	16	10, 12, 14	64	31	—
Netzwerk:	F[153]	SP	F	SP	—

Tabelle 3.5: Vergleich der vorgestellten Algorithmen zur Verschlüsselung.

150. Patentiert bis 2000 in USA.

151. Für digitale Signaturen.

152. Es wird eine Schlüsselgröße von mindestens 1024 empfohlen.

153. F – Feistel-Netzwerk, und SP – Substitutions-Permutations-Netzwerk.

Vier der untersuchten Algorithmen (DES, AES, TEA, PRESENT) sind symmetrische Verfahren und ein Algorithmus (RSA) stellt eine asymmetrische Verschlüsselung dar. Bei den symmetrischen Algorithmen bewegen sich die Blockgrößen zwischen 64 und 128 Bit. Die Anzahl der Runden weist eine höhere Bandbreite auf und reicht von 10 Runden bei AES-128 bis zu 64 Runden bei TEA. Bei RSA können diese beiden Größen sowie der Typ des Netzwerks nicht angewendet werden.

Die Schlüsselgrößen reichen von eher geringeren 56 Bit bei DES bis zu 256 Bit bei AES. Bei RSA wird für moderne Implementierungen eine Schlüsselgröße von mindestens 1024 Bit empfohlen. Prinzipiell gibt es aber bei RSA keine Begrenzung der eingesetzten Schlüsselgröße, lediglich die Laufzeit verlängert sich mit steigender Größe des Schlüssels. Es ist jedoch zu beachten, dass die Schlüsselgröße bei asymmetrischen Verfahren nicht mit der Schlüsselgröße bei symmetrischen Verfahren gleichzusetzen ist. Gewöhnlich werden bei asymmetrischen Verfahren größere Schlüsseln benötigt, um die gleiche kryptografische Stärke zu erreichen als bei symmetrischen Verfahren. Eine gute Übersicht diesbezüglich liefert die NIST Publikation [33, S. 63].

3.3 Empirische Untersuchung der Algorithmen

Die in den letzten Abschnitten vorgestellten Algorithmen werden im Folgenden in einer empirischen Untersuchung getestet. Anschließend findet eine Analyse der Ergebnisse statt mit dem Ziel, geeignete Algorithmen für eine Implementierung auf den untersuchten Mikrocontrollern zu finden. Ausgehend von den Ergebnissen der empirischen Untersuchung wird eine Entscheidung für eine bestimmte Lösung getroffen, die anschließend in einer konzeptionellen Implementierung umgesetzt wird.

3.3.1 Versuchsaufbau

Für die empirische Untersuchung der Algorithmen wurden die in dem Abschnitt 3.1 dargestellte 8- bzw. 32-Bit Mikrocontroller als Plattform ausgewählt. Die Untersuchung wurde mit dem Ziel durchgeführt folgende Fragen zu beantworten:

- Welche Algorithmen benötigen eine möglichst geringere Größe des Speichers für den Quellcode und für das Kompilat?
- Welche Algorithmen liefern auf den ausgewählten Plattformen die besten Ergebnisse in Bezug auf die Dauer der Verarbeitung und den Datendurchsatz?
- Welche Algorithmen bieten eine gewisse Flexibilität bei dem Kompromiss zwischen der Geschwindigkeit und der Größe des benötigten Speichers?
- Welcher Algorithmus ist unter den gegebenen Bedingungen und Einschränkungen für die Implementierung zu empfehlen?

Folglich wurden Tests auf die Dauer der Verschlüsselung bzw. Entschlüsselung, den Datendurchsatz und der Dauer der Generierung von Schlüsseln durchgeführt. Um statistische Abweichungen und mögliche Störungen bei der Messung ausschließen zu können, erfolgte die Messung der Geschwindigkeit der Algorithmen durch eine wiederholte Ausführung der Funktionen.[154] Für die Auswertung wurden anschließend jeweils die Mittelwerte der Ergebnisse gebildet. Zusätzlich wurden Tests durchgeführt, die eine mögliche Abhängigkeit z. B. in der Laufzeit eines Algorithmus bei der Ausführung von bestimmten Daten zeigen sollen.[155]

Werden kryptografische Algorithmen auf Mikrocontrollern insbesondere mit nur 8-Bit Busbreite eingesetzt, so erfordert dies eine genaue Auswahl der Implementierungen. Wie bereits in den vergangenen Abschnitten gesehen, verwenden die meisten kryptografischen Algorithmen einfache mathematische Operationen wie XOR oder bitweise Verschiebungen bzw. Matrixmultiplikationen.[156] Diese Operationen werden auch von den kleinsten Mikrocontrollern unterstützt. Jedoch verwenden zahlreiche bekannte, größere kryptografische Bibliotheken[157] Funktionen des Betriebssystems bzw. Compilers, die auf Mikrocontrollern zum Teil nicht unterstützt werden. Zum Beispiel beträgt auf einem 8-Bit Mikrocontroller die Größe eines Integers 2 Byte. Eine Implementierung aus einer Bibliothek, die für 32-Bit Plattformen ausgelegt ist, kann aber von 4 Byte großen Integerzahlen ausgehen und folglich unerwartetes Verhalten der Operation hervorrufen.

Bei der Auswahl der Implementierungen wurde daher besonderer Wert auf die Optimierung für den Einsatz auf Mikrocontrollern ge-

154. Es wurden jeweils 1000 komplette Durchläufe der Algorithmen gemessen.

155. Derartige Tests werden oft unter dem Begriff *datenabhängige Berechnungen* zusammengefasst.

156. Bei den asymmetrischen Verfahren z. B. RSA kommen komplexere Operationen zum Einsatz.

157. Damit sind in erster Linie die Software-Bibliotheken wie OpenSSL, Crypto++, JCA gemeint.

legt. Der Quellcode von allen benutzten Algorithmen ist auf der zur Arbeit beiliegenden CD zu finden. Für die Durchführung der Untersuchung wurden die im Folgenden dargestellten Implementierungen ausgewählt:

- DES in der angepassten Implementierung von Stuart Levy [28];
- AES in der Implementierung von Brian Gladman [15];
- XTEA in der leicht abgewandelten Version von David Wheeler und Roger Needham [53];
- PRESENT in der Implementierung von den Wissenschaftlern der Shanghai Jiaotong University [16];
- RSA in der Implementierung von Alfredo A. Ortega [36].

Die Anpassungen in den Quellcodes waren notwendig, um eine reibungslose Kompilierung der Bibliotheken auf Mikrocontrollern zu ermöglichen. Einige der Implementierungen, beispielsweise DES, waren aus dem Jahr 1988 und nicht ohne weiteres mit modernen Compilern kompilierbar. Der Algorithmus XTEA wurde zudem bewusst, statt TEA verwendet, da diese verbesserte Variante einen stärkeren Schutz gegen related-key-Angriffe aufweist.[158] Die im Folgenden dargestellten Ergebnisse der Untersuchung beziehen sich auf die zur Untersuchung herangezogenen Implementierungen der Algorithmen. Es ist denkbar, dass andere Implementierungen gegebenenfalls abweichende Werte auf denselben Mikrocontrollern liefern können.[159]

158. Vgl. Abschnitt 3.2.3.

159. Denkbar sind z. B. speziell auf 8-Bit Mikrocontroller angepasste Implementierungen.

Für die Untersuchung und Analyse wurden spezielle Entwicklungsumgebungen und Werkzeuge benutzt. Für die Durchführung der Testreihen am 8-Bit Mikrocontroller wurde die vom Hersteller der Mikrocontroller bereitgestellte Entwicklungsumgebung CodeWarrior [12] verwendet.[160] Für die Testreihen am 32-Bit Mikrocontroller wurde eine spezielle ebenfalls vom Hersteller der Mikrocontroller angepasste Version der Entwicklungsumgebung Eclipse [11] verwendet. Für die grafische Darstellung der Ergebnisse wurde das Paket PGFPlots verwendet.[161]

160. Der Quellcode zum Versuchsaufbau befindet sich im Anhang 2 auf der Seite 106.

161. Weitere Informationen zu dem Paket sind unter [10] zu finden.

3.3.2 Analyse der Ergebnisse

Die empirische Untersuchung wurde auf den im Abschnitt 3.1 vorgestellten Plattformen durchgeführt. Die fünf ausgewählten Implementierungen der Algorithmen lieferten folgende Ergebnisse:

Größe des Quellcodes und des kompilierten Codes

Die Abbildung 3.8 zeigt die unterschiedlichen Größen des Quellcodes der Algorithmen. Bei der Berechnung der Größe des Quellcodes wurden die Teile des Quellcodes nicht miteinbezogen, die nicht zwingend notwendig für eine korrekte Arbeitsweise des Algorithmus sind. Dazu gehören beispielsweise Kommentare, Hilfsfunktionen bzw. Lizenzbedingungen.[162] Es ist festzustellen, dass AES am meisten Platz für den Quellcode beansprucht. XTEA braucht dagegen etwa 22 Mal weniger Speicherplatz wie AES und somit am wenigsten von allen untersuchten Algorithmen.

162. Bei zahlreichen Softwareimplementierungen stehen die Lizenzbedingungen direkt im Quellcode.

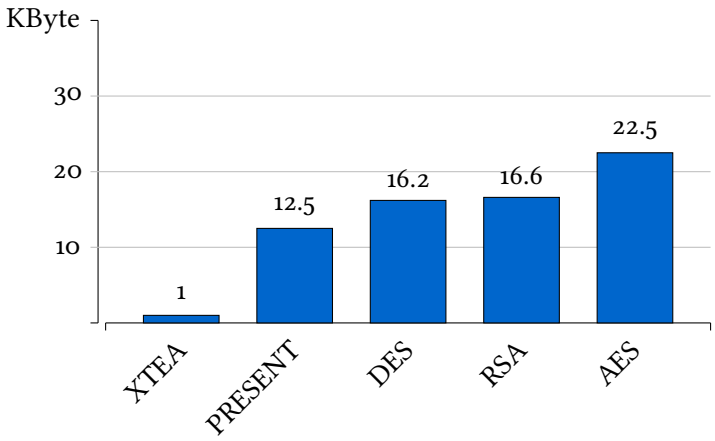


Abbildung 3.8: Größe des Quellcodes der untersuchten Algorithmen.

Neben der Betrachtung der Größe des Quellcodes ist für die Analyse vor allem die Größe des kompilierten Codes von Bedeutung. Der kompilierte Code[163] ist das Resultat der Kompilierung und unterscheidet sich je nach eingesetzten Plattform. Die gemessenen Werte vom kompilierten Code sind in den Abbildungen 3.9 und 3.10 zu finden.

163. Manchmal auch *Kompilat* genannt.

Den mit Abstand kleinsten Quellcode hat der Algorithmus XTEA. Bei der Entwicklung von XTEA wurde besonderer Wert darauf gelegt, einen Algorithmus zu konstruieren, der in der Implementierung möglichst wenig Platz einnimmt. Die Referenzimplementierung der Entwickler in der Programmiersprache C enthält insgesamt weniger

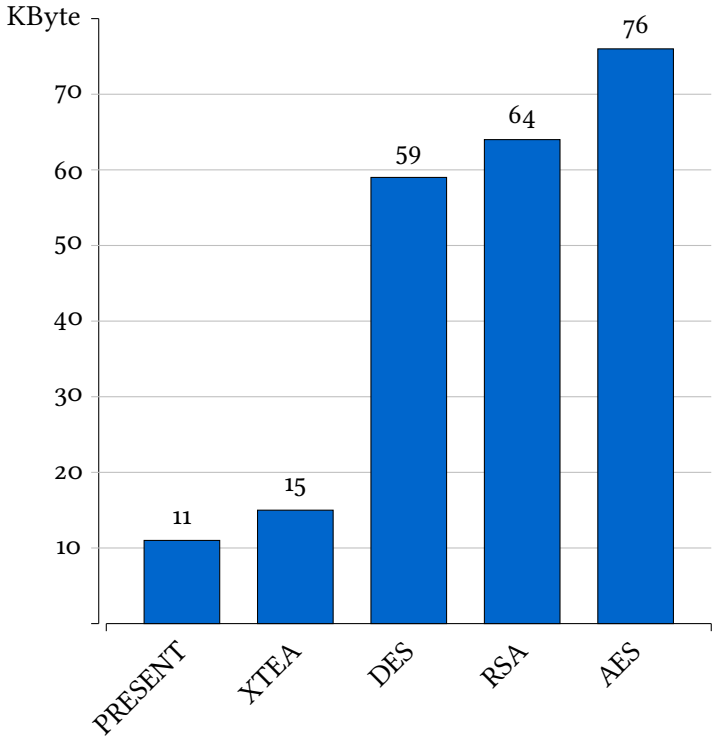


Abbildung 3.9: Größe des kompilierten Codes der untersuchten Algorithmen am 8-Bit Mikrocontroller.

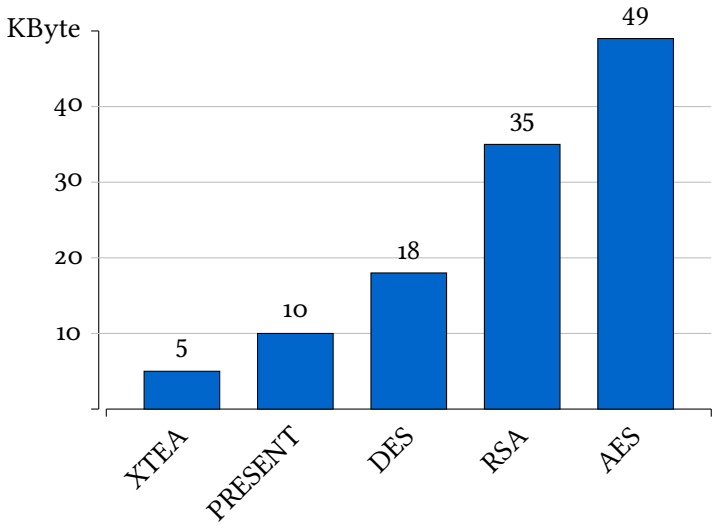


Abbildung 3.10: Größe des kompilierten Codes der untersuchten Algorithmen am 32-Bit Mikrocontroller.

als 1 KByte Quellcode.[164] Jedoch misst das Kompilat von XTEA für den 8-Bit Mikrocontroller 15 KByte. Diese 15-fache Steigerung der Größe des kompilierten Codes beim 8-Bit Mikrocontroller ist im Vergleich zu den anderen Algorithmen überdurchschnittlich.[165] Auf dem 32-Bit Mi-

164. Vgl. den Quellcode 3.2 von XTEA auf der Seite 44.

165. Vgl. die weiteren Ergebnisse in diesem Abschnitt.

krocontroller liegt die kompilierte Codegröße von XTEA bei 5 KByte und hat damit die kleinste Größe.[166]

PRESENT belegt mit etwa 12,5 KByte an Quellcodegröße den zweiten Platz. Eine Besonderheit bei der ausgewählten Implementierung ist der Verzicht auf eine Entschlüsselungsfunktion. Somit kann der Algorithmus entweder nur in einem speziellen Modus für Verschlüsselungen benutzt werden oder der Einsatz beschränkt sich auf die Prüfung der Integrität der Daten bzw. auf die Erzeugung von Hashzahlen. Wichtig ist festzuhalten, dass der kompilierte Code mit 11 KByte auf dem 8-Bit Mikrocontroller am wenigsten Platz benötigt.

DES belegt mit 16.2 KByte Quellcodegröße den mittleren Platz. Die 8 unterschiedlichen Substitutions-Boxen belegen etwa 17 % des Speicherplatzes. Durch zusätzliche Operationen wird in der aktuellen Implementierung die Funktionalität auf 8-Bit Plattformen gesichert. Die Größe des Kompilats liegt mit 59 KByte für den 8-Bit und mit 18 KByte für den 32-Bit Mikrocontroller ebenfalls im mittleren Bereich.

RSA als asymmetrisches Verfahren besitzt keine als Arrays gespeicherten Tabellen für die Durchführung der Substitutionen und besteht in der gewählten Implementierung fast ausschließlich aus Funktionen, die mathematische Operationen umsetzen. Die gewählte Implementierung beinhaltet spezielle Routinen für den Einsatz am 8-Bit Mikrocontroller. Dies vergrößert den Quellcode zusätzlich. Der kompilierter Code für den Einsatz auf dem 8-Bit Mikrocontroller belegt etwa vier Mal mehr Platz als der Quellcode.[167]

AES hat mit 22.5 KByte den größten Quellcode und auch das größte Kompilat. Es ist zu beachten, dass bei zahlreichen Algorithmen insbesondere auch bei AES die Möglichkeit besteht, die Größe des Quellcodes zu verkleinern. Dafür werden in der Regel die Substitutions-Boxen nicht fest als Arrays definiert und im Quellcode gespeichert, sondern es wird numerisch der entsprechende Wert einer Substitutions-Box ausgerechnet.[168] In der Regel ist die numerische Berechnung aufwendiger und nimmt daher mehr Zeit in Anspruch als ein Ersetzen eines Wertes aus einer Array-Tabelle. Auf der anderen Seite wird dadurch Platz eingespart und der Kompilat belegt danach weniger Speicherplatz. Bei der untersuchten Implementierung von AES belegen die Substitutions-Boxen etwa 33 % des gesamten Quellcodes.[169] Der kompilierte Code misst für den Einsatz auf dem 8-Bit Mikrocontroller 76 KByte und 49 KByte für den Einsatz auf dem 32-Bit Mikrocontroller.

166. Die Größe des Kompilats ist auf die internen Operationen zurückzuführen. Bei der Kompilierung werden diese Operationen in einfachere aufgebrochen, die vom Mikrocontroller verarbeitet werden können. Die resultierende Menge der Operationen belegt somit mehr Platz als die nicht kompilierte Variante.

167. Die benötigte Größe des Speichers liegt bei 64 KByte. Bei dem 32-Bit Mikrocontroller wird etwa 35 KByte Speicher benötigt.

168. Gilt ebenfalls für Permutations-Boxen, vgl. Abschnitt 3.2.4.

169. S. den Quellcode von AES auf der beigelegten CD.

Dauer der Verschlüsselung und Entschlüsselung

Bei der Dauer der Verschlüsselung muss zwischen den zwei Mikrocontrollern unterschieden werden. Alle untersuchten Algorithmen haben auf dem 32-Bit Mikrocontroller deutlich bessere Resultate gezeigt als auf dem 8-Bit. Neben der größeren Taktrate ist die größere Busbreite beim 32-Bit Mikrocontroller entscheidend für die schnellere Verarbeitungsgeschwindigkeit. Bei den Untersuchungen konnte zeitlich bis auf 1 Millisekunde genau gemessen werden. Eine feinere Zeitmessung konnte mit der gegebenen Hardware nicht realisiert werden.

Bei Algorithmen, die eine kürzere Dauer der Ausführung lieferten als 1 mSec,[170] wurden mehrere Durchläufe des Algorithmus durchgeführt und anschließend wurde das Ergebnis durch die Anzahl der Durchläufe dividiert. Die Ergebnisse der Verschlüsselung von jeweils einem Block an Daten sind in der Abbildung 3.11 für den 8-Bit Mikrocontroller und in der Abbildung 3.12 für den 32-Bit Mikrocontroller zusammengefasst.[171] Es ist zu beachten, dass bei AES die Blockgröße 128 Bit beträgt, bei PRESENT, DES und XTEA dagegen 64 Bit.

170. Insbesondere auf dem 32-Bit Mikrocontroller.

171. Für die Untersuchung wurde beim 8-Bit Mikrocontroller die Taktung auf 2 MHz gesetzt. Beim 32-Bit Mikrocontroller wurde die Taktung auf 4 MHz festgelegt.

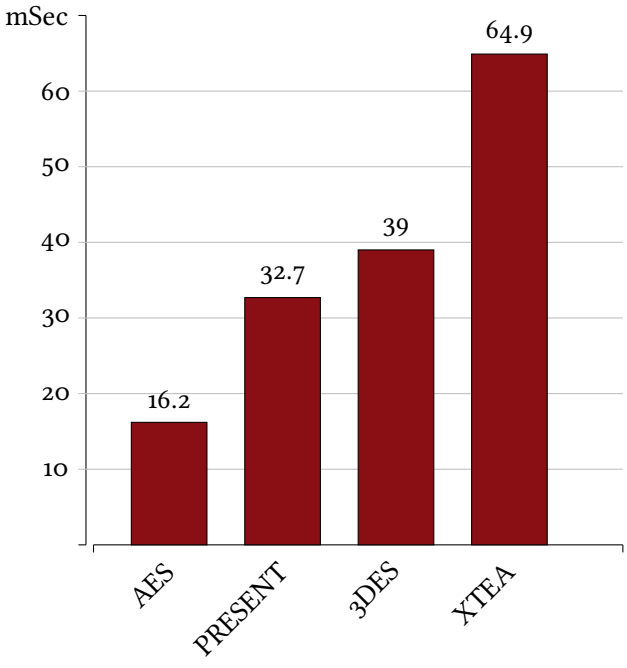


Abbildung 3.11: Dauer der Verschlüsselung eines Blocks auf dem 8-Bit Mikrocontroller.

Bei den Untersuchungen auf dem 8-Bit Mikrocontroller war die kürzeste Dauer für die Verschlüsselung von einem Block bei dem Algorithmus AES zu beobachten.[172] Mit 16,2 Millisekunden war der Wert

172. Es wurde AES mit einer Schlüssellänge von 128 Bit verwendet.

etwas mehr als doppelt so klein wie beim zweitschnellsten Algorithmus PRESENT. Die guten Resultate von AES beruhen auf der Tatsache, dass AES mit dem Ziel implementiert wurde, auch auf kleinsten Recheneinheiten gute Resultate zu erzielen. Beim Auswahlprozess von AES war eine gute Möglichkeit der Implementierung auf 8-Bit Mikrocontrollern eines der Kriterien.[173] Auf dem 32-Bit Mikrocontroller lag die Dauer der Verschlüsselung von einem Block mit AES bei etwa 0,25 Millisekunden und somit etwa 65 Mal kürzer als auf dem 8-Bit Mikrocontroller. Allerdings sind die Werte von AES auf dem 32-Bit Mikrocontroller im Vergleich zu den anderen Algorithmen unterdurchschnittlich ausgefallen: AES ist in Bezug auf die Dauer der Verschlüsselung vom ersten auf den dritten Platz gefallen.

173. Vgl. Abschnitt 3.2.2 über AES.

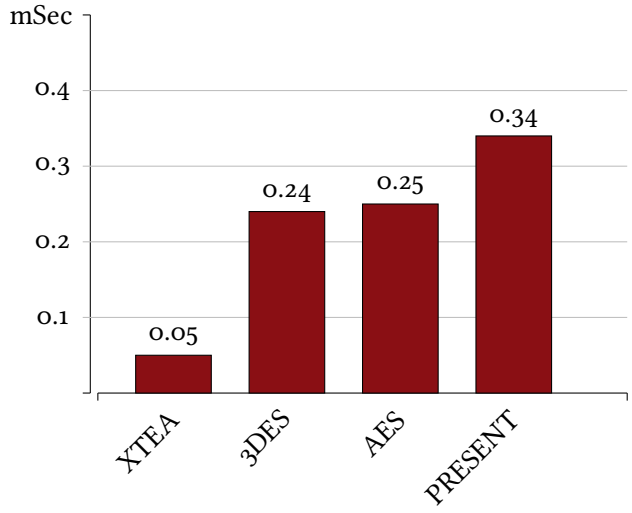


Abbildung 3.12: Dauer der Verschlüsselung eines Blocks auf dem 32-Bit Mikrocontroller.

Um eine Vergleichbarkeit in Bezug auf die kryptografische Sicherheit beim Algorithmus DES zu den anderen Algorithmen zu erhalten, wurde stets die dreifache Anwendung von DES, die 3DES untersucht.[174] Bei den Tests wurde die Variante von 3DES in der Version ENC-DEC-ENC mit jeweils drei unterschiedlichen Schlüsseln und einer effektiven Schlüssellänge von 112 Bit ausgewählt.[175] Die gewählte 3DES Variante ist um den Faktor 3 langsamer als die einfache DES Variante. Genaue Resultate von DES und 3DES sind in der Tabelle 3.6 zu finden. Insbesondere auf dem 32-Bit Mikrocontroller waren die Resultate von 3DES überdurchschnittlich gut. 3DES zeigte eine um den Faktor 160 kürzere Dauer der Verschlüsselung als auf dem 8-Bit Mikrocontroller.

174. Die Schlüsselgröße von 56 Bit bei DES gilt nicht mehr als sicher.

175. Vgl. Schneier [44, S. 342].

	DES	3DES
8-Bit MC	13	39
32-Bit MC	0,08	0,24

Tabelle 3.6: Dauer der Verschlüsselung in mSec von DES und 3DES.

Dafür ist insbesondere die interne Struktur und der Aufbau von DES verantwortlich.[176]

176. Vgl. Abschnitt 3.2.1 und die Seite 60 in diesem Abschnitt.

Die Verschlüsselung eines Blocks mit PRESENT dauerte auf dem 8-Bit Mikrocontroller im Schnitt 32.7 Millisekunden. Wie im Abschnitt 3.2.4 bereits dargestellt, lag der Hauptfokus bei der Entwicklung von PRESENT auf einer einfachen und schnellen Implementierung in Hardware. Bei der ausgewählten Software-Implementierung von PRESENT konnten demnach keine klaren Vorteile gegenüber anderen untersuchten Algorithmen in Bezug auf die Dauer der Verschlüsselung festgestellt werden. Auf dem 32-Bit Mikrocontroller war die Dauer der Verschlüsselung mit etwa 0.34 Millisekunden etwa 100 Mal schneller als auf dem 8-Bit Mikrocontroller. Trotzdem belegte PRESENT auf dem größeren der beiden Mikrocontroller in Bezug auf die Dauer der Verschlüsselung von einem Block den letzten Platz unter allen untersuchten Blockalgorithmen.

XTEA belegte mit knapp 65 Millisekunden den letzten Platz unter den Blockalgorithmen auf dem 8-Bit Mikrocontroller. Dieselbe Implementierung von XTEA zeigte dagegen auf dem 32-Bit Mikrocontroller die mit Abstand besten Resultate. Die Dauer der Verschlüsselung von einem Block von Daten lag bei 0,05 Millisekunden. Im Vergleich zu den Ergebnissen auf dem 8-Bit Mikrocontroller stellt dies eine um den Faktor 1.300 kürzere Ausführungszeit dar.

Diese beachtlichen Resultate von XTEA sind auf die interne Struktur des Algorithmus zurückzuführen. XTEA ist als ein Feistel-Netzwerk implementiert und jede Runde wird aus zwei 32-Bit großen Teilblöcken konstruiert.[177] Ein 32-Bit großer Teilblock kann auf einem 32-Bit großen Mikrocontroller in einem Schritt aus dem EEPROM bzw. RAM in die Register des Prozessors geladen und verarbeitet werden. Der auf beispielsweise 8-Bit Architekturen notwendige zweite, dritte und vierte Zugriff auf die entsprechenden Speicherstellen entfällt dadurch komplett. Ist die Verarbeitung der Operation abgeschlossen, kann wiederum in einem Speicherzugriff der komplette Inhalt eines Teilblocks verarbeitet und gespeichert werden. Pro Runde sind folglich bis zu sechs Speicherzugriffe auf den RAM weniger notwendig, als dies bei einer 8-Bit Architektur der Fall ist. DES ist intern ebenfalls auf einem Feistel-Netzwerk mit einer Größe des Teilblocks von 32-Bit aufgebaut. Folglich gelten für diesen Algorithmus die gleichen Überlegungen. Daher haben sowohl XTEA als auch DES bei den Untersuchungen

177. Vgl. Abschnitt 3.2.3.

auf dem 32-Bit Mikrocontroller überdurchschnittlich gute Resultate gezeigt und jeweils die ersten zwei Plätze belegt.

Bei der Messung der Dauer der Entschlüsselung bei den Algorithmen XTEA und DES konnten keine signifikanten Abweichungen zu der Dauer der Verschlüsselung festgestellt werden. Bei PRESENT gibt es in der gewählten Implementierung nur die Verschlüsselungsoperation.[178] Für die gleiche Dauer der Verschlüsselung und der Entschlüsselung sind im Wesentlichen folgende Punkte entscheidend:

178. Vgl. Abschnitt 3.2.4.

- sowohl XTEA als auch DES sind Feistel-Chiffren;
- bei den Feistel-Chiffren erfolgt die Entschlüsselung durch die Anwendung der Rundenfunktion in umgekehrter Reihenfolge.

Daher wird auf eigene Diagramme für die Dauer der Entschlüsselungsoperationen verzichtet. Die Rohdaten zu der Dauer der Entschlüsselung sind in Tabelle 3.9 auf Seite 68 zu finden.

Beim Algorithmus AES war eine Abweichung bei der Dauer der Entschlüsselung messbar. Die Entschlüsselung auf dem 8-Bit Mikrocontroller lieferte eine etwa um 3 % von der Dauer der Verschlüsselung abweichende Messung. Beim 32-Bit Mikrocontroller war die Abweichung mit 8 % sogar noch größer. Die Tabelle 3.7 zeigt die exakten Werte der Untersuchung. Die Entschlüsselung dauert sowohl auf dem 8-Bit Mikrocontroller als auch und auf dem 32-Bit Mikrocontroller länger als die Verschlüsselung.

	ENC	DEC
8-Bit MC	16,2	16,8
32-Bit MC	0,25	0,27

Tabelle 3.7: Dauer der Verschlüsselung und Entschlüsselung in mSec von AES-128 auf 8-Bit und 32-Bit Mikrocontrollern. Dabei bedeutet *ENC* die Verschlüsselung und *DEC* die Entschlüsselung.

Der Grund für diese Abweichung ist der interne Aufbau von AES.[179] AES, als ein Substitutions-Permutations-Netzwerk, besitzt zur Durchführung einer Entschlüsselung für jede Operation ein Inverses. Speziell bei AES ist jedoch die inverse Operation zu MixColums aufwendiger als die eigentliche MixColumns Operation:[180]

179. Vgl. Abschnitt 3.2.2.

180. Vgl. Hamburg [19, S. 11].

$$C = P \cdot M = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \end{pmatrix}$$

Genaue Beschreibung und Herleitung der Matrix ist beispielsweise bei Haan [18] zu finden. Folglich dauert die Entschlüsselung länger als die Verschlüsselung und führt zur gemessenen Abweichung.

Untersuchung von RSA

Zur besseren Lesbarkeit wurde in den Diagrammen 3.11 und 3.12 darauf verzichtet die Werte für RSA direkt einzutragen.[181] Die durchgeführten Tests lieferten für die Verschlüsslung und Entschlüsselung von 512 Bit großen Daten mit einem Schlüssel von ebenfalls 512 Bit folgende in Tabelle 3.8 dargestellte Ergebnisse:

181. Die Werte für RSA würden ansonsten die Diagramme optisch stark verzerren.

	Verschlüsselung in mSek	Entschlüsselung in mSek
RSA-512, 8-Bit MC:	1820	142800
RSA-512, 32-Bit MC:	215	16648

Tabelle 3.8: Dauer der Verschlüsselung und Entschlüsselung bei RSA-512 auf 8-Bit und 32-Bit Mikrocontrollern.

Insbesondere die Durchführung der Operationen auf dem 8-Bit Mikrocontroller beanspruchten im Vergleich zu den Blockalgorithmen deutlich mehr Zeit. Die Verschlüsselung mit RSA-512 dauert auf dem 8-Bit Mikrocontroller etwa 1820 Millisekunden. Dies entspricht einer Dauer von etwa 1,8 Sekunden. Dieser Wert ist etwa um den Faktor 27 größer als die langsamste zuvor untersuchte Blockverschlüsselung.[182] Die Entschlüsselung mit RSA-512 beansprucht noch mehr Zeit und dauert etwa 2,38 Minuten. Die Entschlüsselung mit RSA nimmt folglich deutlich mehr Zeit in Anspruch als die Verschlüsselung. Das ist auf die Konstruktion von RSA zurückzuführen und basiert auf der Tatsache, dass der für die Verschlüsselung gewählte Exponent e ein schnelleres Potenzieren ermöglicht als der zum Entschlüsseln benutzte Exponent d.[183] In der aktuellen Untersuchung lag der Unterschied zwischen der Dauer der Verschlüsselung und der Dauer der Entschlüsselung etwa beim Faktor 78 sowohl für den 8-Bit als auch für den 32-Bit Mikrocontroller.

182. Vgl. Tabelle 3.11.

183. Vgl. Abschnitt 3.2.5 über RSA.

Es ist zudem zu beachten, dass die gewählte Schlüsselgröße von 512 Bit nicht den modernen Standards kryptografischer Sicherheit und den geforderten Mindestlängen von Schlüsseln bei asymmetrischen Verfahren entspricht. Um derzeit das gleiche Niveau an Sicherheit bei RSA zu erreichen, wie es mit einem 128 Bit langen Schlüssel bei einem symmetrischen Verfahren gegeben ist, werden mindestens 2.304 Bit lange Schlüsseln verlangt.[184] Die Verarbeitung von derartig langen Schlüssel

184. Vgl. Schneier [44, S. 194].

war jedoch auf den untersuchten Plattformen nicht möglich.[185] Aus diesen Überlegungen und auch aufgrund der deutlich längeren Zeiten für den Durchlauf einer Operation geht hervor, dass RSA für den produktiven Einsatz auf den untersuchten Mikrocontrollern eher nicht geeignet.

185. Die größtmögliche Länge des Schlüssels betrug auf dem 8-Bit Mikrocontroller 512 Bit.

Berechnung des Datendurchsatzes

Als weiteres Kriterium bei der Untersuchung der Algorithmen wurde der Durchsatz pro Sekunde gemessen. Als Berechnungsgrundlage wurde die Dauer der Verschlüsselung genommen. Der Datendurchsatz pro Sekunde wird demnach wie folgt ausgerechnet:

$$\text{Durchsatz}_{\text{Sek}} = \frac{1000}{\text{Dauer}_{\text{mSek}}} \cdot \text{Blockgröße}$$

Die Dauer der Verschlüsselung und die Blockgröße sind aus früheren Untersuchungen bereits bekannt. Bei den Algorithmen DES, XTEA und PRESENT beträgt die Blockgröße 64 Bit, bei AES liegt die Blockgröße bei 128 Bit. Bei RSA wurde mit Nachrichten von der Länge 512 Bit gearbeitet. Folglich ergeben sich die in den Abbildungen 3.13 und 3.14 dargestellten Ergebnisse.

Auf dem 8-Bit Mikrocontroller liefert AES mit 7.9 Kilobit (KBit) den mit Abstand höchsten Datendurchsatz für Verschlüsselung pro Sekunde. Der hohe Durchsatz ist vor allem auf die geringe Dauer für die Ausführung einer Verschlüsselung zurückzuführen sowie auf die im Vergleich zu anderen Algorithmen größere Blockgröße von 128 Bit. PRESENT und 3DES liefern mit jeweils 1,96 und 1,64 ähnliche Resultate.[186] Bei XTEA liegt der Datendurchsatz mit 970 Bit bei knapp unter einem Kilobyte pro Sekunde und somit etwa um den Faktor 8 geringer als bei AES. Bei RSA ergab die Untersuchung einen Durchsatz von etwa 280 Bit pro Sekunde ergibt und somit den geringsten Wert auf dem 8-Bit Mikrocontroller.

186. Der Algorithmus DES in der Ausführung mit nur einem Schlüssel liefert 3 Mal so hohe Datendurchsätze; vgl. den Abschnitt 3.3.2.

Auf dem 32-Bit Mikrocontroller ergab die Untersuchung bei XTEA mit 1,28 Megabit (MBit) den höchsten Datendurchsatz. Die deutlich kürzere Dauer der Verschlüsselung auf dem 32-Bit Mikrocontroller als auf dem 8-Bit Mikrocontroller war bei XTEA für dieses Resultat entscheidend. AES belegt mit 512 KBit den zweiten Platz. 3DES und PRESENT landen mit jeweils 266 und 188 KBit pro Sekunde auf dem

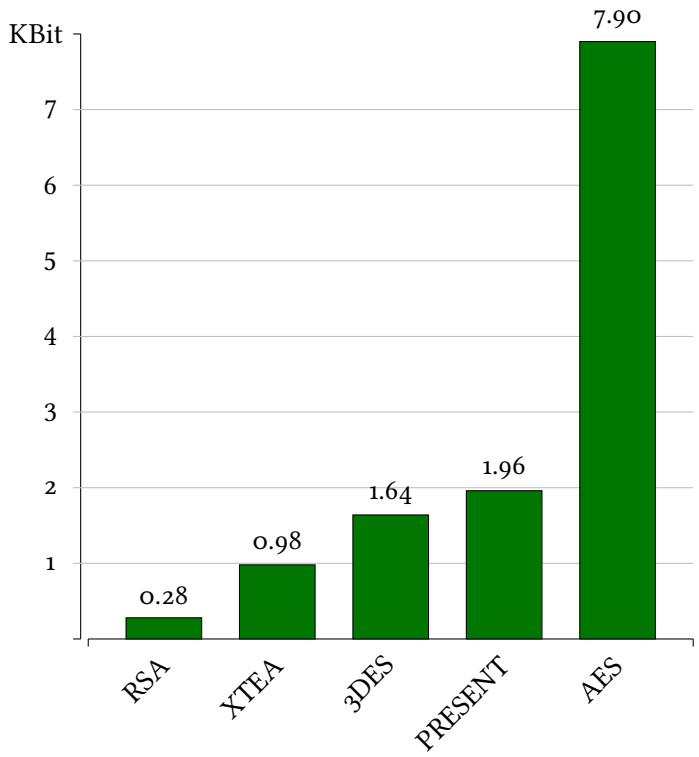


Abbildung 3.13: Der Datendurchsatz für Verschlüsselung in KBit pro Sekunde auf dem 8-Bit Mikrocontroller.

dritten und vierten Platz. Auf dem fünften Platz mit nur 2,38 KBit pro Sekunde liegt RSA.[187] Damit belegt RSA mit einem großen Abstand zu den anderen Algorithmen den letzten Platz. Diese Untersuchung bestätigt die anfängliche Vermutung über den geringen Datendurchsatz von RSA im Vergleich zu symmetrischen Blockalgorithmen.[188] Der Datendurchsatz von RSA ist auf dem 32-Bit Mikrocontroller etwa 530 Mal geringer als bei XTEA und etwa 215 Mal geringer als bei AES. Die Werte für den Datendurchsatz bei RSA sind somit sowohl auf dem 8-Bit Mikrocontroller als auch auf dem 32-Bit Mikrocontroller für einen produktiven Einsatz viel zu gering.

187. Bei Messung des Datendurchsatzes für Entschlüsselung würde der Wert noch geringer ausfallen.

188. Vgl. Abschnitt 3.2.5.

3.3.3 Datenabhängige Berechnungen

Im Folgenden wird in einer weiteren Untersuchung gemessen, inwiefern sich die Dauer der Verschlüsselung ändert, wenn die zu verschlüsselnden Daten sich ebenfalls verändern. Für die Bestimmung einer möglichen Abhängigkeit der Dauer der Verschlüsselung von dem jeweiligen Algorithmus wurde der Hamming-Abstand als Kriterium für

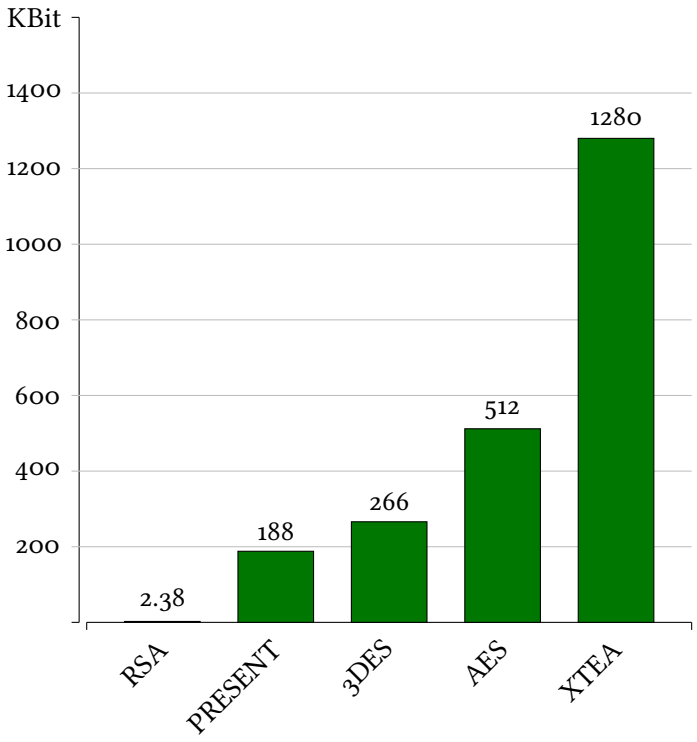


Abbildung 3.14: Der Datendurchsatz für Verschlüsselung in KBit pro Sekunde auf dem 32-Bit Mikrocontroller.

die Messung gewählt. Der Hamming-Abstand [189] ist definiert als:

$$\triangle(x, y) = \sum_{x_i \neq y_i} 1, \quad i = 1, ..., n$$

wobei i der jeweiligen Stelle in der Bitfolge der Daten x und y in der Binärdarstellung entspricht. Folglich ist der Hamming-Abstand umso höher, je stärker sich die jeweiligen Daten voneinander unterscheiden. Bei der Bestimmung des Hamming-Abstands werden die Daten dabei jeweils bitweise verglichen. Der Abstand von dem Vektor »0« wird häufig als Hamming-Gewicht bezeichnet.[190]

Da viele untersuchte Algorithmen eine Blockgröße von 64 Bit haben, wurde bei den durchgeführten Untersuchungen 64 als das größtmögliche Hamming-Gewicht eines Datenblocks angenommen. Bei AES beträgt die Blockgröße 128 Bit, folglich hätten bei dem Algorithmus größere Hamming-Gewichte pro Block gemessen werden können. Es wurden bei der Messung Daten mit insgesamt 16 unterschiedlichen Hamming-Gewichten von 0 bis 64 untersucht.[191]. Dadurch konnte eine mögliche Abhängigkeit z. B. in der Dauer der Berechnung von

189. Benannt noch dem amerikanischen Mathematiker Richard Wesley Hamming (1915–1998).

190. Vgl. Daemen u. Rijmen [6, S. 17 f.].

191. Von 0 angefangen wurde in jedem Schritt das Hamming-Gewicht um 4 erhöht: 0, 4, 8...60, 64.

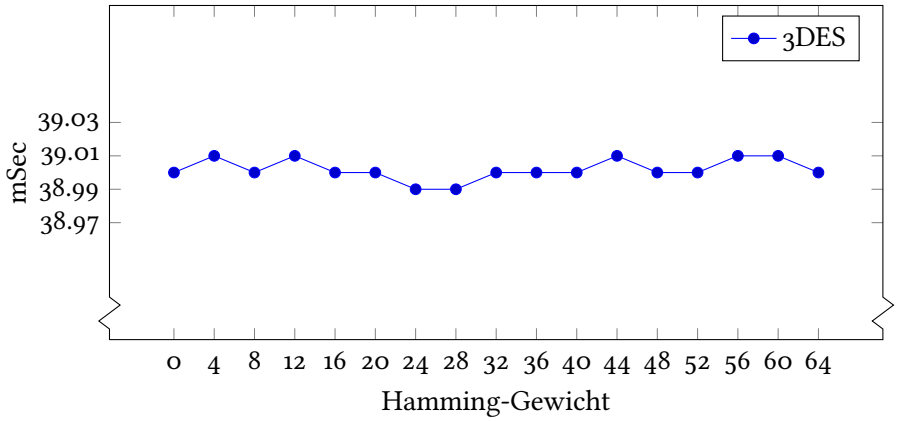


Abbildung 3.15: Dauer der Verschlüsselung mit unterschiedlichen Hamming-Gewichten mit 3DES auf dem 8-Bit Mikrocontroller.

Daten bei steigenden Hamming-Gewichten gemessen werden. Diese Betrachtung liefert jedoch keine vollständige Untersuchung, um beispielsweise Seitenkanalangriffe auf den Algorithmus oder die Hardware ausschließen zu können. Für eine derartige Untersuchung sind weitere umfangreiche Tests notwendig, die unter anderem die Unterschiede beim Energieverbrauch oder der Dauer der Erzeugung von Schlüsseln messen.

Es wurden die vier vorgestellten Blockalgorithmen analysiert, wobei jeweils die Klartextdaten und der Schlüssel pro Messung variiert wurde. Bei den vier Blockalgorithmen sind minimale Abweichungen in der Dauer der Berechnung festzustellen, jedoch war keine signifikante Tendenz erkennbar. In den Abbildungen 3.15, 3.16, 3.17 und 3.18 sind die Ergebnisse der Verschlüsselung mit unterschiedlichen Klartextdaten auf dem 8-Bit Mikrocontroller abgebildet. Bei einer Veränderung des Schlüssels gab es ebenfalls keine signifikante Abweichung, sodass auf diese Darstellungen verzichtet wird.[192]

192. Mit einer feineren Messtechnik sind unter Umständen genauere Ergebnisse zu erzielen.

Beim Algorithmus RSA konnten bei der durchgeführten Messung keine verifizierbaren Ergebnisse ermittelt werden. Die Abweichungen bei den Versuchen waren oft stark und wiesen zudem keine Konstanz auf. So war beispielsweise bei den ersten Messungen die Dauer bei kleineren Hamming-Gewichten etwas geringer als bei größeren Hamming-Gewichten. Bei der anschließenden Wiederholung der Messung gab es jedoch gegensätzliche Ergebnisse.[193] Mögliche Gründe für diese Abweichungen sind die deutlich längere Dauer der einzelnen Ausführung bei RSA und die damit verbundene größere Abweichung und eine geringe Busbreite speziell auf dem 8-Bit Mikrocontroller.

193. In einer weiteren durchgeführten Messung wurde eine längere Dauer bei kleinen Hamming-Gewichten festgestellt.

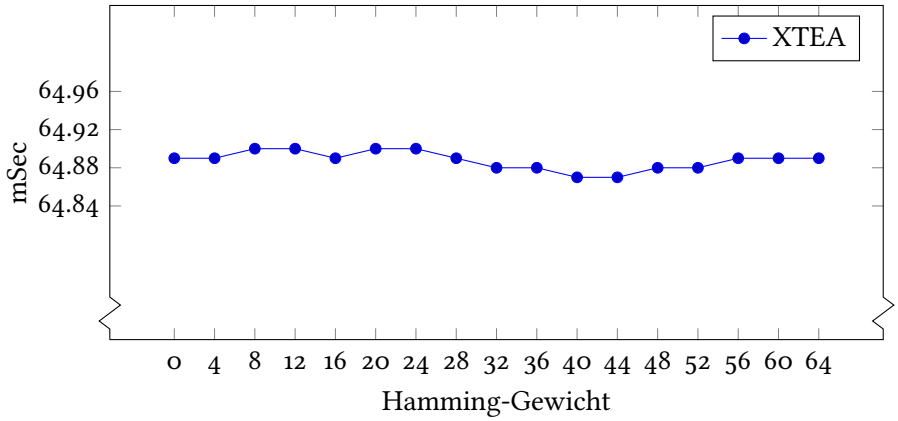


Abbildung 3.16: Dauer der Verschlüsselung mit unterschiedlichen Hamming-Gewichten mit XTEA auf dem 8-Bit Mikrocontroller.

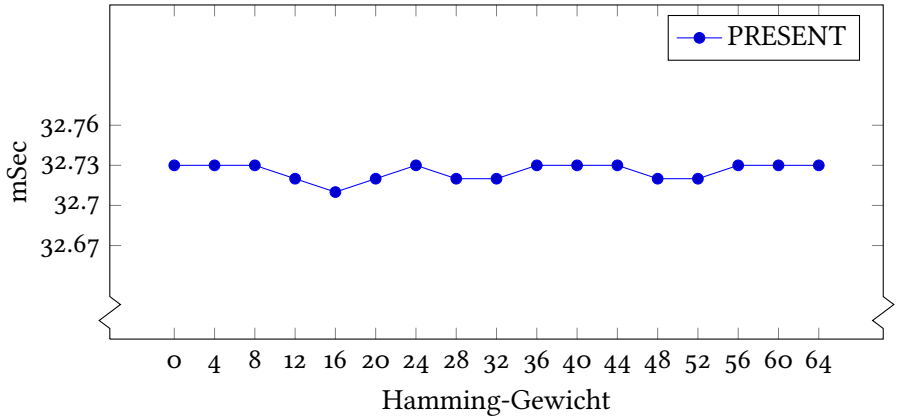


Abbildung 3.17: Dauer der Verschlüsselung mit unterschiedlichen Hamming-Gewichten mit PRESENT auf dem 8-Bit Mikrocontroller.

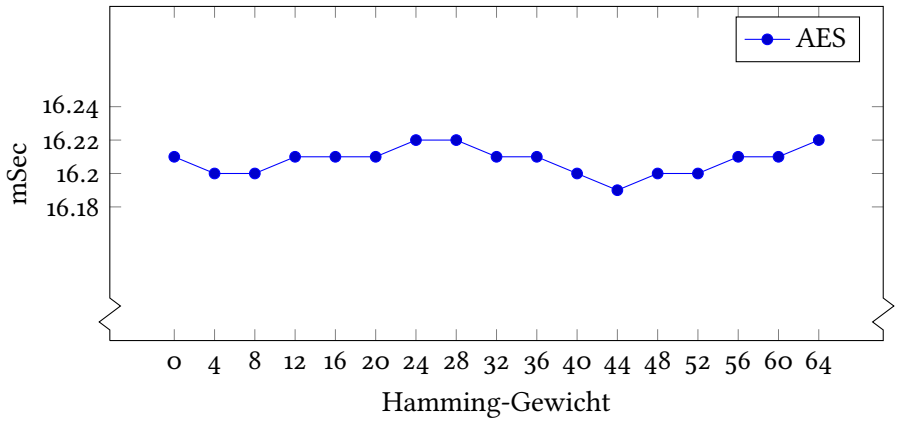


Abbildung 3.18: Dauer der Verschlüsselung mit unterschiedlichen Hamming-Gewichten mit AES-128 auf dem 8-Bit Mikrocontroller.

3.4 Zusammenfassung der Ergebnisse

In diesem Abschnitt werden die wichtigsten Ergebnisse der durchgeführten empirischen Untersuchung zusammengefasst. Die Tabelle 3.9 gibt einen Überblick über die erhaltenen nummerischen Werte. Die zentralen Ergebnisse der Untersuchung sind:

Algorithmen	*3DES*	*AES*	*XTEA*	*PRESENT*	*RSA*
Größe des Quellcodes in KBit (8 Bit)	16,2	22,5	1	12,5	16,6
Dauer der Verschlüsselung in mSec (8-Bit)	39	16,2	64.9	32,7	1820
Dauer der Entschlüsselung in mSec (8-Bit)	39	16,8	64.9	32,7	142800
Datendurchsatz in KBit pro Sekunde (8-Bit)	1,64	7,90	0,98	1,96	0,28
Größe des Quellcodes in KBit (32 Bit)	16,2	22,5	1	12,5	16,6
Dauer der Verschlüsselung in mSec (32-Bit)	0,24	0,25	0,05	0,34	215
Dauer der Entschlüsselung in mSec (32-Bit)	0,24	0,27	0,05	0,34	16648
Datendurchsatz in KBit pro Sekunde (32-Bit)	266	512	1280	188	2,38

Tabelle 3.9: Übersicht über die Ergebnisse der durchgeführten Untersuchungen auf den 8-Bit und 32-Bit Mikrocontrollern.

- Der Algorithmus XTEA belegt am wenigsten Speicher für den Quellcode und am wenigsten Speicher für das Kompilat auf dem 32-Bit Mikrocontroller. Auf dem 8-Bit Mikrocontroller belegte das Kompilat von PRESENT am wenigsten Speicher. AES benötigte am meisten Speicherplatz sowohl für den Quellcode als auch für das Kompilat.

- Die Dauer der Ausführung auf dem 8-Bit Mikrocontroller und auf dem 32-Bit Mikrocontroller unterscheidet sich bei einigen Algorithmen sehr stark. So belegt XTEA auf dem 8-Bit Mikrocontroller den letzten Platz und auf dem 32-Bit Mikrocontroller dagegen den ersten Platz. Diese Unterschiede sind auf die interne Struktur der Algorithmen zurückzuführen.

- XTEA und DES zeigen besonders gute Resultate auf dem 32-Bit Mikrocontroller, da die interne Zustandsrepräsentation in einer 32-Bit großen Feistel-Runde und in einem gleichgroßen Bussystem vergleichsweise schnell, ohne zusätzliche Speicherzugriffe, durchgeführt werden kann.

- Bei den Algorithmen XTEA und DES konnten keine relevanten Unterschiede zwischen der Dauer der Verschlüsselung und der Dauer der Entschlüsselung festgestellt werden. Beim Algorithmus PRESENT war nur die Verschlüsselungsroutine definiert.

- Beim Algorithmus AES dauert die Entschlüsselung dagegen länger als die Verschlüsselung. Dies ist mit der komplexeren inversen Operation zu erklären.

- Bei dem asymmetrischen Algorithmus RSA war die Dauer der Verschlüsselung etwa um den Faktor 78 geringer als die Dauer der Entschlüsselung. Dieser Unterschied ist vor allem auf die Wahl der Parameter e bei der Verschlüsselung zurückzuführen.

- Beim Datendurchsatz liefert AES auf dem 8-Bit Mikrocontroller das mit Abstand beste Resultat. Auf dem 32-Bit Mikrocontroller hat XTEA die besten Werte gezeigt. Auf beiden Mikrocontrollern lieferte RSA jeweils die schlechtesten Werte.[194]

- Bei der Untersuchung in Bezug auf die datenabhängigen Berechnungen konnte mit den gegebenen Messverfahren keine

194. Vgl. die Abbildung 3.19.

> signifikante Abweichung bzw. keine Tendenz bei steigenden Hamming-Gewichten festgestellt werden. Bei allen symmetrischen Algorithmen lieferte die Untersuchung konstante Ergebnisse in Bezug auf die Dauer der Verschlüsselung. Beim Algorithmus RSA konnte aufgrund der starken Abweichungen der jeweiligen Werte keine verifizierbare Messung durchgeführt werden.

Aufgrund der sehr langen Dauer der Verschlüsselung und vor allem der Entschlüsslung bei RSA ist der Einsatz dieser Art von Algorithmen auf den untersuchten Plattformen nicht zu empfehlen. Falls im konkreten Anwendungsfall die Eigenschaften der asymmetrische Kryptografie gewünscht sind, muss auf alternative Konzepte zurückgegriffen werden. Denkbar sind Ansätze, die auf der Kryptografie mit elliptischen Kurven basieren.[195]

195. Vgl. den Abschnitt 5.2.

Die Abbildung 3.19 liefert einen Vergleich der untersuchten Algorithmen in Bezug auf den Datendurchsatz und die Codegröße. Auf der Abszissenachse ist der Datendurchsatz in KBit pro Sekunde auf dem 32-Bit Mikrocontroller abgebildet. Auf der Ordinatenachse ist der Datendurchsatz in KBit pro Sekunde auf dem 8-Bit Mikrocontroller angegeben. Im Koordinatensystem sind die untersuchten Algorithmen abgebildet, wobei die jeweilige Codegröße der Algorithmen durch die Größe der Kreise dargestellt ist. Die Flächenmäßig größeren Kreise repräsentieren Algorithmen, die verhältnismäßig mehr Platz im Speicher benötigen. Die Algorithmen, die durch kleinere Kreise repräsentiert sind, benötigen dagegen weniger Platz im Speicher des Mikrocontrollers. Die beste Bewertung erreichen Algorithmen, die sich in der oberen rechten Ecke des Koordinatensystems befinden. Die schlechteste Bewertung bekommen dagegen die Algorithmen, die sich in der unteren linken Ecke des Koordinatensystems befinden.

Der Algorithmus RSA hat sowohl auf dem 8-Bit als auch auf dem 32-Bit Mikrocontroller schlechte Resultate in Bezug auf den Datendurchsatz gezeigt und belegt demnach die Position unten links. Die Algorithmen 3DES und PRESENT zeigen auf beiden Plattformen bessere Resultate als RSA, wobei 3DES bessere Resultate auf dem 32-Bit Mikrocontroller als PRESENT aufweist. XTEA zeigt unter allen untersuchten Algorithmen die besten Resultate auf dem 32-Bit Mikrocontroller, hat aber einen geringen Datendurchsatz auf dem 8-Bit Mikrocontroller.[196]

196. Der Datendurchsatz liegt unter dem von 3DES und PRESENT; vgl. Abbildung 3.13.

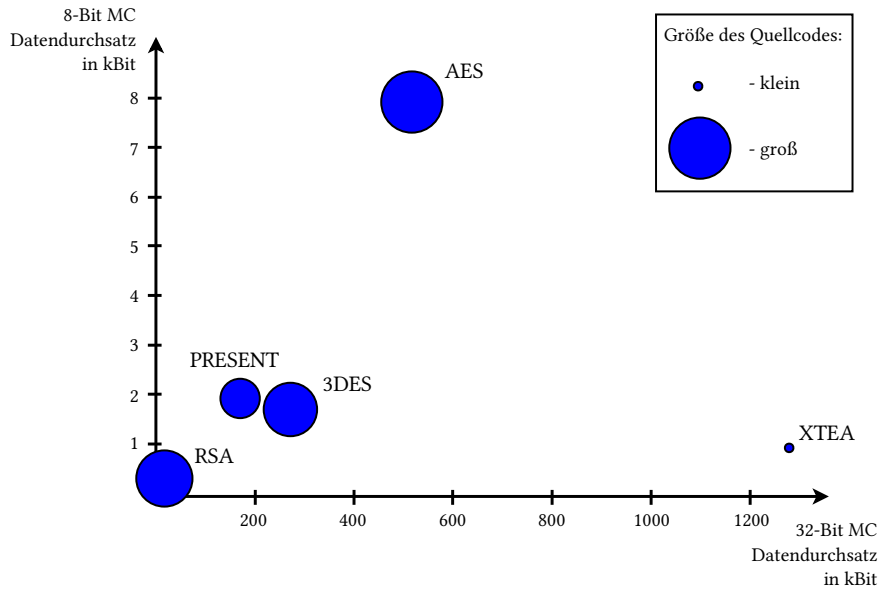


Abbildung 3.19: Vergleich der untersuchten Algorithmen in Bezug auf den Datendurchsatz und die Größe des Quellcodes.

AES hat mit Abstand den höchsten Datendurchsatz auf dem 8-Bit Mikrocontroller und einen mittleren Datendurchsatz unter den untersuchten Algorithmen auf dem 32-Bit Mikrocontroller.[197]

Ausgehend von der durchgeführten Analyse und unter Betrachtung des Kriteriums Datendurchsatz ist der Einsatz des Algorithmus AES zu empfehlen. Dies gilt insbesondere dann, wenn beide Mikrocontroller in der Anwendung in etwa gleich häufig benutzt werden.[198] AES hat gegenüber anderen Algorithmen den Vorteil, sehr verbreitet zu sein und von zahlreichen Bibliotheken unterstützt zu werden. Wird jedoch nur der 32-Bit Mikrocontroller in einer konkreten Anwendung verwendet, sollte die Benutzung vom Algorithmus XTEA in Betracht gezogen werden.

197. Vergleichbare Ergebnisse liefert eine Untersuchung von Rinne u. a. [39].

198. Wird nur der 8-Bit Mikrocontroller benutzt, sollte ebenfalls der Algorithmus AES ausgewählt werden.

UMSETZUNG DER SICHERHEITSANFORDERUNGEN

4

» ... ›textbook crypto‹ is only good in an ideal world where data are random and bad guys behave nicely.«

Wenbo Mao [29, S. ix].

Ziel dieses Kapitels ist es, eine mögliche Umsetzung der Sicherheitsanforderungen auf den untersuchten Plattformen zu zeigen. Dabei wird auf die Ergebnisse der empirischen Untersuchung aus dem 3. Kapitel zurückgegriffen. Nach einer Beschreibung der gestellten Sicherheitsanforderungen findet eine Darstellung einer möglichen Implementierung statt. Nacheinander wird die Implementierung einer Verschlüsselung, die Prüfung der Integrität sowie die Durchführung der Authentifizierung beschrieben. Zum Schluss wird der Aufbau der zur Arbeit beiliegenden CD erläutert.

4.1 BESCHREIBUNG DER SICHERHEITSANFORDERUNGEN

Ausgehend von der empirischen Untersuchung im 3. Kapitel wurde ermittelt, dass AES die besten Ergebnisse in Bezug auf den Datendurchsatz liefert. Dabei wird stets angenommen, dass beide der untersuchten Mikrocontroller bei der Anwendung verwendet werden. Die zentralen Sicherheitsanforderungen an eine Anwendung sind häufig: [199]

199. Vgl. Abschnitt 2.1.1.

1. Sicherstellung der Vertraulichkeit der Informationen;
2. Sicherstellung der Integrität der Informationen;

3. eindeutige Authentifizierung von Benutzern.

Ein kryptografisches Protokoll ist eine konkrete Realisierung einer Sicherheitsanforderung, die auf einer Konstruktion von kryptografischen Primitiven basiert. Ein kryptografisches Protokoll muss zudem so aufgebaut sein, dass es für den Angreifer nicht möglich sein sollte zusätzliche Informationen aus der Kommunikation zu erfahren als die, die im Protokoll definiert sind.[200] Bei den folgenden Umsetzungen der Sicherheitsanforderungen wird AES als das zugrunde liegende Algorithmus verwendet. Die vorgestellte Implementierung basiert daher ausschließlich auf den Verfahren der symmetrischen Kryptografie.

200. Vgl. Schneier [44, S. 26].

4.2 Implementierung der Algorithmen

Bei der folgenden Implementierung der Sicherheitsanforderungen wird auf eine bestehende Implementierung der kryptografischen Primitive zurückgegriffen. In der vorgestellten Umsetzung der Sicherheitsanforderungen wird die bereits in der Untersuchung verwendete AES-Implementierung von Gladman als Grundlage benutzt.[201]

201. Vgl. Gladman [15].

4.2.1 Verschlüsselung

Die Verschlüsselung wird im Cipher-Block-Chaining Modus (CBC) mit einer Schlüsselgröße von 128 Bit durchgeführt. Der CBC-Modus wurde aus folgenden Gründen für die Implementierung ausgewählt:

- einfache Struktur und Sicherstellung der nicht vorhandenen Unterscheidbarkeit von Geheimtexten;[202]
- Möglichkeit zur Durchführung einer Integritätsprüfung;
- eine bereits vorhandene Realisierung von CBC-Modus in der Version von Gladman [15].

202. Der Angreifer kann nicht unterscheiden, von welchem Klartext der Kryptotext stammt.

Im CBC-Modus werden die Daten in Blöcken von der Größe 128 Bit aufgeteilt. Belegen die Daten den letzten Block nicht komplett, werden die restlichen Bits im Block mit der Folge »1, 0, 0...0« aufgefüllt. Durch ein derartiges Auffüllen ist es möglich zu erkennen, wann der letzte

Datenblock endet. Dazu wird der letzte Block von hinten durchgegangen und es wird nach der ersten »1« gesucht. Wird diese gefunden, sind alle Bits vor dieser »1« die eigentlichen Daten.[203] Zudem muss beim Auffüllen von Nachrichten stets beachtet werden, dass bei den Nachrichten, die genau n-Blöcke lang sind, ein Hinzufügen eines neuen Blocks, der vollständig aus »1, 0, 0...0« besteht, erforderlich ist.

203. Diese Art des Auffüllens von Blöcken ist im ISO-Standard [20] spezifiziert.

Anschließend erfolgt eine Verschlüsselung im CBC-Modus. Der Initialisierungsvektor wird pro Verschlüsselungsprozess zufällig generiert und zusammen mit den verschlüsselten Daten übermittelt und gespeichert. Der Initialisierungsvektor selbst muss dabei nicht geheim übertragen werden. In der Abbildung 4.1 wird der Prozess der Verschlüsselung im CBC-Modus erläutert.[204]

204. Mehr zum CBC-Modus im Abschnitt 2.2.3. Die Erläuterung der jeweiligen Symbole der Abbildungen ist im Anhang 1 auf der Seite 105 zu finden.

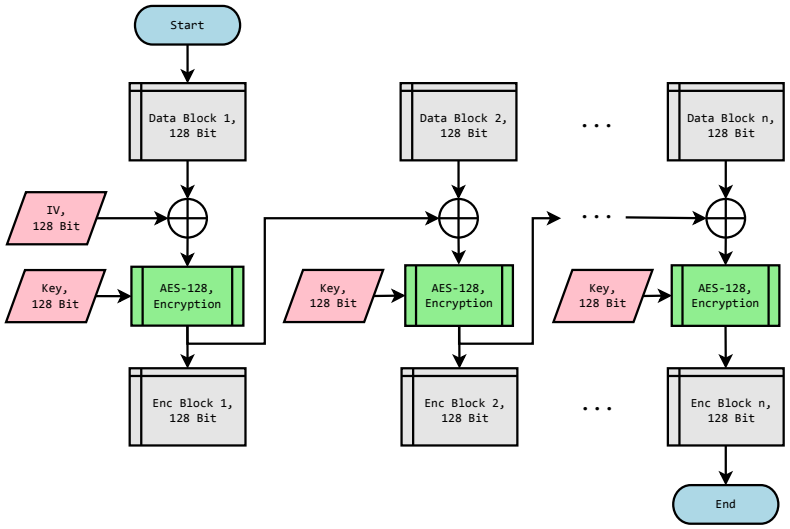


Abbildung 4.1: Verschlüsselung mit AES im CBC-Modus.

Bei der Entschlüsselung im CBC-Modus wird nach der Anwendung der Entschlüsselungsfunktion der erste Block mit dem Initialisierungsvektor mittels XOR verknüpft. In allen folgenden Blöcken wird mit dem vorhergehenden Kryptotextblock mittels XOR verknüpft.[205] Der Initialisierungsvektor kann anschließend verworfen werden.

205. Vgl. Abbildung 4.2.

Ist die Datenmenge gering und entspricht sie nur einigen wenigen Blöcken, kann die zusätzliche Speicherung des 128-Bit großen Initialisierungsvektors ein Problem darstellen. Sind die Sicherheitsanforderungen nicht besonders hoch, kann der Initialisierungsvektor als ein fortlaufender Zähler implementiert werden. In diesem Fall muss der IV selbst nicht zusammen mit der Nachricht übermittelt bzw. abgespeichert werden. Alternativ sind auch Konstruktionen mit einem

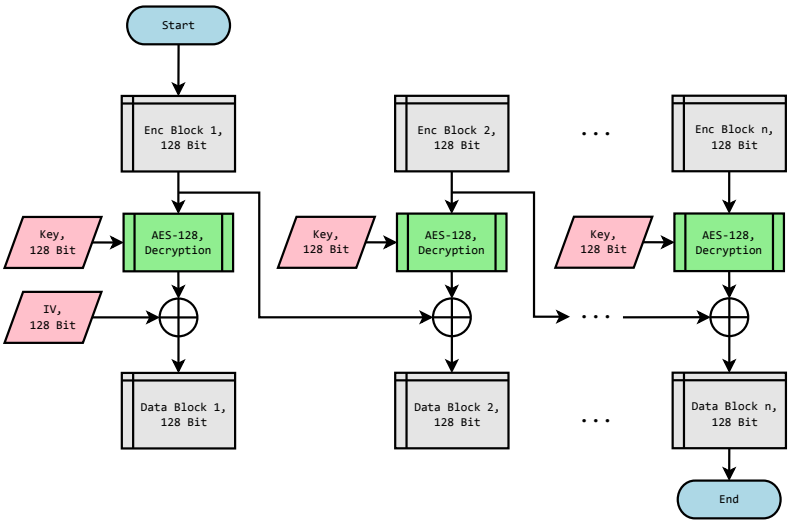


Abbildung 4.2: Entschlüsselung mit AES im CBC-Modus.

festen Initialisierungsvektor vorstellbar. Je nach konkreter Sicherheitsanforderung ist die geeignete Variante auszuwählen.

Der Algorithmus AES existiert in unterschiedlichen Implementierungen und Varianten der Ausführung. Die Geschwindigkeit der Ausführung von AES kann gesteigert werden, wenn bestimmte Operationen im Voraus berechnet werden. So kann beispielsweise die Expansion des Schlüssels in die Rundenschlüssel im Voraus erfolgen. Dies beschleunigt somit den eigentlichen Prozess der Verschlüsselung. Dieser Effekt ist besonders dann bemerkbar, wenn häufig mit demselben Schlüssel gearbeitet wird.

4.2.2 *Prüfung der Integrität*

Die Prüfung der Integrität ist eine wesentliche kryptografische Anforderung, die bewirkt, dass eine nicht legitime Manipulation der Daten aufgedeckt werden kann. Dafür wird an den Kryptotext eine *kryptografische Prüfsumme*[206] angehangen, die nur durch die Kenntnis eines geheimen Schlüssels und des Klartextes generiert werden kann. Die Verwendung eines geheimen Schlüssels bei der Erzeugung von kryptografischen Prüfsummen ist notwendig, da sonst der Angreifer bei der Manipulation der Daten gleichzeitig auch die Prüfsumme erneut generieren und abspeichern könnte. Die durch die Verwendung von einem geheimen Schlüssel erzeugten Prüfsummen werden als Message Authentication Code (MAC) bezeichnet.[207] Eine Möglichkeit einen

206. Auch kryptografische Hashfunktion genannt.

207. Mehr zu den MAC s. Paar u. Pelzl [37, S. 319 ff.].

MAC zu erzeugen ist die Verwendung einer Blockverschlüsselung im CBC-Modus. Der Vorteil der Verwendung von CBC-MAC im aktuellen Fall ist, dass für die Integritätsprüfung die gleiche Funktion, wie schon für die Verschlüsselung der Daten, benutzt werden kann. Dadurch kann die gesamte Quellcodegröße und die Größe des Kompilats verringert werden. Die Abbildung 4.3 stellt den Aufbau einer Funktion für die Erzeugung einer MAC im CBC-Modus dar.

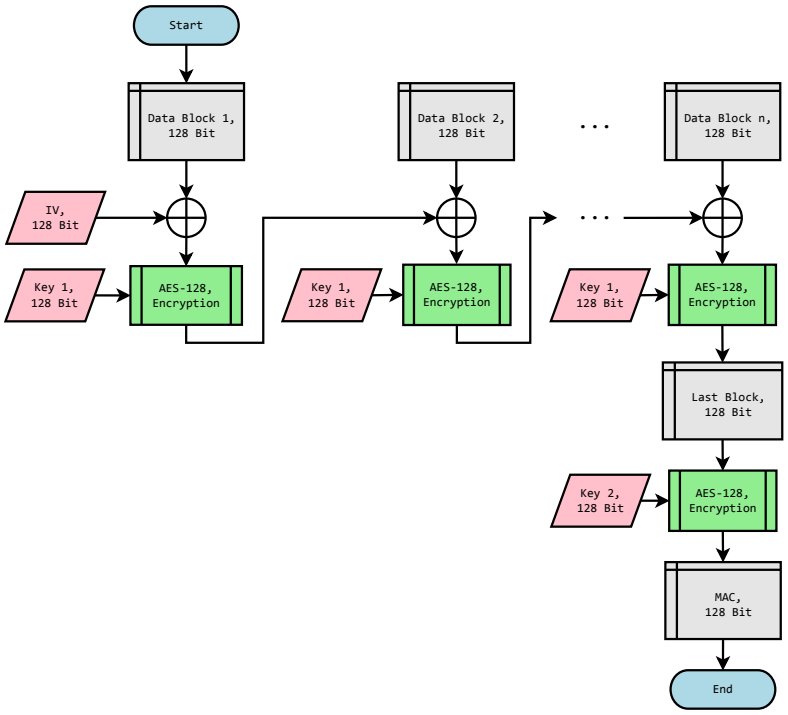


Abbildung 4.3: Erzeugung einer MAC mit Hilfe von AES im CBC-Modus.

Bei der Prüfung der Integrität wird die bereits vorgestellte CBC-Konstruktion angewendet. Jedoch wird nur der letzte Kryptotextblock als Ausgabe verwendet. Alle vorherigen Blöcke werden nicht benötigt und verworfen. Da der CBC-Modus eine Verkettung aller Blöcke darstellt, bewirkt eine Veränderung in jedem beliebigen Block eine Veränderung im letzten Kryptotextblock. Das ist genau die von einer Prüfsumme geforderte Eigenschaft.

Für die Erzeugung einer MAC im CBC-Modus muss unbedingt ein zu der eigentlichen Verschlüsselungsfunktion separater Schlüssel verwendet werden. Ansonsten kann ein Angreifer (Oscar) den gesamten Kryptotext bis auf den letzten Block verändern und der MAC würde dabei weiterhin gültig sein. Dieser Angriff beruht auf der Überlegung, dass bei einer CBC-MAC die eigentliche MAC gleich dem

letzten Kryptotextblock ist, sofern bei beiden Operationen derselbe Schlüssel verwendet wurde:[208]

208. Vgl. die Abbildungen 4.1 und 4.3 vor der letzten Verschlüsselungsoperation.

$$\mathrm{MAC} = e_k\left(p_n \oplus c_{n-1}\right) = c_n.$$

Daher ist es für den Angreifer weiterhin einfach, eine Nachricht unentdeckt zu modifizieren, sofern im CBC-MAC der gleiche Schlüssel verwendet wird wie für die Verschlüsselung.

Um sich einem weiteren gefährlichen Angriff zu widersetzen ist zudem eine erneute Verschlüsselung des letzten Kryptotextblocks im CBC-Modus erforderlich.[209] Der Grund für diesen letzten Schritt ist folgender möglicher Angriff: Gegeben seien m – eine Nachricht von der Blocklänge des zugrundeliegenden Algorithmus und t – der dazugehörige, mit CBC-MAC generierter, MAC. Dieser MAC wird jedoch ohne die letzte Verschlüsselungsfunktion erzeugt. Dann existiert jedoch eine Nachricht $m' \neq m$ für die t ebenfalls ein gültiger MAC ist. Diese Art von Angriffen werden *cut-and-paste Angriffe* genannt und funktionieren dabei wie folgt:[210]

209. Vgl. die letzte Operation in der Abbildung 4.3.

210. Vgl. Knudsen u. Robshaw [24, S. 79].

1. der Angreifer wählt eine Nachricht m und fordert den MAC für diese Nachricht an;

2. der Angreifer bekommt ein Paar (m, t) zurück, wobei t der zu m dazugehörige MAC ist;

3. nun ist t aber ebenfalls ein gültiger MAC für die zusammengesetzte Nachricht $m' = (m \,\|\, t \oplus m)$.

Denn wenn der letzte Verschlüsselungsschritt im CBC-Modus nicht durchgeführt wird, gilt:

$$\begin{aligned}\mathrm{MAC} = e_k\left(m \,\|\, t \oplus m\right) &= e_k\left(e_k(m) \oplus (t \oplus m)\right) \\ &= e_k\left(t \oplus (t \oplus m)\right) = e_k(m) = t.\end{aligned}$$

Somit kann ein Angreifer eine beliebige Nachricht m wählen und anschließend eine sogenannte *Kollision* der MACs erzeugen. Der Schlüssel, der in der letzten Verschlüsselungsfunktion verwendet wird, muss folglich ein anderer sein als der Schlüssel, der in der eigentlichen CBC-Funktion verwendet wurde.

Die Abbildung 4.4 stellt die Funktion für die Überprüfung eines gegebenen MAC dar. Im ersten Schritt wird für die gegebenen Daten ein MAC erzeugt.[211] Der daraus resultierende MAC wird nun mit dem vorher übermittelten bzw. gespeicherten MAC verglichen. Stimmen beide MACs überein, kann davon ausgegangen werden, dass die Daten nicht modifiziert wurden. Sind die zwei MACs unterschiedlich, wird eine entsprechende Fehlermeldung ausgegeben.

211. Es wird das gleiche Vorgehen wie in Abbildung 4.3 angewendet.

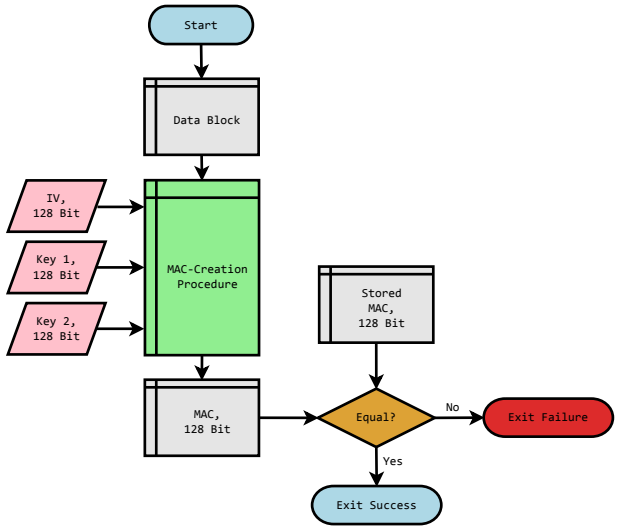


Abbildung 4.4: Verifikation eines MAC, der mit AES im CBC-Modus erzeugt wurde.

In der vorliegenden Umsetzung wird für die Verschlüsselung der Daten AES im CBC-Modus angewendet. Daher kann die Verschlüsselung/Entschlüsselung von Daten mit den gleichen Funktionen erfolgen wie die Erzeugung/Verifikation des MAC. Der folgende Quellcode beschreibt die Funktion für die Entschlüsselung und für die Verifikation eines MAC mit AES im CBC-Modus in der Programmiersprache C. Für die Verifikation müssen folgende Daten übergeben werden:

- in: die eigentlichen Daten;
- mac: der zu den Daten dazugehörige MAC;
- iv: der für die Verschlüsselung der Daten verwendete MAC;
- blocks: die Länge der Daten gemessen in Blockgröße von AES;[212]
- key: Schlüssel für die Verschlüsselung und Erzeugung der MAC;
- out: die entschlüsselten Daten.

212. Bei AES liegt die Blockgröße bei 128 Bit.

Die Ver- bzw. Entschlüsselung mit AES funktioniert in der gewählten Implementierung in zwei Schritten. Zuerst wird der Schlüssel expandiert und die Rundenschlüssel werden erzeugt.[213] Anschließend erfolgt die eigentliche Ver- bzw. Entschlüsselungsoperation. Die Entschlüsselung der Daten und Verifikation des MAC erfolgt in drei Schritten:

213. Vgl. Quellcode 4.1 die Zeilen 8, 12 und 16.

1. Entschlüsseln der gegebenen Daten mit dem ersten Schlüssel (Zeilen 8 und 9);
2. Berechnung des MAC durch das Verschlüsseln der Daten mit dem zweiten Schlüssel (Zeilen 12 und 13);
3. Verschlüsselung des letzten Blocks mit dem dritten Schlüssel (Zeilen 16 und 17).

```
1  int encrypt_and_verify(const unsigned char *in,
2                const unsigned char mac,
3                const unsigned char iv,
4                const int blocks,
5                const unsigned char key[48],
6                unsigned char *out {
7    //Expand AES Key1 and decrypt the message
8    aes_set_key(&key[0], 16, &aes-key);
9    aes_cbc_decrypt(in, out, blocks, iv, aes-key);
10
11   //Expand AES Key2 and create MAC
12   aes_set_key(&key[16], 16, &aes-key);
13   aes_cbc_encrypt(out, pre-mac, blocks, iv, aes-key);
14
15   //Expand AES Key3 and do last incryption step
16   aes_set_key(&key[32], 16, &aes-key);
17   aes_encrypt(&pre-mac[offset], out-mac, aes-key);
18
19   //Verify MAC and return 0 in case of success
20   return memcmp(mac, out-mac, 16);
21 }
```

Quellcode 4.1: Verifikation eines MAC mit AES im CBC-Modus.

Zum Schluss wird der generierte MAC mit dem übergebenen MAC verglichen (Zeile 20) und das Ergebnis des Vergleichs als Rückgabewert der Funktion zurückgegeben. Sind die zwei MACs übereinstimmend,

wird eine »0« zurückgegeben, anderenfalls wird eine Zahl größer bzw. kleiner null zurückgegeben.[214]

214. Die Funktion `memcmp` vergleicht zwei übergebene Werte und gibt eine Zahl $\neq 0$, falls die zwei Werte ungleich sind.

Die Länge eines MAC ist für die Integrität der Daten entscheidend. Längere MACs haben generell eine höhere Sicherheit. Bei der Verwendung von AES als die zugrunde liegende Funktion ist der MAC 128 Bit lang. Die Größe ist nach den heutigen Standards ausreichend. Je nach gegebenen Sicherheitsanforderungen kann die MAC jedoch auch verkürzt werden.[215] Es ist dabei zu beachten, dass zu kurz gewählte MACs, beispielsweise kleiner als 48 Bit, nicht mehr als sicher angesehen werden und erfolgreich über Brute-Force-Attacken angegriffen werden können.

215. Vgl. Knudsen u. Robshaw [24, S. 83 f.].

4.2.3 *Authentifizierung*

Eine Authentifizierung dient dem Zweck, autorisierte Benutzer von nicht autorisierten zu unterschieden. Der Nachweis der Berechtigung zum Zugriff kann auf folgenden Arten erfolgen:[216]

216. Vgl. Stinson [49, S. 353 ff.].

- durch den Nachweis vom geforderten Wissen;
- durch den Nachweis vom geforderten Besitz;
- durch eine biometrische Identifikation.

Vor dem eigentlichen Prozess der Authentifizierung muss über einen sicheren Kanal zuerst ein Austausch des für die Authentifizierung benötigten geheimen Schlüssels erfolgen. Eine zusätzliche Anforderung bei der Authentifizierung besteht darin, dass die Authentifizierungsdaten nicht im Klartext im System gespeichert werden dürfen. Bei einer Kompromittierung des Systems würden ansonsten alle gespeicherten Daten dem Angreifer zur Verfügung stehen. Gewöhnlich wird daher eine Hashfunktion auf die Daten angewendet und die resultierende Prüfsumme im Speicher abgelegt. Findet nun eine Authentifizierung statt, wird auf die übermittelten Daten wieder die Hashfunktion angewendet und der resultierende Wert mit der gespeicherten Prüfsumme verglichen.

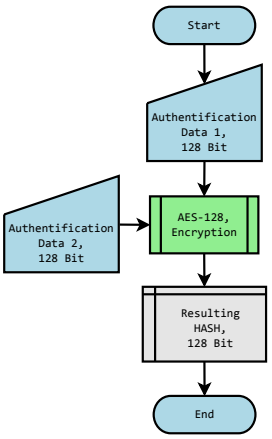


Abbildung 4.5: Erzeugung einer Prüfsumme auf der Grundlage von AES.

In der aktuellen Implementierung kann AES als die zugrunde liegende Funktion verwendet werden und somit ein Ersatz für die Hashfunktion sein. In der Abbildung 4.5 wird die Durchführung der Authentifizierung auf der Grundlage von AES beschrieben. Die vom

Benutzer zur Autorisierung übergebenen Daten werden mit AES verschlüsselt. Die resultierende Prüfsumme wird im Speicher abgelegt. Zur Erhöhung der Sicherheit können zu den bereitgestellten Benutzerdaten ergänzend Zufallszahlen hinzugefügt werden.[217] Die zusätzlichen Zufallszahlen verhindern oder erschweren einige Angriffe, wenn die Benutzerdaten zu leicht zu erraten sind bzw. eine zu kurze Länge besitzen. Die Hashfunktion wird anschließend über diese zusammengesetzten Daten ausgeführt.[218]

217. Diese Daten werden häufig als *salt* bezeichnet.

218. Da die Speicherung eines Schlüssels für die Erzeugung der Hashzahlen nicht wünschenswert ist, kann die Hälfte der Daten für die Authentifizierung als Schlüssel verwendet werden.

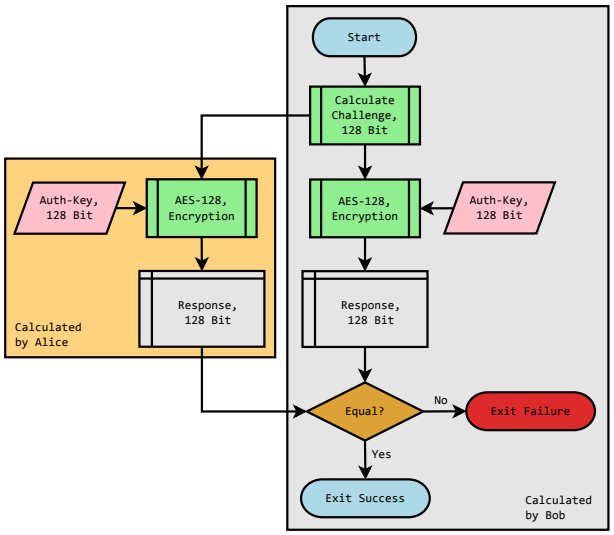


Abbildung 4.6: Durchführung einer Authentifizierung mit Challenge-Response-Verfahren auf der Grundlage von AES.

Eine weitere Verbesserung der Sicherheit wird durch das sogenannte *Challenge-Response-Verfahren*[219] erreicht. Das Verfahren auf der Grundlage von AES wird in der Abbildung 4.6 beschrieben. Angenommen Alice will sich gegenüber Bob authentifizieren. Dabei wird für jeden Authentifizierungsvorgang eine neue Zufallszahl (*Challenge*) von Bob generiert und an Alice gesendet. Daraufhin wendet Alice AES mit dem geheimen Schlüssel[220] auf die Challenge an und sendet die Antwort an Bob (*Response*). Bob führt währenddessen die gleiche Operation durch und vergleicht das erhaltene Response mit der eigenen Berechnung. Stimmen beide Datensätze überein, wird der Zugriff auf die Ressource gewährt. Anderenfalls wird der Zugriff verweigert. Weitere Informationen zu Challenge-Response-Verfahren sind bei Stinson [49, S. 356 ff.] zu finden.

219. Deutsch: Aufforderung und Antwort.

220. Der geheime Schlüssel muss vorher ausgetauscht sein.

Die Verschlüsselung, die Prüfung der Datenintegrität und die Authentifizierung sind demnach mit nur einem zugrunde liegenden Algo-

rithmus durchzuführen. Diese Herangehensweise erlaubt es Platz im Flash-Speicher der Mikrocontroller zu sparen und vereinfacht zudem die Wartung des Quellcodes. Die durchgeführte empirische Untersuchung hat ergeben, dass bei einer Verwendung von einem Controller mit einer Busbreite von 8-Bit, AES den höchsten Datendurchsatz ermöglicht und somit für den Einsatz zu empfehlen ist. Grundsätzlich kann jedoch die vorgestellte Umsetzung der Sicherheitsanforderungen auch auf anderen zugrundeliegenden Algorithmen aufbauen.

4.3 Aufbau der beigelegten CD

Der vorliegenden Arbeit ist ein Compact Disc (CD) beigelegt auf dem sich folgende Daten befinden:

- Quellcode von den folgenden untersuchten Algorithmen: DES, AES, XTEA, PRESENT, RSA;
- Diplomarbeit im PDF Format;
- Programm für die empirische Untersuchung von RSA am 8-Bit Mikrocontroller;
- Programm für die empirische Untersuchung von RSA am 32-Bit Mikrocontroller;
- Programm für die empirische Untersuchung von symmetrischen Algorithmen am 8-Bit Mikrocontroller;
- Programm für die empirische Untersuchung von symmetrischen Algorithmen am 32-Bit Mikrocontroller.

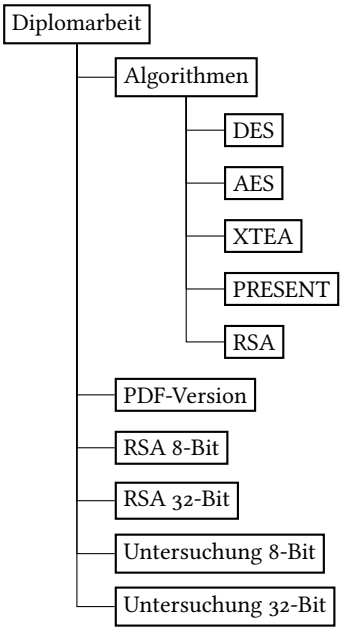


Abbildung 4.7: Aufbau der zu Arbeit beigelegten CD.

Die Abbildung 4.7 verdeutlicht die Zusammensetzung der Daten auf der CD. Für eine korrekte Ausführung der Programme für die empirische Untersuchung werden spezielle, vom Hersteller der Mikrocontroller bereitgestellte, Entwicklungsumgebungen benötigt. Weitere Informationen und Anleitungen zum Herunterladen der Softwarepakete sind unter [11] und [12] zu finden.

SCHLUSSBEMERKUNGEN

5

»Only amateurs attack machines; professionals target people.«

Bruce Schneier [43].

In diesem Kapitel wird eine Zusammenfassung der wichtigsten Ergebnisse der Arbeit sowie ein Ausblick über die weiterführenden Forschungsthemen gegeben. Die zentralen Ergebnisse der Arbeit sind durch die durchgeführte empirische Untersuchung auf den vorgestellten Mikrocontrollern belegt. Bei den möglichen Themen für eine weiterführende Forschung ist insbesondere die Untersuchung von asymmetrischen Verfahren auf der Basis von elliptischen Kurven für den Einsatz auf kleineren Mikrocontrollern hervorzuheben.

5.1 ZUSAMMENFASSUNG DER ARBEIT

Die vorliegende Arbeit hatte das Ziel, bekannte Verschlüsselungsalgorithmen beim Einsatz auf Mikrocontrollern unter begrenzten Ressourcen zu analysieren und dabei durch eine empirische Untersuchung eine Bewertung der Algorithmen vorzunehmen. Im Folgenden werden die zentralen Ergebnisse der jeweiligen Kapitel der Arbeit vorgestellt.

Nach der Einleitung wurde im zweiten Kapitel die Einführung in die zentralen Begriffe und Konstrukte der Kryptologie gegeben. Ein geschichtlicher Abriss war sinnvoll, um die zeitliche Entwicklung der Techniken bei der Konstruktion der Algorithmen hin zu den modernen digitalen kryptografischen Verfahren verstehen und bewerten zu können. Zudem wurde die Trennung der modernen Kryptografie in symmetrische und asymmetrische Kryptografie verdeutlicht. Mit der Darstellung der Kerckhoffs Prinzipien wurden die Richtlinien vorgestellt, die für die moderne Entwicklung von kryptografischen Algorithmen prägend sind. Eine Beschreibung vom Aufbau

des Substitutions-Permutations-Netzwerks und des Feistel-Netzwerks lieferte mathematische Definition und das Verständnis für die spätere genaue Analyse der jeweiligen Algorithmen. Daneben wurden mit der Darstellung der drei unterschiedlichen Klassen der Angriffe – nämlich den Angriffen auf die Implementierung, den social engineering Angriffen und den Angriffen auf die eigentliche Konstruktion der Algorithmen – wichtige Möglichkeiten der Kryptoanalyse gezeigt.

Im dritten Kapitel wurden die zwei bei der Untersuchung verwendeten Mikrocontroller beschrieben. Vor der eigentlichen empirischen Untersuchung wurde der Aufbau der ausgewählten Algorithmen im Detail vorgestellt. Dabei wurde die Konstruktion der Algorithmen DES, AES, XTEA, PRESENT und RSA unter formalen Kriterien miteinander verglichen. In der empirischen Untersuchung wurden diese Algorithmen in Bezug auf die Größe des Quellcodes, die Größe des kompilierten Codes, die Dauer der Ver- und Entschlüsselung sowie auf den möglichen Datendurchsatz analysiert. Dabei wurde stets auf den eingesetzten 8-Bit und 32-Bit Mikrocontrollern getrennt getestet. Die Untersuchung auf zwei verschiedenen Mikrocontrollern ermöglichte die Unterschiede der beiden Plattformen beim Einsatz der Algorithmen zu beleuchten. Es konnte gezeigt werden, dass die unterschiedlichen Algorithmen eine große Bandbreite beim benötigten Speicherplatz und dem möglichen Datendurchsatz aufweisen. Beim Algorithmus XTEA ist der Datendurchsatz auf dem 8-Bit Mikrocontroller sehr gering, dagegen ist er auf dem 32-Bit Mikrocontroller überdurchschnittlich hoch. Auf der anderen Seite ist der kompilierte Code von XTEA für den 8-Bit Mikrocontroller etwa drei Mal so umfangreich wie für den 32-Bit Mikrocontroller. Durch die durchgeführte Untersuchung konnte empirisch ermittelt werden, dass der Algorithmus AES auf dem 8-Bit Mikrocontroller und der Algorithmus XTEA auf dem 32-Bit Mikrocontroller die mit Abstand besten Resultate in Bezug auf den möglichen Datendurchsatz pro Sekunde liefern. Weitere interessante Ergebnisse lieferte die Untersuchung der Dauer der Verschlüsselung und der Entschlüsselung der jeweiligen Algorithmen. Die Dauer der Verschlüsselung unterscheidet sich kaum von der Dauer der Entschlüsselung bei den Algorithmen XTEA und DES. Bei Algorithmus AES gibt es dagegen eine Abweichung nach oben bei der Dauer der Entschlüsselung im Vergleich zu der Dauer der Verschlüsselung. Die Untersuchung vom Algorithmus RSA auf den Mikrocontrollern hat ergeben, dass

der Einsatz von RSA auf den gegebenen Plattformen nicht effizient möglich ist. Die Auswertung sowohl auf dem 8-Bit Mikrocontroller als auch auf dem 32-Bit Mikrocontroller lieferte sehr geringe Werte für den möglichen Datendurchsatz.

Im vierten Kapitel wurden die Sicherheitsanforderungen wie Verschlüsselung, Prüfung der Integrität und Authentifizierung unter Nutzung eines kryptografischen Algorithmus umgesetzt. Ausgehend von den Ergebnissen der empirischen Untersuchung wurde AES als der zugrunde liegende Algorithmus für die Umsetzung ausgewählt. Alle drei vorgestellten kryptografischen Anforderungen sind mit diesem Algorithmus zu realisieren und ermöglichen somit die gesamte Größe des Quellcodes und des kompilierten Codes möglichst gering zu halten. Für die Verschlüsselung wurde AES im CBC-Modus umgesetzt. Die Prüfung der Integrität erfolgte über eine Konstruktion mit CBC-MAC. Die Authentifizierung wurde mit Hilfe von Challenge-Response-Verfahren ebenfalls auf der Grundlage von AES umgesetzt. Somit lassen sich die drei kryptografischen Anforderungen ausschließlich mit den Mitteln der symmetrischen Kryptografie realisieren.

5.2 AUSBLICK

Im Rahmen der vorliegenden Arbeit konnte eine Reihe von Fragen in Bezug auf den Einsatz von kryptografischen Methoden auf Mikrocontrollern untersucht werden. Es haben sich zudem weitere Fragen eröffnet, die im Rahmen von zusätzlichen Untersuchungen geklärt werden können. Im Folgenden werden die ausgewählten Ansatzpunkte skizziert, die bei einer weitergehenden Forschung Verwendung finden können:

- Die Anwendung von kryptografischen Methoden, die auf dem Einsatz von elliptischen Kurven (ECC)[221] basieren, ist ein interessantes Feld für weitergehende Analysen und praktische Untersuchungen. Insbesondere für den Einsatz auf Mikrocontrollern unter begrenzten Ressourcen ist ECC eine sehr aussichtsreiche Variante die gewünschten Sicherheitsanforderungen zu erfühlen. Einige Resultate in dieser Hinsicht sind bei Kumar u. Paar [26], Gura u. a. [17] und Blaß u. Zitterbart [3] zu finden.

221. ECC – Elliptic Curve Cryptography.

- Ein weiteres Feld für zukünftige Untersuchungen bietet die Analyse von Stromalgorithmen. Stromalgorithmen haben den Vorteil, höheren Datendurchfluss als Blockalgorithmen zu ermöglichen, gelten jedoch als weniger gut erforscht.[222] Eine von der Europäischen Union unterstütze Initiative unter dem Namen eSTREAM erarbeitete in den Jahren 2004 bis 2008 neue Stromalgorithmen, die unter anderem auch für den Einsatz in Hardware mit eingeschränkten Ressourcen entwickelt wurden. Weitere Informationen zu den ausgewählten Algorithmen sind auf der Projektseite unter [8] zu finden.

222. Vgl. Paar u. Pelzl [37, S. 49].

- Insbesondere auf Mikrocontrollern sind Seitenkanalangriffe häufig eine reale Bedrohung. Um derartige Angriffe zu erschweren, sollten die eingesetzten Implementierungen von Algorithmen gegen die gängigen Techniken dieser Angriffsvariante analysiert werden. Dazu gehören in erster Linie eine genaue Analyse der Ausführungsdauer einzelner Schritte, eine Analyse des Energieverbrauchs am Mikrocontroller sowie eine Untersuchung der emittierten elektromagnetischen Strahlung.[223] Bei derartigen Untersuchungen sollte stets die Veränderungen von den jeweiligen Messwerten in Abhängigkeit von den benutzten Daten gemessen und diskutiert werden.

223. Für derartige Untersuchungen sind unter Umständen sehr feine Messinstrumente erforderlich.

- Bei einem praktischen Einsatz von AES sollte die gewählte Implementierung genauer analysiert und an die gegebenen Bedingungen angepasst werden. Bei AES existiert eine große Bandbreite von unterschiedlichen Möglichkeiten, wie die eigentliche Verschlüsselung zu erfolgen hat. Dabei kann beispielsweise die Geschwindigkeit der Ausführung zu Lasten der Quellcodegröße erhöht werden. Es existieren Konstruktionen, bei denen die Rundenschlüssel nicht vor der eigentlichen Verschlüsselung, sondern direkt zur Laufzeit des Algorithmus erzeugt werden.[224]

224. Vgl. die »on the fly« Version von AES bei Gladman [15].

Neben der Untersuchung von mathematischen Angriffsmethoden auf die Algorithmen und den Angriffen auf die jeweilige Implementierung sollten auch die social engineering Angriffe bei der Entwicklung von neuen oder der Analyse von bestehenden Systemen untersucht und bewertet werden. In den kommenden Jahren ist laut Schneier [43] ein Anstieg von derartigen Angriffen zu erwarten.

Trotz der langen Geschichte der Kryptologie und den zahlreichen vorhandenen Standards und Richtlinien durchlebt das Gebiet aktuell eine rasante Entwicklung. Bisher unbekannte Anwendungssituationen für Kryptologie, die sich aus der Einführung und Verbreitung von neuen Möglichkeiten der Vernetzung und der Übermittlung von Daten ergeben, werden folgen und damit neue Herausforderungen aufwerfen. Ich möchte daher die Arbeit mit den Worten von Bruce Schneier abschließen: »You can't defend. You can't prevent. The only thing you can do is detect and respond.« [225]

225. Bruce Schneier, zit. nach [51].

LITERATURVERZEICHNIS

[1] Bauer, Friedrich L.: *Entzifferte Geheimnisse: Methoden und Maximen der Kryptologie.* 2. erweiterte Auflage. Berlin: Springer-Verlag, 1997. – ISBN 3–540–62632–8 (Zitiert auf Seiten 10 und 19).

[2] Biham, Eli ; Shamir, Adi: Differential cryptanalysis of DES-like cryptosystems. In: *Journal of Cryptology* 4, Nr. 1 (1991), S. 3 – 72 (Zitiert auf Seite 36).

[3] Blaß, E. ; Zitterbart, M.: Efficient Implementation of Elliptic Curve Cryptography for Wireless Sensor Networks. 2005 (TM-2005-1). – Forschungsbericht (Zitiert auf Seite 87).

[4] Bogdanov, A. ; Knudsen, L. R. ; Leander, G. ; Paar, C. ; Poschmann, A. ; Robshaw, M. J. B. ; Seurin, Y. ; Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. Version: 2007. http://homes.esat.kuleuven.be/~abogdano/papers/present_ches07.pdf, Abruf: 03. Oktober 2012. In: *Proceedings of CHES 2007*. Springer-Verlag, 2007 (Zitiert auf Seiten 45, 46, und 47).

[5] Borisov, Alexander: *Formate zur Speicherung von Schriften in der digitalen Typografie.* 2012. – Humboldt-Universität zu Berlin, Studienarbeit, http://www.aborisov.de/studienarbeit.html, Abruf: 10. Januar 2013 (Zitiert auf Seite b).

[6] Daemen, Joan ; Rijmen, Vincent: *The Design of Rijndael: AES – The Advanced Encryption Standard.* Berlin: Springer, 2002. – ISBN 3–540–42580–2 (Zitiert auf Seiten 41 und 65).

[7] Dinger, Jochen ; Hartenstein, Hannes: *Netzwerk- und IT-Sicherheitsmanagement: eine Einführung.* Karlsruhe: KIT Scientific Publishing, 2008. – ISBN 978–3–866–44209–2 (Zitiert auf Seite 29).

[8] European Network of Excellence for Cryptology: *The eSTREAM Project*. http://www.ecrypt.eu.org/stream/project.html, Abruf: 13. August 2012 (Zitiert auf Seite 88).

[9] Feistel, Horst: Cryptography and Computer Privacy. In: *Scientific American* 228, 5 (1973), 15 – 23. http://www.apprendre-en-ligne.net/crypto/bibliotheque/feistel/index.html, Abruf: 18. November 2012 (Zitiert auf Seite 33).

[10] Feuersänger, Christian: *PGFPlots - A LaTeXPackage to create normal/logarithmic plots in two and three dimensions*. http://pgfplots.sourceforge.net/, Abruf: 25. Oktober 2012 (Zitiert auf Seite 54).

[11] Freescale Semiconductor, Inc.: *CodeWarrior for MCUs (Eclipse IDE)*. http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=CW-MCU10&parentCode=null&nodeId=0152102726E4C7E4CB, Abruf: 25. Oktober 2012 (Zitiert auf Seiten 54 und 83).

[12] Freescale Semiconductor, Inc.: *CodeWarrior for Microcontrollers (Classic IDE)*. http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=CW-MICROCONTROLLERS&parentCode=null&nodeId=0152102726E4C7E4CB, Abruf: 25. Oktober 2012 (Zitiert auf Seiten 54 und 83).

[13] Freescale Semiconductor, Inc.: *MC9S08SH32 MC9S08SH16 Data Sheet*. 04.2008. – HCS08 Microcontrollers, Rev. 2 (Zitiert auf Seite 31).

[14] Freescale Semiconductor, Inc.: *MCF51CN128 ColdFire Microcontroller, Data Sheet: Technical Data*. 05.2009. – Rev. 4 (Zitiert auf Seite 31).

[15] Gladman, Brian: *Byte Oriented AES (Low Resource Version)*. http://gladman.plushost.co.uk/oldsite/AES/index.php, Abruf: 27. September 2012 (Zitiert auf Seiten 54, 74, und 88).

[16] Gong, Zheng ; Hartel, Pieter ; Nikova, Svetla ; Zhu, Bo: *Software Implementation of Block Cipher PRESENT for 8-Bit Platforms*. http://cis.sjtu.edu.cn/index.php/Software_

Implementation_of_Block_Cipher_PRESENT_for_8-Bit_Platforms, Abruf: 07. Oktober 2012 (Zitiert auf Seite 54).

[17] Gura, Nils ; Patel, Arun ; Wander, Arvinderpal ; Eberle, Hans ; Shantz, Sheueling C.: Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In: *CHES2004, Cambridge (Boston)* (2004). http://labs.oracle.com/projects/crypto/CHES_2004.pdf, Abruf: 10. Dezember 2012 (Zitiert auf Seite 87).

[18] Haan, Kristian L.: *Advanced Encryption Standard (AES).* http://www.codeplanet.eu/tutorials/cpp/51-advanced-encryption-standard.html, Abruf: 07. November 2012 (Zitiert auf Seite 61).

[19] Hamburg, Mike: *Accelerating AES with Vector Permute Instructions.* http://shiftleft.org/papers/vector_aes/vector_aes.pdf, Abruf: 07. November 2012 (Zitiert auf Seite 61).

[20] International Organization for Standardization: *Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher, ISO/IEC 9797-1:2011, 2011* (Zitiert auf Seite 75).

[21] Kahn, David: *The Codebreakers: The Story of Secret Writing.* 2. Auflage. New York: Scribner, 1996. – ISBN 0–684–83130–9 (Zitiert auf Seiten 8, 9, 11, und 12).

[22] Kelsey, John ; Schneier, Bruce ; Wagner, David: Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In: *Information and Communications Security-Proceedings of ICICS 1997* Bd. 1334, Springer-Verlag, 1997 (Zitiert auf Seite 43).

[23] Kerckhoffs, Auguste: La cryptographie militaire. In: *Journal des sciences militaires* 9 (1883), 161 – 191. http://www.petitcolas.net/fabien/kerckhoffs/crypto_militaire_1.pdf, Abruf: 18. November 2012 (Zitiert auf Seite 13).

[24] Knudsen, Lars R. ; Robshaw, Matthew J. B.: *The Block Cipher Companion.* Berlin: Springer, 2011. – ISBN 978–3–642–17341–7 (Zitiert auf Seiten 33, 37, 40, 41, 78, und 81).

[25] Knuth, Donald E.: *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. 1. Auflage. Addison-Wesley, 2009. – ISBN 978–0321637130 (Zitiert auf Seite 49).

[26] Kumar, Eep S. ; Paar, Christof: Are standards compliant elliptic curve cryptosystems feasible on RFID. In: *In Proc. of RFIDSec'06*, 2006 (Zitiert auf Seite 87).

[27] Leander, Gregor: *Lightweight Block Cipher Design*. https://www.cosic.esat.kuleuven.be/ecrypt/courses/albena11/slides/gregor_leander_lightweight.pdf, Abruf: 07. November 2012 (Zitiert auf Seite 25).

[28] Levy, Stuart: *DES implementation, fast and portable, April 1988*. http://www.schneier.com/book-applied-source.html, Abruf: 25. Oktober 2012 (Zitiert auf Seite 54).

[29] Mao, Wenbo: *Modern Cryptography: Theory and Practice*. 1. Auflage. New Jersey: Pearson Education, 2004. – ISBN 0–13–066943–1 (Zitiert auf Seiten 22, 36, 39, 48, 50, und 73).

[30] Merkle, R. C. ; Hellman, M. E.: On the Security of Multiple Encryption. In: *Communications of the ACM* Vol. 24, Nr. 7 (1981), S. 465 – 467 (Zitiert auf Seite 37).

[31] National Institute of Standards and Technology: *Federal Information Processing Standards Publication 197: Advanced Encryption Standard (AES), November 26, 2001*. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, Abruf: 26. Mai 2012 (Zitiert auf Seite 41).

[32] National Institute of Standards and Technology: *Federal Information Processing Standards Publication 46-3: Data Encryption Standard (DES), Oktober 25, 1999*. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf, Abruf: 07. Oktober 2012 (Zitiert auf Seite 34).

[33] National Institute of Standards and Technology: *NIST Special Publication 800-57: Recommendation for Key Management – Part 1: General (Revised), March, 2007*. http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-

Part1-revised2_Mar08-2007.pdf, Abruf: 13. Oktober 2012 (Zitiert auf Seite 52).

[34] National Institute of Standards and Technology: *SKIPJACK and KEA Algorithm Specifications*. http://csrc.nist.gov/groups/STM/cavp/documents/skipjack/skipjack.pdf, Abruf: 10. Dezember 2012 (Zitiert auf Seite 22).

[35] National Security Agency: *NSA Suite B Cryptography*. http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml, Abruf: 07. Juli 2012 (Zitiert auf Seite 13).

[36] Ortega, Alfredo A.: *Implementation of the RSA algorithm in a 8-bit microcontroller*. https://sites.google.com/site/ortegaalfredo/pic18rsa, Abruf: 07. Oktober 2012 (Zitiert auf Seite 54).

[37] Paar, Christof ; Pelzl, Jan: *Understanding Cryptography: A Textbook for Students and Practitioners*. 1. Auflage. Berlin: Springer, 2010. – ISBN 978–3–642–04100–6 (Zitiert auf Seiten 6, 13, 14, 15, 17, 18, 19, 20, 24, 25, 27, 33, 36, 37, 38, 46, 47, 76, und 88).

[38] Poschmann, Axel ; Eisenbarth, Thomas: *Lightweightcrypto: Your ressource for everything related to efficient cryptography*. http://www.lightweightcrypto.org, Abruf: 07. November 2012 (Zitiert auf Seite 25).

[39] Rinne, S. ; Eisenbarth, T. ; Paar, C.: Performance Analysis of Contemporary Light-Weight Block Ciphers on 8-bit Microcontrollers. Version: 2007. http://www.lightweightcrypto.org/papers.php, Abruf: 10. Dezember 2012. In: *ecrypt workshop SPEED - Software Performance Enhancement for Encryption and Decryption*. -, 2007 (Zitiert auf Seite 71).

[40] Rivest, R. L. ; Shamir, A. ; Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. In: *Communications of the ACM* 21 (1978), S. 120 – 126 (Zitiert auf Seiten 48 und 49).

[41] Rivest, Ronald L. ; Leeuwen, J. V. (Hrsg.): *Cryptography: Handbook of Theoretical Computer Science*. Bd. 1. Amsterdam: Elsevier, 1990. – 717 – 755 S. (Zitiert auf Seite 5).

[42] Salomon, David: *Data Privacy and Security: Encryption and Information Hiding*. 1. Auflage. New York: Springer, 2003. – ISBN 978–0–38700–311–5 (Zitiert auf Seite 5).

[43] Schneier, Bruce: *Semantic Attacks: The Third Wave of Network Attacks*. http://www.schneier.com/crypto-gram-0010.html, Abruf: 10. Dezember 2012 (Zitiert auf Seiten 85 und 88).

[44] Schneier, Bruce: *Angewandte Kryptographie*. 1. Auflage. München: Pearson Studium, 2006. – ISBN 3–8273–7228–4 (Zitiert auf Seiten 13, 14, 16, 17, 34, 35, 36, 48, 49, 59, 62, und 74).

[45] Schneier, Bruce ; Kelsey, John ; Whiting, Doug ; Wagner, David ; Hall, Chris ; Ferguson, Niels: *Twofish: A 128-Bit Block Cipher*. http://www.schneier.com/paper-twofish-paper.pdf, Abruf: 03. Oktober 2012 (Zitiert auf Seite 42).

[46] Shannon, Claude E.: Communication Theory of Secrecy Systems. In: *Bell System Technical Journal* 28 (1949), Nr. 4, S. 656 – 715 (Zitiert auf Seiten 11 und 20).

[47] Shepherd, Simon: *Website about the The Tiny Encryption Algorithm*. http://143.53.36.235:8080/tea.htm, Abruf: 03. Oktober 2012 (Zitiert auf Seite 41).

[48] Singh, Simon: *Geheime Botschaften: Die Kunst der Verschlüsselung von der Antike bis in die Zeiten des Internet*. 2. Auflage. München: Deutscher Taschenbuch Verlag, 2002. – ISBN 3–423–33071–6 (Zitiert auf Seiten 8 und 9).

[49] Stinson, Douglas R.: *Cryptography*. 3. Auflage. Boca Raton: Chapmann & Hall, 2005. – ISBN 978–1–58488–508–5 (Zitiert auf Seiten 7, 11, 12, 19, 20, 24, 26, 39, 81, und 82).

[50] The White House: *Executive Order 13526 – Classified National Security Information*. http://www.whitehouse.gov/the-press-office/executive-order-classified-national-security-information, Abruf: 07. Juli 2012 (Zitiert auf Seite 13).

[51] Thorsberg, Frank: *PC World Poll Highlights Privacy Concerns*. http://www.pcworld.com/article/64824/article.html, Abruf: 10. Januar 2013 (Zitiert auf Seite 89).

[52] Uggedal, Eiving: *Social Navigation on the Social Web*, University of Oslo, Diplomarbeit, 2008 (Zitiert auf Seite b).

[53] Wheeler, David J. ; Needham, Roger M.: *Tea extensions*. http://www.cix.co.uk/~klockstone/tea.pdf, Abruf: 03. Oktober 2012 (Zitiert auf Seiten 44, 45, und 54).

[54] Wheeler, David J. ; Needham, Roger M.: TEA, a tiny encryption algorithm. In: *Fast Software Encryption: Second International Workshop* 1008 (1994), 363 – 366. http://www.cix.co.uk/~klockstone/tea.pdf, Abruf: 03. Oktober 2012 (Zitiert auf Seiten 41, 42, und 44).

[55] Whitman, Michael ; Mattord, Herbert ; Green, Andrew: *Guide to Firewalls and VPNs*. 3. Auflage. Clifton Park: Delmar Cengage Learning, 2011. – ISBN 978–1–111–13539–3 (Zitiert auf Seite 1).

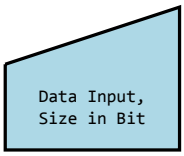
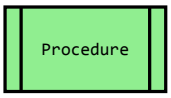
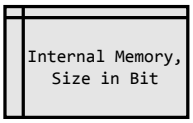
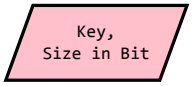
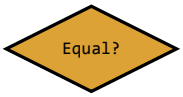
ABBILDUNGSVERZEICHNIS

QUELLCODEVERZEICHNIS

TABELLENVERZEICHNIS

ANHANG

Anhang 1: In der vorliegenden Arbeit verwendete Symbole

Symbole	Beschreibung
	Prozessbeginn oder Ende
	XOR-Verknüpfung
	Verbindung der Elemente
	Dateneingabe
	Interne Funktion
	Daten im internen Speicher
	Kryptografische Schlüsseln
	Entscheidung

Anhang 2: Quellcode der empirischen Untersuchung

```
#include <hidef.h> // for EnableInterrupts macro
#include "derivative.h" // include peripheral declarations

//For RAND
#include <stdlib.h>

#ifdef __cplusplus
  extern "C"
#endif

// Device initialization function declaration
void MCU_init(void);
__interrupt void isrVrtc(void);

// define if you have a fast memcpy function
#if 1
#  define HAVE_MEMCPY
#  include <string.h>
#  if defined( _MSC_VER )
#    include <intrin.h>
#    pragma intrinsic( memcpy )
#  endif
#endif

#include "aes.h"
#include "des.h"
#include "xtea.h"
#include "present.h"

//Timers
int Timer = 0;

int StartAES = 0;
int EndAES = 0;

int StartDES = 0;
int EndDES = 0;

```

```
39 int StartXTEA = 0;
40 int EndXTEA = 0;
41
42 int StartPRESENT = 0;
43 int EndPRESENT = 0;
44
45 int i = 0;
46 int iv[16];
47
48 //Variables for AES
49 unsigned char in[16];
50 unsigned char out[16];
51 unsigned char key[16];
52
53 //Variables for DES
54 unsigned char keyDES[8];
55 unsigned char blockDES[8];
56 keysched *ksDES;
57
58 //Variables for XTEA
59 unsigned long v[2];
60 unsigned long k[4];
61
62 //Variables for PRESENT
63 unsigned char plain[8];
64 unsigned char cipher[8];
65 unsigned char presentkey[16];
66 unsigned char result_present[8];
67
68 void main(void) {
69   MCU_init(); /* call Device Initialization */
70
71   //Initialise Arrays
72   for (i = 0; i < 16; i++) {
73     in[i] = 'a';
74     out[i] = 'b';
75     key[i] = 'c';
76     presentkey[i] = 'a';
77     iv[i] = 'a';
78   }
```

```
79
80  for (i = 0; i < 8; i++) {
81    keyDES[i] = 'a';
82    blockDES[i] = 'b';
83    plain[i] = 'a';
84    cipher[i] = 'a';
85    result_present[i] = 'b';
86  }
87
88  for (i = 0; i < 2; i++) {
89    v[i] = (unsigned long) i;
90  }
91
92  for (i = 0; i < 4; i++) {
93    k[i] = (unsigned long) i;
94  }
95
96  //Start AES
97  StartAES = Timer;
98    aes_set_key(&key[16], N_BLOCK, &ctx);
99    aes_encrypt(&in, out, &ctx);
100   //aes_decrypt(&out, in, &ctx); // Decryption
101 EndAES = Timer - StartAES;
102
103 //DES
104 StartDES = Timer;
105   fsetkey(keyDES, ksDES);
106   fencrypt(blockDES, 0, ksDES);
107   //fencrypt(blockDES, 1, ksDES); // Decryption
108 EndDES = Timer - StartDES;
109
110 //XTEA
111 StartXTEA = Timer;
112   encipher(v,k);
113   //decipher(v,k); // Decryption
114 EndXTEA = Timer - StartXTEA;
115
116 //PRESENT
117 StartPRESENT = Timer;
118   present_rounds(iv, presentkey, 31, cipher);
```

```
119     //for (i = 0; i < 8; i++){ // Decryption
120     //  cipher[i] = plain[i] ^ cipher[i];
121     //}
122     //present_rounds(iv, presentkey, 31, result_present);
123   EndPRESENT = Timer - StartPRESENT;
124
125   for(;;) {
126     /* __RESET_WATCHDOG(); By default COP is disabled with
           device init. When enabling, also reset the
          watchdog. */
127   } /* loop forever */
128   /* please make sure that you never leave main */
129 }
130
131 /*
132 ** ===================================
133 **     Interrupt handler : isrVrtc
134 **
135 **     Description :
136 **         User interrupt service routine.
137 **     Parameters : None
138 **     Returns    : Nothing
139 ** ===================================
140 */
141 __interrupt void isrVrtc(void)
142 {
143   RTCSC_RTIF = 1;
144   Timer++;
145 }
146 /* end of isrVrtc */
```

Quellcode 5.1: Versuchsaufbau für die empirische Untersuchung.

SELBSTÄNDIGKEITSERKLÄRUNG

Ich erkläre hiermit, dass ich die vorliegende Diplomarbeit selbständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Diplomarbeit in diesem Studiengebiet erstmalig einzureichen.

Berlin, 28. Januar 2013

Alexander Borisov

EINVERSTÄNDNISERKLÄRUNG

Ich bin damit einverstanden, dass Exemplare dieser Diplomarbeit in der Bibliothek der Humboldt-Universität zu Berlin ausgestellt werden.

Berlin, 28. Januar 2013

Alexander Borisov