Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Sven Kapitza

Implementierung einer Kryptanalyse in digitaler Hardware zur quantitativen Verifikation des Geschwindigkeitsvorteils gegenüber softwareseitigen Lösungen

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Sven Kapitza

Implementierung einer Kryptanalyse in digitaler Hardware zur quantitativen Verifikation des Geschwindigkeitsvorteils gegenüber softwareseitigen Lösungen

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Robert Fitz
Zweitgutachter : Prof. Dr. rer. nat. Thomas Lehmann

Abgegeben am 28. April 2010

Sven Kapitza

Thema der Diplomarbeit

Implementierung einer Kryptanalyse in digitaler Hardware zur quantitativen Verifikation des Geschwindigkeitsvorteils gegenüber softwareseitigen Lösungen.

Stichworte

Kryptographie, Kryptanlyse, digitale Hardware, Software, FPGA, AES, Brute-Force, Virtex 5, VHDL, MATLAB

Kurzzusammenfassung

Im Rahmen dieser Arbeit soll eine Kryptanalyse eines zuvor ausgewählten Algorithmus in digitaler Hardware umgesetzt werden. Im Anschluss daran soll diese Implementierung mit softwareseitigen Lösungen verglichen und auf einen Geschwindigkeitsvorteil untersucht werden.

Sven Kapitza

Title of the paper

Implementation of a cryptanalysis in digital hardware for a quantitative verification of the speed advantage compared with software solutions.

Keywords

Cryptography, cryptanalysis, hardware, software, FPGA, AES, brute-force, Virtex 5, VHDL, MATLAB

Abstract

The intention of this paper is a cryptanalysis of a previously selected algorithm in hardware. Furthermore this implementation should be compared to software solutions and a speed advantage should be verified.

Danksagung

Liebe Leserin, lieber Leser,

die Ihnen vorliegende Arbeit bildet den Abschluss meines Studiums an der Hochschule für Angewandte Wissenschaften Hamburg. An dieser Stelle möchte ich mich bei vielen Personen bedanken, ohne die diese Arbeit so nicht entstanden wäre.

Ich möchte meinen betreuenden Professor Herr Prof. Dr. Fitz danken, dass er mir die Möglichkeit für diese Arbeit gegeben und die Entstehung der Arbeit betreut hat. Des Weiteren danke ich Herrn Prof. Dr. Lehmann, der sich dazu entschlossen hat, diese Arbeit als zweiter Gutachter zu betreuen.

Besonderen Dank gilt meinen Kommilitonen, welche mich oft unterstützt haben, nicht nur während der Diplomarbeit, sondern auch während des kompletten Studiums. Besonders hervorheben möchte ich hier Jan, Stephan, Clemens, Axel, Jonas, Gerrit und Lars sowie Markus für einen entscheidenen Hinweis. Des Weiteren danke ich auch Martin für viele abwechslungsreiche und unterhaltsame Stunden. Mein Dank geht auch an Jenny und Thomas sowie allen anderen Korrekturlesern.

Des Weiteren möchte ich Benjamin Krill danken, welcher den Treiber für das LCD-Display geschrieben hat.

Einen besonders großen Dank möchte ich meiner Familie aussprechen, besonders meiner Mutter und meinem Vater, welche mich immer unterstützt haben, nicht nur während des Studiums, sondern auch während meines ganzen Lebens.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AES	Advanced Encryption Standard
BCD	Binary Coded Decimal
DES	Data-Encryption-Standard
DIP	dual in-line package
FPGA	Field Programmable Gate Array
GF(2^8**)**	Galois-Feld (2^8)
IEEE	Institute of Electrical and Electronics Engineers
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LSB	Least Significant Bit
LUT	Lookup-Table
MD5	Message-Digest Algorithm 5
MSB	Most Significant Bit
NIST	National Institute of Standards and Technology
OTP	One-Time-Password
PIN	Persöhnliche Identifikationsnummer
RSA	Rivest, Shamir und Adleman
S-Box	Substitution box
TAN	Transaktionsnummer
VHDL	Very High Speed Integrated Circuit Hardware Description Language
XOR	Antivalenz, Exklusiv-Oder

1 Einführung in die Kryptologie

1.1 Motivation zu dieser Arbeit

Die Kryptologie umfasst die Themenbereiche der Kryptographie und der Kryptanalyse. Die Anfänge der Kryptologie liegen hauptsächlich im militärischen Bereich. Schlachtpläne und Truppeninformationen mussten so übermittelt werden, dass sie für den Feind unbrauchbar waren, sollte dieser den Informationen habhaft werden. Mit der Verbreitung des Internets und dem Einzug von Computern in private Haushalte wurde der Bedarf an Sicherheit und Schutz der eigenen Daten vor unbefugten Zugriff immer größer. Dadurch umfasst die Kryptologie heutzutage weitere Themenbereiche wie digitale Signaturen, elektronisches Geld und Identifikationsprotokolle. Nun ist es natürlich interessant zu wissen, wie sicher Algorithmen in der heutigen Zeit sind und ob die verwendeten Verfahren auch in Zukunft den Sicherheitsanforderungen gerecht werden. Die Bedeutung des Wortes Sicherheit wird in dieser Arbeit genauer bestimmt. Der Kern der vorliegenden Arbeit ist jedoch eine Kryptanalyse in digitaler Hardware mit einem anschließendem Geschwindigkeitsvergleich gegenüber Softwarerealisierungen. Das Thema dieser Arbeit entstand aus der Idee heraus, dass Kryptanalysen in der Regel in Software realisiert werden und auf herkömmlichen Rechnerarchitekturen ausgeführt werden. Für eine Geschwindigkeitserhöhung werden zunemends größere Rechnerverbunde aufgebaut. Realisierungen in digitaler Hardware wurden bisher nur wenig genutzt.
In den folgenden Abschnitten wird eine kleine Einführung in die Kryptologie und deren Fachausdrücke gegeben. Die im Text genannten Beispielalgorithmen werden im Kapitel 2 noch genauer beschrieben. In Kapitel 3 wird der ausgewählte Algorithmus vorgestellt und die Kryptanalyse entwickelt. Im Anschluss daran wird im Kapitel 4 die Kryptanalyse auf einem Entwicklungsboard umgesetzt und mit Softwarelösungen verglichen.

1.2 Kryptographie

Die Kryptographie (oder auch Kryptografie) bezeichnet die Lehre der Absicherung einer Nachricht durch Verschlüsselung. Verschlüsselung oder auch Chiffrierung nennt sich das Verfahren, mit dem eine Nachricht, welche im Klartext vorliegt und somit leserlich ist, unverständlich gegenüber Dritten gemacht wird. Zur Rückgewinnung des ursprünglichen

Textes muss die Nachricht mit dem passenden Schlüssel entschlüsselt oder dechiffriert werden. Eine verschlüsselte Nachricht wird Ciphertext oder Chiffrat genannt, der Klartext wird auch als Plaintext bezeichnet.

Die Ziele der Kryptographie sind:

- **Geheimhaltung:** Unbefugte Personen sollen die Nachricht nicht lesen können.
- **Authentifizierung:** Der Empfänger der Nachricht muss deren Herkunft und somit auch den Absender verifizieren können.
- **Integrität:** Der Empfänger der Nachricht sollte erkennen können, ob die Nachricht während der Übermittlung verändert oder gar ausgetauscht wurde.
- **Verbindlichkeit:** Der Sender der Nachricht kann nicht leugnen, dass die Nachricht von ihm ist.

Eine der wichtigsten Thesen der Kryptographie wurde im Jahre 1883 von dem Niederländer Auguste Kerckhoffs aufgestellt. Das nach ihm benannte Prinzip sagt aus, dass die Sicherheit einer Nachricht nicht von der Geheimhaltung des Algorithmus abhängen sollte, sondern von der Wahl beziehungsweise der Geheimhaltung des Schlüssels [4], [Sch96]. Durch eine Veröffentlichung von neuen Algorithmen können diese schon vor einer eventuellen kommerziellen oder massenhaften Nutzung von Kryptanalytikern auf Schwachstellen untersucht werden. Diese Methode der Entwicklung von Algorithmen bezeichnet man als starke Kryptographie. Als ein Beispiel dafür sei hier der Advanced Encryption Standard (AES) genannt. Bei der schwachen Kryptographie hingegen ist die Sicherheit einer Nachricht von der Geheimhaltung des Algorithmus abhängig. Ein Beispiel für einen geheimen Algorithmus ist der RC4, welcher noch sieben Jahre nach seiner Entwicklung nicht öffentlich war.

Für die Sicherheit eines Algorithmus trifft im Normalfall einer der folgenden Punkte zu:

- **Unsicher:** Eine Verschlüsselung mit einem unsicheren Algorithmus kann leicht dechiffriert werden. Angriffe auf verschlüsselte Nachrichten sind erfolgreich.
- **Berechnungssicher:** Ein Algorithmus ist dann berechnungssicher, wenn ein damit chiffrierter Text mit den zur Verügung stehenden Mitteln und in einer absehbarer Zeit nicht entschlüsselt werden kann. So ein Algorithmus sollte auch für einen längeren Zeitraum berechnungssicher bleiben.
- **Uneingeschränkt sicher:** Ein Algorithmus gilt erst dann als uneingeschränkt sicher, wenn ein Klartext nicht dechiffriert werden kann, obwohl Chiffrat in unbegrenzter Menge vorhanden ist. Das einzige Beispiel hierfür ist das One-time-pad.

Während die uneingeschränkte Sicherheit eines Algorithmus mathematisch bewiesen wird, werden bei der Betrachtung der Berechnungssicherheit die aktuellen Ressourcen berücksichtigt. Hierbei werden hauptsächlich drei Aspekte betrachtet. Zum Ersten wird der Rechenaufwand abgeschätzt. Dieser ist abhängig von der zur jeweiligen Zeit verfügbaren Hardware. So werden heutige, berechnungssichere Algorithmen mit der Entwicklung von Quantencomputern wahrsscheinlich als unsicher eingestuft werden. Ein weiterer Aspekt ist die Speicheranforderung. Hierbei wird abgeschätzt, wie viel Daten bei einem Angriff gespeichert werden müssen. Zum Dritten muss auch die Menge an vorhandenen Daten berücksichtig werden, denn für einen Angriff müssen ausreichend Eingangsdaten zur Verfügung stehen.
Alle Aspekte sind jedoch auch relativ zu der Nachricht zu betrachten, welche chiffriert werden soll. So gilt eine Verschlüsselung als sicher, wenn der finanzielle Aufwand, der betrieben werden muss, um ein Chiffrat zu entschlüsseln, höher ist als der Wert der Nachricht selbst. Ein anderes Beispiel ist, wenn eine verschlüsselte Nachricht zwar in wenigen Stunden dechiffriert werden kann, der Inhalt der Nachricht dann aber zu diesem Zeitpunkt nicht mehr aktuell ist. In diesem Fall gilt der Algorithmus für diese Nachricht als berechnungssicher.

1.3 Basistechniken der Kryptographie

In den nachfolgenden Abschnitten werden einige kryptographische Grundtechniken erklärt. Eingegangen wird sowohl auf symmetrische als auch auf asymmetrische Algorithmen. Des Weiteren zählen zu den Basistechniken die digitalen Signaturen und die Generierung von Zufallszahlen beziehungsweise die Generierung von Pseudozufallszahlenfolgen. Bei den digitalen Signaturen wird die eigenhändige Unterschrift auf einem digitalen System abgebildet, um damit zum Beispiel elektronische Nachrichten zu versehen. Eine digitale Signatur soll, wie eine manuelle Unterschrift, einzigartig, fälschungssicher, unveränderbar und verbindlich sein. Zufallszahlen werden benötigt, um mit Hilfe von eben diesen einen guten Verschlüsselungsalgorithmus zu erhalten. Aufgrund von zufälligen Elementen soll eine Kryptanalyse nicht möglich sein. Das Problem ist jedoch, dass es in der digitalen Datenverarbeitung keinen Zufall gibt, da diese Zahlengeneratoren auch einem Algorithmus unterliegen und die Zufallszahlen somit berechnet werden. Man spricht deshalb von Pseudozufallszahlen. Es soll aber erreicht werden, dass die generierten Zahlen im Bezug zum Verwendungszweck als zufällig gelten. Im Rahmen dieser Arbeit werden beide Themen jedoch zurückgestellt, da sich zum einen Schnittmengen mit den anderen Bereichen bilden und zum anderen beide Themen für diese Arbeit als Randgebiete anzusehen sind.

1.3.1 Symmetrische Verfahren

Von einem symmetrischen Verschlüsselungsverfahren spricht man, wenn sowohl der Sender als auch der Empfänger einer Nachricht den gleichen Schlüssel besitzen. Ein Geheimtext wird also mit dem gleichen Schlüssel dechiffriert, mit dem er auch erstellt wurde. Symmetrische Verfahren sind meistens rundenbasierend. Eine Runde ist eine Abfolge von mehreren Operationen. Diese werden mehrmals in der gleichen Reihenfolge nacheinander durchlaufen. Nachteilig bei symmetrischen Verfahren ist, dass der geheime Schlüssel vorher über einem sicheren Kanal oder bei einer direkten Zusammenkunft der Kommunikationspartner ausgetauscht werden muss.

1.3.2 Asymmetrische Verfahren

Asymmetrische Verfahren werden auch Public-Key-Verfahren genannt. Im Gegensatz zu symmetrischen Verfahren werden hierbei zwei Schlüssel benötigt, ein öffentlicher Schlüssel und ein privater Schlüssel. Der öffentliche Schlüssel ist frei zugänglich und kann von jeder Person benutzt werden, um eine Nachricht zu verschlüsseln. Der private Schlüssel ist jedoch geheim und wird nur benötigt, um eine codierte Nachricht zu dechiffrieren. Genannt sei an dieser Stelle noch, dass dieses Verfahren bei digitalen Signaturen umgekehrt angewandt wird. Ein Dokument wird mit dem privaten Schlüssel chiffriert und mit dem öffentlich zugänglichem Schlüssel kann jeder Anwender das Dokument entschlüsseln. Ein weiteres Merkmal der asymetrischen Kryptographie ist, dass sich der private Schlüssel nicht aus dem öffentlichen Schlüssel ableiten lässt.

1.3.3 Kryptographische Protokolle

Ein normales Protokoll enthält mehrere Anweisungen für eine Kommunikation zwischen zwei oder mehreren Parteien. Mit dieser Kommunikation soll eine bestimmte Aufgabe abgeschlossen werden. Jeder Teilnehmer kennt das Protokoll und ist auch verpflichtet, sich an die Anweisungen zu halten. Ein Betrugsversuch wird bei einem korrekten Protokoll sofort erkannt. Anweisungen dürfen nicht fehlen oder mehrdeutig sein. Jede Aktion ist genau festgelegt. Des Weiteren sollte das Protokoll so abstrakt gehalten werden, dass es prinzipiell in jeder Umgebung verwendet werden kann. Bei einem kryptographischen Protokoll werden, im Gegensatz zu gewöhnlichen Protokollen, kryptographische Algorithmen verwendet. Das Ziel eines solchen Protokolls ist oft die Identifikation der Teilnehmer und die Vermeidung von Betrug. Sollte ein Teilnehmer dennoch betrügen, so wird der Versuch durch das Protokoll aufgedeckt. Dadurch können zum Beispiel Verträge zwischen Teilnehmern abgeschlossen werden, welche sich gegenseitig nicht vertrauen.

1.3.4 Einwegverschlüsselung

Als Einwegverschlüsselung wird ein Algorithmus bezeichnet, bei dem eine Verschlüsselung schnell und einfach berechenbar ist, eine Entschlüsselung ist jedoch nur mit erheblich größerem Aufwand möglich. Als Beispiel sei die Multiplikation zweier Primzahlen genannt. Diese Berechnung ist einfach, jedoch die Primfaktorzerlegung als Umkehrfunktion ist schwer berechenbar. Einwegfunktionen kommen bei der Passwortverschlüsselung zur Anwendung. Für jeden Benutzer eines Systems wird das Passwort verschlüsselt und gespeichert. Möchte sich ein Benutzer nun am System anmelden, so wird sein Passwort verschlüsselt und mit dem gespeicherten Passwort verglichen. Eine Entschlüsselung des Ciphertextes ist nicht erforderlich. Wenn eine Einwegfunktion jedoch mit Hilfe einer Zusatzinformation leicht umgekehrt werden kann, dann spricht man von einer Falltürfunktion. Solche Funktionen werden in asymmetrischen Verschlüsselungsverfahren angewendet. Als Beispiel sei hier der RSA-Algorithmus genannt.

1.3.5 Challenge-Response-Verfahren und Zero-Knowledge-Verfahren

Bei dem Challenge-Response-Verfahren beweist ein Teilnehmer seine Authentizität mit einem Passwort, jedoch ohne dieses Passwort zu übermitteln. Eine prüfende Partei sendet einen Zufallswert (Challenge) an den Teilnehmer, dieser verschlüsselt diesen Wert mit einem Algorithmus und seinem Passwort und sendet diesen Wert zurück zum Prüfer (Response). Die Prüfende Partei kennt zu jedem Teilnehmer das Passwort und verschlüsselt nun ebenfalls die Zufallszahl mit dem zugehörigen Passwort des Teilnehmers, welcher sich legitimieren soll. Beide verschlüsselten Texte werden verglichen und wenn diese gleich sind, so hat der Teilnehmer erfolgreich seine Identität bewiesen, ohne sein Passwort zu senden. Zu beachten ist, dass bei diesem Verfahren keine Entschlüsselung der Ciphertexte vorgesehen ist.
Challenge-Response-Verfahren existieren in den unterschiedlichsten Ausführungen. So kann die Challenge ein von der Zeit abhängiger Wert sein oder der Response ist nur für eine kurze Zeit gültig. Eine häufige Verwendung findet was Verfahren beim Online-Banking. Für die Berechnung des Response werden hier kleine Hardwaregeräte, so genannte OTP-Token (**O**ne-**T**ime-**P**assword) verwendet. Diese Token existieren in den unterschiedlichsten Ausführungen, zum Beispiel mit und ohne Chipkartenleser.
Bei dem Zero-Knowledge-Verfahren wird ebenfalls der Beweis für ein Geheimnis erbracht, ohne jedoch dieses zu verraten. Das bekannteste Beispiel stammt aus der Geschichte. Der italienische Mathematiker Nicolo Tartaglia entdeckte im Jahr 1535 eine Formel zur Lösung von kubischen Gleichungen. Als Beweis seiner Lösung löste er innerhalb relativ kurzer Zeit 30 Gleichungen. Dadurch war seine Rechenmethode bewiesen, ohne dass Tartaglia die Formel veröffentlicht hat.

1.3.6 Einmal-Passwort

Bei einem Einmal-Passwort oder auch Einmalkennwort wird der Schlüssel nur einmal verwendet. Danach ist dieser ungültig. Sollte also der Schlüssel bei der Verwendung abgefangen werden, so kann dieser trotzdem nicht mehr verwendet werden. Genutzt werden Einmal-Passwörter meistens für eine Authentisierung. Zu beachten ist hierbei, dass die Sicherheit des Schlüssels weiterhin vom verwendeten Algorithmus abhängig ist. In der Praxis wird das Verfahren häufig im Bankwesen eingesetzt. Hier wird dem Kunden eine Liste mit Passwörtern, den so genannten TANs (**T**rans**a**ktions**n**ummer) gegeben, welche dieser beim Online-Banking verwenden kann. Der Verlust dieser Liste ist allerdings fatal, denn dadurch können Unbefugte sich als eine andere Person ausgeben. Aus diesem Grund werden Einmalkennwörter häufig erst kurz vor dem Gebrauch generiert. Ein Passwortgenerator kann außerdem noch eine zeitliche Abhängigkeit in den Schlüssel mit einbeziehen. Dadurch wird der generierte Schlüssel nur für ein bestimmtes Zeitfenster von der Gegenstelle akzeptiert. Ein Zeitfenster wird benötigt, weil eine exakte, zeitliche Synchronisation nicht statt finden kann. So wird zum Beispiel die prüfende Stelle einen Schlüssel bekommen, welcher zu dem Zeitpunkt der Überprüfung ein paar Minuten vorher, oder aber auch später, wenn die Uhrzeit des Teilnehmers nicht richtig ist, erstellt wurde. Des Weiteren kann hierbei auch das Challenge-Response-Verfahren verwendet werden. Hierbei schickt die prüfende Partei einen Zahlenwert und erwartet ein Passwort, welches mit diesem Wert erstellt wurde. Dieses Verfahren muss zeitlich nicht synchronisiert werden.

1.3.7 Hashfunktionen

Eine Hashfunktion im Allgemeinen ist eine Funktion, welche aus einer großen Menge an Werten eine kleinere Menge als Abbild erzeugt. Der Name leitet sich aus dem englischen Verb *to hash* (dt. zerhacken, kleinschneiden) ab. Eine kleine Menge von Werten, der so genannte Hashwert, soll eine eindeutige Erkennungsmarke der großen Datenmenge sein. Vergleichbar ist der Hashwert mit dem Fingerabdruck eines Menschen. Bei einer Hashfunktion kommt es allerdings vor, dass ein Hashwert auf mehrere mögliche Quellwerte hindeuten kann. Dieser Fall wird Kollision genannt. Anwendungen für Hashfunktionen sind zum Beispiel Prüfsummen und digitale Signaturen. Prüfsummen werden erstellt, um eine absichtliche oder zufällige Veränderung von Daten erkennen zu können. Eine Veränderung ist erkennbar, wenn sich eine errechnete Prüfsumme von der originalen Prüfsumme unterscheidet. Für digitale Signaturen können ebenfalls Hashfunktionen verwendet werden. Diese Funktionen sind dann ebenfalls Einwegfunktionen. Als Beispiel für eine Hashfunktion sei MD5 genannt, welcher lange Zeit die gebräuchlichste Hashfunktion war. Bei einem Challenge-Response-Verfahren kann der Response auch ein Hashwert sein.

1.4 Kryptanalyse

Die Kryptanalyse (oder auch Kryptoanalyse) befasst sich mit der Wiedergewinnung eines Textes aus einer verschlüsselten Nachricht. Hierbei wird entweder nach einem passenden Schlüssel für die Dechiffrierung gesucht, oder es kann untersucht werden, ob die Nachricht auch ohne einen Schlüssel aus dem chiffrierten Text erstellt werden kann. Die Kryptanalyse wird auch verwendet, um die Sicherheit eines kryptographischen Algorithmus zu testen. Wenn versucht wird, einen chiffrierten Text ohne das Kennwort zu entschlüsseln, dann wird von einem Angriff gesprochen. Hierbei wird unterschieden zwischen dem passiven Angriff und dem aktiven Angriff. Bei dem passiven Angriff wird zum Beispiel der Schlüssel abgefangen. Hingegen wird von einem aktiven Angriff gesprochen, wenn ein direkter Einfluß ausgeübt wird. Zum Beispiel kann durch Einbringen von Informationen versucht werden, den passenden Schlüssel zu extrahieren. Eine andere Möglichkeit ist es, Fehlinformationen zu verbreiten, welche dann an anderer Stelle als Wahrheit angenommen werden. Des Weiteren wird bei der Kryptanalyse unterschieden zwischen Entziffern und Entschlüsseln. Beim Entziffern (umgangssprachlich spricht man hierbei von brechen oder knacken) wird versucht, eine Nachricht ohne den Schlüssel zu dechiffrieren. Beim Entschlüsseln ist der Schlüssel vorhanden und es wird der Klartext aus der chiffrierten Nachricht erstellt.

1.4.1 Angriffe gegen Algorithmen

Bei der Kryptanalyse gibt es verschiedene Möglichkeiten eines Angriffs. Die folgende Auflistung wurde aus [Sch96] entnommen und erweitert. Diese Liste ist zum Teil deckungsgleich mit anderen Auflistungen aus der Fachliteratur.

- **Ciphertext-only-Angriff:** Bei diesem Angriff liegen mehrere verschlüsselte Nachrichten vor. Es wird versucht, daraus den Klartext oder sogar den Schlüssel, mit dem diese Nachrichten erstellt wurden, zu gewinnen.
- **Known-plaintext-Angriff:** Bei dieser Methode liegen sowohl der Klartext, als auch der dazugehörige chiffrierte Text vor. Durch diese beiden Texte wird versucht, den dazugehörigen Schlüssel zu rekonstruieren. Alternativ kann ein Algorithmus gesucht werden, welcher das Vorhandensein des Schlüssels überflüssig werden lässt, indem der verschlüsselte Text direkt in einen Klartext umgewandelt werden kann.
  Ein **Brute-Force-Angriff** (dt. rohe Gewalt), ein Angriff bei dem alle möglichen Schlüssel ausprobiert werden, gehört zu der Gruppe der known-plaintext-Angriffen.
- **Chosen-plaintext-Angriff:** Dieser Angriff ähnelt der vorherigen Methode. Hierbei kann nun aber der Klartext frei gewählt werden. Der Vorteil hierbei ist, dass gezielt nach verschlüsselten Textpassagen überprüft werden kann. Anhand dessen lassen sich dann gegebenenfalls Rückschlüsse auf den Schlüssel ziehen oder es lässt sich ein Algorithmus finden, der die Nachricht in Klartext umwandeln kann.

- **Adaptive-chosen-plaintext-Angriff:** Diese Methode besagt nur, dass die Ergebnisse eines Chosen-plaintext-Angriffs für einen neuen Angriff berücksichtigt werden können. Zum Beispiel können zuerst häufig vorkommende Worte wie *der*, *das*, *eine* untersucht werden und darauf aufbauend werden dann längere Textpassagen chiffriert und anschließend untersucht.
- **Chosen-ciphertext-Angriff:** Bei diesem Verfahren stehen verschiedene chiffrierte Texte sowie deren Klartexte zur Verfügung. Anhand beider Quellen wird dann versucht, den passenden Schlüssel zu bekommen.
- **Chosen-key-Angriff:** Hierbei liegen verschiedene Informationen über die Zusammenhänge von mehreren Schlüsseln vor. Anhand dessen wird versucht, einen Schlüssel zu erstellen.
- **Public-key-only-Angriff:** Dieses Verfahren ist für asymmetrische Verschlüsselungen gedacht. Hierbei kann ein Angreifer mit dem öffentlichen Schlüssel einen Klartext verschlüsseln und somit den Klartext mit dem Geheimtext vergleichen.
- **Kryptanalyse mit Gewalt:** Durch Bedrohungen, Erpressungen oder ähnlichen kriminellen Handlungen wird versucht, in Besitz des Schlüssels zu kommen. Diese Methode ist in den meisten Fällen sehr wirkungsvoll, da: *Bei den Verfahren der [...] Kryptographie stellt meist der Mensch als Besitzer des Schlüssels die größte Sicherheitslücke dar.* [Ert07]

1.4.2 Man-in-the-middle Angriff

Bei einem Man-in-the-middlie-Angriff befindet sich ein Angreifer zwischen den Parteien, ohne das diese davon Kenntnis haben. Diese Angriffsart eignet sich besonders gut für Public-key-Algorithmen. Hierbei fängt der Angreifer die öffentlichen Schlüssel der Parteien ab und leitet dafür seinen eigenen öffentlichen Schlüssel jeweils an die Kommunikationspartner weiter. Wenn nun eine Nachricht verschlüsselt wurde, so kann der Angreifer sie mit seinem eigenen privaten Schlüssel öffnen, da für die Verschlüsselung der Nachricht sein öffentlicher Schlüssel benutzt wurde. Dadurch ist es dem Angreifer möglich, geheime Texte zu lesen und sogar falsche Informationen einzubringen. Damit das Vorhandensein des Angreifers nicht entdeckt wird, müssen alle Nachrichten über den Angreifer gesendet werden, da sonst die Kombination aus öffentlichen und privaten Schlüsseln nicht mehr übereinstimmt. Diese Art von Angriff ist auch für Netzwerke verbreitet, hierbei gibt sich der Angreifer zum Beispiel als einen Access Point (dt. Zugangspunkt) aus und kann so den Datenverkehr mitlesen.

2 Analyse von Algorithmen

In diesem Kapitel werden verschiedene Algorithmen kurz beschrieben. Zum einen sind Verfahren als historische Beispiele mit Kryptanalyse aufgelistet, zum anderen werden Beispiele für aktuelle Algorithmen genannt. Es wird kurz beschrieben, wie der jeweilige Algorithmus funktioniert und wo dieser eingesetzt wird beziehungsweise wurde. Außerdem werden in diesem Kapitel weitere Fachbegriffe eingeführt und es wird ein allgemeiner Überblick über die Kryptologie gegeben.

2.1 Caesar-Verschlüsselung

Bei dieser Verschlüsselung handelt es sich um eine einfache Verschiebung der Buchstaben im Alphabet. Dieses wird auch als monoalphabetischer Substitutionschiffre oder Verschiebechiffre bezeichnet. Der Name geht zurück auf den römischen Feldherren Gaius Julius Caesar, welcher diese Verschlüsselung für seine militärischen Nachrichten verwendete. Caesar nutzte für die Chiffrierung seine Befehle eine Verschiebung um drei Stellen im Alphabet. Für Buchstaben am Ende des Alphabets wird ein Umbruch an den Anfang vollzogen, so wird zum Beispiel bei einer Verschiebung um drei Stellen das X auf das A abgebildet. Um eine so verschlüsselte Nachricht wieder zu dechiffrieren, muss nur die Verschiebung rückgängig gemacht werden. Hierfür muss natürlich bekannt sein, um wie viele Stellen die Buchstaben verschoben wurden.

2.1.1 Kryptanalyse der Caesar-Verschlüsselung

Diese Art von Verschlüsselung gehört zu den unsicheren Algorithmen. Der größte Nachteil dieser Methode ist, dass es nur 25 verschiedene Schlüssel gibt, welche für diesen Algorithmus verwendet werden können. Das bedeutet, dass maximal 25 Versuche benötigt werden, um das Chiffrat zu entziffern.
Ein anderer Nachteil ist, dass der Verschiebechiffre anfällig für eine Häufigkeitsanalyse der Buchstaben ist. Für jeden Buchstaben in einem Text gibt es eine prozentuale Häufigkeit die angibt, wie oft dieser Buchstabe verwendet wird. Als Beispiel sei die deutsche Sprache genannt. Hier werden viele Vokale in Sätzen verwendet. Wenig Gebrauch wird jedoch von

den Buchstaben *x*, *y* oder *z* gemacht. Die Häufigkeiten für die einzelnen Buchstaben sind bekannt. Die Tabelle 2.1 zeigt die Häufigkeiten der Buchstaben der deutschen Sprache.

Buchstabe	Häufigkeit	Buchstabe	Häufigkeit
E	17,40 %	M	2,53 %
N	9,78 %	O	2,51 %
I	7,55 %	B	1,89 %
S	7,27 %	W	1,89 %
R	7,00 %	F	1,66 %
A	6,51 %	K	1,21 %
T	6,15 %	Z	1,13 %
D	5,08 %	P	0,79 %
H	4,76 %	V	0,67 %
U	4,35 %	J	0,27 %
L	3,44 %	Y	0,04 %
C	3,06 %	X	0,03 %
G	3,01 %	Q	0,02 %

Tabelle 2.1: Häufigkeiten der Buchstaben der deutschen Sprache, [Beu09].

Bei dem Verschiebechiffre bleibt die Häufigkeit der Buchstaben an sich erhalten, jedoch verschiebt sich diese ebenfalls um den Wert, mit dem die Nachricht verschlüsselt wird. Anhand einer Häufigkeitsanalyse lassen sich also Rückschlüsse ziehen, welcher Buchstabe um wie viel Stellen verschoben wurde, indem der Buchstabe mit dem häufigsten Vorkommen dem *E* zugeordnet wird.
Für eine erfolgreiche Kryptanalyse ist es zudem hilfreich, wenn die Leerzeichen zwischen den Wörtern nicht ebenfalls verschoben oder gar entfernt wurden. Denn dadurch, dass die Länge von einzelnen Wörtern bekannt ist, lassen sich häufig verwendete Worte wie *der*, *die* oder *das* gezielt dechiffrieren.

2.1.2 Varianten des Algorithmus

Es gibt viele verschiedene Variationen von Substitutionschiffren. So können Buchstaben willkürlich auf andere Buchstaben abgebildet werden, es muss sich hierbei nicht um eine Verschiebung handeln. In diesem Fall spricht man dann von einer einfachen monoalphabetischen Substitution. Eine andere Möglichkeit ist es, einzelne Buchstaben gegen eine Gruppe von Buchstaben auszutauschen.

Als eine Variante der Caesar-Verschlüsselung sei hier der **ROT13**-Algorithmus genannt. Hierbei handelt es sich um eine Caesar-Verschlüsselung mit der Zahl 13 als Schlüssel. Da das lateinische Alphabet aus 26 Buchstaben besteht, kann mit einer Anwendung des Algorithmus eine Nachricht entweder verschlüsselt oder ein Chiffrat entschlüsselt werden. Der ROT13-Algorithmus funktioniert sogar unabhängig von der Schieberichtung. Angewandt wurde diese Art von Verschlüsselung erstmals im Usenet und wird heutzutage immernoch verwendet. Allerdings ist die Verwendung nicht zur geheimen Nachrichtenübertragung gedacht. Stattdessen können so nicht gerne gelesene Worte verschleiert werden oder es kann ein Witz mit einer Pointe gleichzeitig übertragen werden, ohne dass die Pointe sofort sichtbar ist. Auf einem UNIX-System kann für ROT13 die Standardfunktion `tr` verwendet werden.

2.2 Vigenère-Verschlüsselung

Die Vigenère-Verschlüsselung leitet sich teilweise aus der vorherigen Caesar-Verschlüsselung ab. Der Algorithmus wurde von dem Franzosen Blaise de Vigenère entwickelt und im Jahr 1586 veröffentlicht. Es dauerte ungefähr 300 Jahre, bis eine erfolgreiche Kryptanalyse durchgeführt werden konnte.
Bei dieser Art von Verschlüsselungen können gleiche Buchstaben aus dem Klartext in unterschiedliche Buchstaben im Chiffrat abgebildet werden. Dieses wird durch die Verwendung von verschiedenen Alphabeten realisiert. Die Vigenère-Verschlüsselung ist somit eine polyalphabetische Verschlüsselung. Um mehrere Alphabete zu erstellen, wird das lateinische Alphabet um jeweils eine Stelle mehrmals hintereinander verschoben. Dadurch entsteht das so genante Vigenère-Quadrat (A.1). Mit einem Schlüsselwort kann nun ein Klartext verschlüsselt werden, indem in der Tabelle nach den chiffrierten Buchstaben gesucht wird. Die Spalten bilden hierbei die Buchstaben des zu verschlüsselnden Textes und die Zeilen sind die Buchstaben des Schlüsselwortes. Wenn der Schlüssel kürzer als die Nachricht ist, so wird dieser wiederholt, bis alle Zeichen chiffriert wurden.

2.2.1 Kryptanalyse der Vigenère-Verschlüsselung

Für eine erfolgreiche Kryptanalyse ist die Ermittlung der Schlüssellänge eine Vorraussetzung. Nur dadurch kann dann eine Häufigkeitsanalyse für jedes einzelne Alphabet erfolgen. Denn wenn die Schlüssellänge bekannt ist, kann jedes Zeichen des Chiffrates einem Alphabet zugeordnet werden. Für die Ermittlung der Schlüssellänge gibt es zwei Verfahren, die kombiniert angewandt werden. Beide Verfahren werden nachfolgend kurz erläutert. Diese können auch für andere Algorithmen verwendet werden, aber die Anwendung bei der Vigenère-Verschlüsselung ist charakteristisch.

Kasiski-Test

Der Kasiski-Test wurde von dem englischen Mathematiker Charles Babbage im Jahr 1854 entwickelt, um die Vigenère-Verschlüsselung zu brechen. Allerdings wurde das Verfahren nicht von Babbage veröffentlicht, sondern erst neun Jahre später vom preussischen Infanteriemajor Friedrich Kasiski, welcher dieses Verfahren ebenfalls entdeckte.
Der Kasiski-Test gründet in der Tatsache, dass Wiederholungen im Klartext auch als Wiederholungen im Chiffrat auftreten, sofern der Abstand der beiden Textstellen ein Vielfaches der Schlüssellänge ist. Für das Verfahren werden alle Wiederholungen von Zeichenfolgen mit einer Mindestlänge von drei erfasst und deren Abstand notiert. Die Ergebnisse werden in ihre Primfaktoren zerlegt. Anhand der Primfaktoren kann die Schlüssellänge bestimmt werden. Diese ist entweder die am häufigsten auftretende Primzahl oder ein Vielfaches von zwei häufigen Primzahlen.

Friedman-Test

Der Friedman-Test wurde von dem Amerikaner William Frederick Friedman entwickelt und um 1925 veröffentlicht. Der Ansatz für das Verfahren ist, dass in einem Text die Buchstaben nicht zufällig verteilt sind. Zuerst wird in einem Text die Wahrscheinlichkeit, dass zwei zufällig gewählte Buchstaben gleich sind, berechnet. Diese Wahrscheinlichkeit wird Koinzidenzindex I genannt und wird mit der Formel (2.1) berechnet.

$$I = \frac{\sum_{i=1}^{26} m_i(m_i - 1)}{m(m-1)} \tag{2.1}$$

Die Variable m steht für die Anzahl aller Buchstaben in einem Text. Für lange deutsche Texte gilt ein durchschnittlicher Koinzidenzindex von $I_d = 0,0762$. Für einen völlig zufälligen Text, bei dem die Buchstaben gleichverteilt sind, ergibt sich ein Koinzidenzindex $I_r = \frac{1}{26} \approx 0,0385$. Anhand der nachfolgenden Formel (2.2) lässt sich die Schlüssellänge k näherungsweise bestimmen. Allerdings lässt sich bei dieser Approximation selten die genaue Schlüssellänge bestimmen, jedoch kann zusammen mit dem Ergebnis aus einem Kasiski-Test ein exaktes Ergebnis geliefert werden.

$$k \approx \frac{(I_d - I_r)m}{I \cdot (m-1) - I_r m + I_d} = \frac{0,0377m}{I \cdot (m-1) - 0,0385m + 0,0762} \tag{2.2}$$

2.3 One-Time-Pad

Das One-Time-Pad wurde im Jahr 1917 und 1918 von den Amerikanern Gilbert Vernam und Joseph O. Mauborgne entwickelt und basiert auf den Vigenère-Algorithmus. Das Verschlüsselungsverfahren ist uneingeschränkt sicher, es kann auch unabhängig von der Hardware nicht entschlüsselt werden. Des Weiteren ist der Algorithmus ein Beispiel für ein Stromchiffre. Das bedeutet, dass jedes Zeichen einzeln codiert wird. Das ist besonders für Echtzeitanwendungen vom Vorteil, da eine Verschlüsselung sehr schnell abgearbeitet werden kann. Das Verschlüsselungsverfahren ist einfach. Jedes Zeichen des Klartextes wird mit einem Zeichen des Schlüssels verknüpft. Dadurch muss der Schlüssel genau so lang sein wie die Nachricht selbst. Ferner muss der Schlüssel zufällig gewählt sein und darf auch nur einmal verwendet werden. In der Praxis wird ein sehr langer Schlüssel generiert und ausgetauscht. Von diesem Schlüssel wird dann die benötigte Anzahl an Zeichen abgenommen, wenn eine Nachricht verschlüsselt werden soll. Bei der Entschlüsselung werden dann, sofern kein Fehler aufgetreten ist, die gleichen Zeichen aus dem Schlüssel verwendet. Die Realisierung des Algorithmus erfolgt binär mit einer XOR-Verknüpfung.

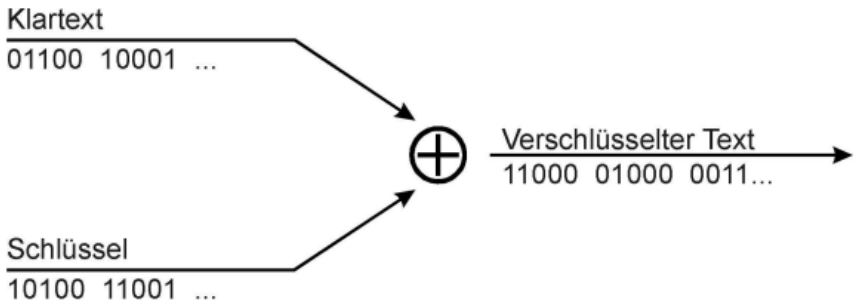

Abbildung 2.1: Das Verfahren des One-Time-Pads, Quelle: [1].

Die Sicherheit des One-Time-Pads lässt sich an einem Beispiel verdeutlichen. Wenn das Wort *Mord* mit einem zufälligen Schlüssel chiffriert wird, dann entsteht ein Chiffrat *FJEL*. Sollte nun versucht werden, dieses zu entschlüsseln, kann der Klartext nicht mehr eindeutig rekonstruiert werden. Statt *Mord* sind auch Wörter wie *Maus*, *Mama* oder *Bier* zulässig.
Besonders während des Kalten Krieges wurde das One-Time-Pad zur Absicherung der Fernschreiberverbindung zwischen der USA und der Sowjetunion, dem so genannten Roten Telefon, benutzt. Das größte Problem ist allerdings der unhandliche Schlüssel des One-Time-Pads. Es muss ein sehr langer und möglichst zufälliger Schlüssel generiert und gespeichert werden. Speicherung, Schlüsselaustausch zwischen den Teilnehmern und Handhabung machen das One-Time-Pad unhandlich. Ein Lösungsansatz ist, dass eine echte Zufallsfolge von Bits, welche mit einer physikalischen Quelle generiert wird, von einem Satelliten permanent gesendet wird. Um eine Nachricht zu verschlüsseln, kann nun die Bitfolge vom Satelliten

genommen werden. Wenn dieser Ciphertext dechiffriert werden soll, so muss die beteiligte Partei nur den Zeitpunkt wissen, ab dem die Bitfolge vom Satelliten verwendet wird. Der Startzeitpunkt ist dann der eigentliche Schlüssel und muss geheim bleiben. Durch diese Vorgehensweise muss kein großer Schlüssel vorab gespeichert und ausgetauscht werden. Eine praktische Realisierung ist bisher noch nicht erfolgt [5].

2.4 RC4

Der RC4-Algorithmus ist ein weiteres Beispiel für ein Stromchiffre. Der Algorithmus wurde im Jahr 1987 von dem Amerikaner Ronald Linn Rivest entwickelt. Der Name RC ist eine Abkürzung für Ron's Code. Dieser Algorithmus wurde geheim gehalten, bis er anonym in einer Mailing-Liste im Jahr 1994 veröffentlicht wurde [6]. Der Algorithmus wird unter anderem bei HTTPS, WEP und WPA angewendet. Nachfolger von RC4 sind RC5 und RC6. Der Schwerpunkt des Algorithmus liegt in der so genannten S-Box (englisch: substitution box). Die S-Box bei RC4 ist dynamisch, der Inhalt wird automatisch aus dem Schlüssel generiert. Jedes Zeichen im Klartext wird mit einem Zeichen aus der S-Box mittels XOR verknüpft. Des Weiteren verändert sich der Inhalt der S-Box permanent. Nach jeder Verknüpfung werden zwei Elemente in der S-Box vertauscht. Der RC4-Algorithmus gilt als sicher. Der Schwachpunkt liegt in der Initialisierung der S-Box. Diese ist zu Anfang immer mit aufsteigenden Zahlen von Null an belegt. Dadurch können weitergehende Untersuchungen vollzogen werden.

2.5 DES

Die Anfänge von diesem Algorithmus gehen auf Horst Feistel zurück. Dieser entwickelte im Auftrag von IBM im Jahre 1973 einen Verschlüsselungsalgorithmus mit dem Namen Lucifer. Basierend auf diesem Projekt wurde im Jahr 1977 der DES-Algorithmus als Gewinner einer öffentlichen Ausschreibung veröffentlicht. Die Abkürzung DES steht für **D**ata-**E**ncryption-**S**tandard. Der Algorithmus ist ein symmetrischer Blockchiffre mit einer Blockgröße von 64 Bit. Die Länge des Schlüssels beträgt ebenfalls 64 Bit, wobei die letzen acht Bits als Paritätskontrolle benötigt werden. Des Weiteren ist der DES-Algorithmus rundenbasierend, insgesamt werden 16 Runden durchlaufen und für jede Runde wird ein Teilschlüssel benötigt, welcher aus dem ursprünglichen Schlüssel erstellt wird. In jeder Runde wird der Eingangsblock in eine linke und in eine rechte Hälfte aufgeteilt. Die rechte Hälfte wird zum einen unverändert als linke Hälfte der nächsten Runde verwendet, zum anderen wird die F-Funktion (**F**eistel-Funktion) auf die rechte Hälfte angewendet. Hierbei wird durch eine Permutation (dt. Vertauschung) die Eingangsblockgröße verdoppelt und

das Ergebnis wird mit einem Rundenschlüssel XOR-verknüpft. Danach erfolgt eine Substitution und gleichzeitige Blockgrößenreduzierung. Im Anschluss erfolgt eine Permutation der verbliebenen 32 Bits. Nach der F-Funktion wird das Ergebnis mit der linken Hälfte per XOR verknüpft. Die Abbildung 2.2 zeigt eine Übersicht über den Algorithmus. Für eine Entschlüsselung kann der selbe Algorithmus verwendet werden, hierbei muss dieser nur in umgekehrter Richtung durchlaufen werden. Es werden nur neue Rundenschlüssel benötigt, welche durch eine zyklische Verschiebung erstellt werden.

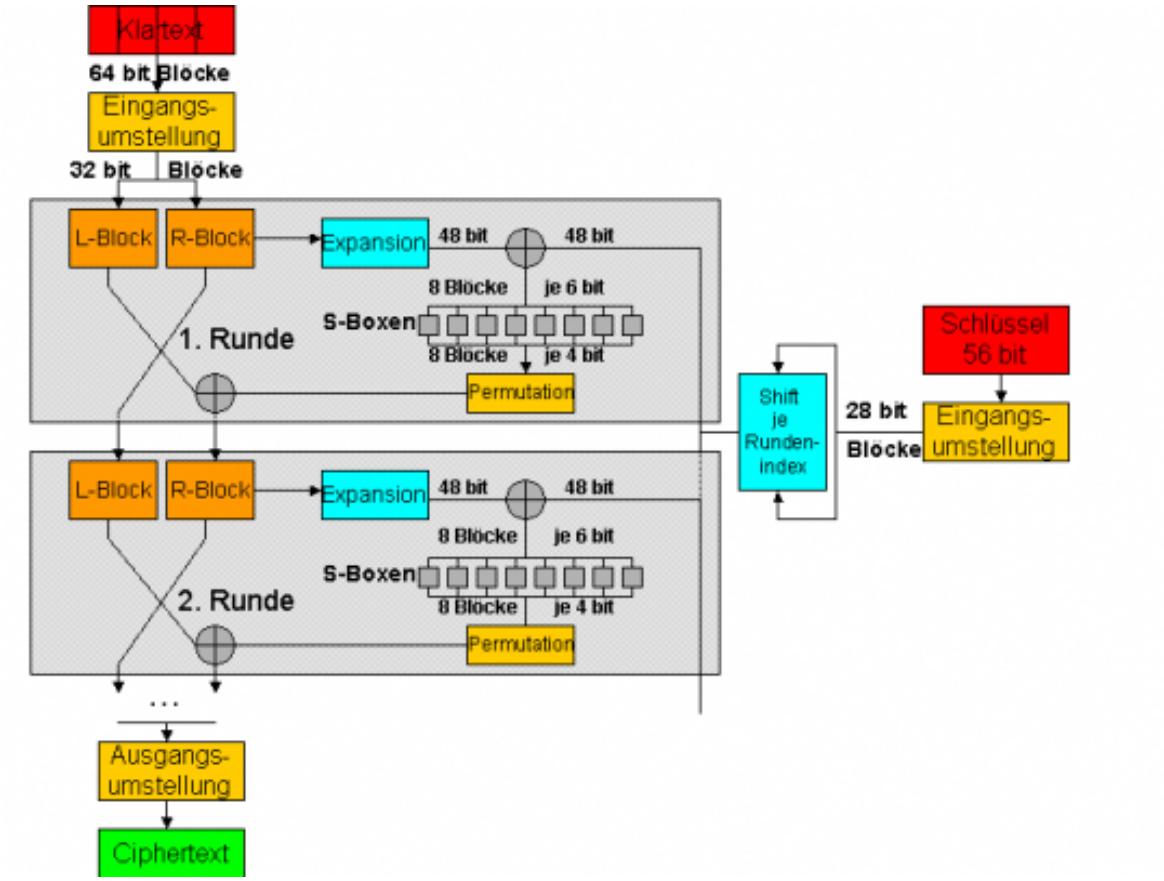

Abbildung 2.2: Ablaufdiagramm von DES, Quelle: [2].

2.5.1 Kryptanalyse gegen DES

Aufgrund des relativ kurzen Schlüssels mit 2^{56} wurde der Algorithmus schon mehrmals durch einen Brute-Force-Angriff gebrochen. Nachfolgend werden zwei Beispiele genannt. Ein Spezialrechner mit dem Namen Deep Crack wurde im Jahr 1998 von der Electronic Frontier Foundation (**EFF**) gebaut, um damit die DES Challenge II zu gewinnen. Hierbei

galt es, den Algorithmus in möglichst kurzer Zeit zu knacken [7]. Bei dem Wettbewerb wurde der Algorithmus in 56 Stunden überwunden. Bereits ein Jahr später gelang mit dem selben Spezialrechner und einem weltweiten Netzwerk aus 100000 Computern ein neuer Rekord. Hier wurde bei der DES Challenge III der Schlüssel bereits in 22,25 Stunden gefunden. Ein anderes Projekt wurde von den Universitäten in Kiel und Bochum realisiert. Hierbei entstand eine Maschine mit dem Namen COPACOBANA. Diese Maschine besteht aus 120 FPGAs vom Typ Spartan3-1000 und kann eine vollständige Schlüsselsuche in 12,8 Tagen ausführen.

2.5.2 Anwendung von DES

Obwohl der Algorithmus schon mehrfach überwunden wurde, so wird er heutzutage immer noch eingesetzt. Eine weite Verbreitung findet in der Sprachverschlüsselung von Sprechfunk statt. Hierbei ist der Algorithmus immer noch sicher, da für eine Kryptanalyse eine Echtzeitverarbeitung notwendig wäre. Des Weiteren wird der Algorithmus in Bankautomaten verwendet, um die PIN des Kunden zu verschlüsseln und diese dann zum entfernten Server zu schicken. Eine Weiterentwicklung zur Erhöhung der Sicherheit ist Triple-DES. Hierbei wird der DES-Algorithmus dreimal durchlaufen.

2.6 RSA

Der hier vorgestellte Algorithmus ist der bekannteste und populärste Algorithmus zur aysmmetrischen Kryptographie. Der Name RSA setzt sich zusammen aus den Nachnahmen der Entwickler, Ron **R**ivest, Adi **S**hamir und Leonard **A**dleman. Der Algorithmus stammt aus dem Jahr 1977, ist aber bei korrekter Verwendung immer noch ein sicheres Verfahren. Für ein besseres Verständnis wird der Algorithmus anhand von Zahlen erklärt. Die Sicherheit des Algorithmus beruht darauf, dass die Multiplikation von zwei Primzahlen leicht zu realisieren ist, das Faktorisieren allerdings, welches die Umkehrfunktion ist, kann nicht so leicht realisiert werden. Nachfolgende Auflistung erklärt das Verfahren:

- Zuerst werden zwei roße Primzahlen p und q gewählt und daraus das Produkt $n = p \cdot q$ berechnet. Im Anschluß werden zwei Schlüssel erzeugt.
- Der Chiffrierschlüssel ergibt sich aus n und einer gewählten, natürlichen Zahl e, welche teilerfremd zu n sein muss.
- Mit dem erweiterten euklidschen Algorithmus wird d berechnet, welcher der Dechiffrierschlüssel ist. Es gilt: $d = e^{-1} \mod ((p-1) \cdot (q-1))$.
- Die Primzahlen p und q werden nun nicht mehr benötigt, müssen aber geheim bleiben, da diese für die Sicherheit garantieren.

- Mit dem Klartext m und dem Geheimtext c gilt: $c = m^e \mod n$ für den Geheimtext.
- Um den Klartext wiederzuerlangen gilt $m = c^d \mod n$.

Zu erwähnen ist noch, dass die Schlüssellänge n frei wählbar ist. Je länger der Schlüssel ist, desto sicherer ist die Verschlüsselung. Gängige Schlüssellängen sind 1024 Bit und 2048 Bit [Sch09].

2.6.1 Kryptanalyse gegen RSA

Für einen asymetrischen Algorithmus gibt es viele Ansätze einer Kryptanalyse. Der einfachste Ansatz wäre eine vollständige Schlüsselsuche. Da allerdings die Schlüssellänge frei bestimmt werden kann und damit auch relativ große Schlüssel verwendet werden, wird dieses nicht zum Erfolg führen. Ein weiteres Verfahren ist ein Faktorisierungsangriff, bei dem versucht wird, die Zahl n nach p und q aufzulösen. Hierfür gibt es zwar viele Verfahren, jedoch sind diese aufwendig und rechenintensiv. Genannt seien hier die Verfahren der elliptischen Kurven, das Quadratische Sieb und das Zahlkörpersieb. Ansonsten können die Verfahren des Chosen-cipertext-Angriffs und des Public-key-only-Angriffs angewendet werden.

2.7 IDEA

IDEA ist die Abkürzung für **I**nternational **D**ata **E**ncryption **A**lgorithm. Der Algorithmus wurde 1992 in der jetzigen Form von Xuejia Lai und James Massey veröffentlicht. Die IDEA-Verschlüselung ist ein symmetrischer Blockchiffe und arbeitet mit einer Blocklänge von 64 Bit. Der Schlüssel ist auf 128 Bit festgelegt. Der Algorithmus besteht aus drei algebraischen Operationen, der XOR-Verknüpfung, der Addition modulo 2^{16} und aus der Multiplikation modulo $2^{16}+1$. Die folgende Grafik 2.3 zeigt einen schematischen Aufbau einer Runde von IDEA. Der Algorithmus besteht insgesamt aus acht Runden und einer Abschlussrunde. Das Quadrat steht für die Addition, der Kreis mit dem Kreuz entspricht der XOR-Operation und der Kreis mit dem markanten Punkt steht für die Multiplikation [3].

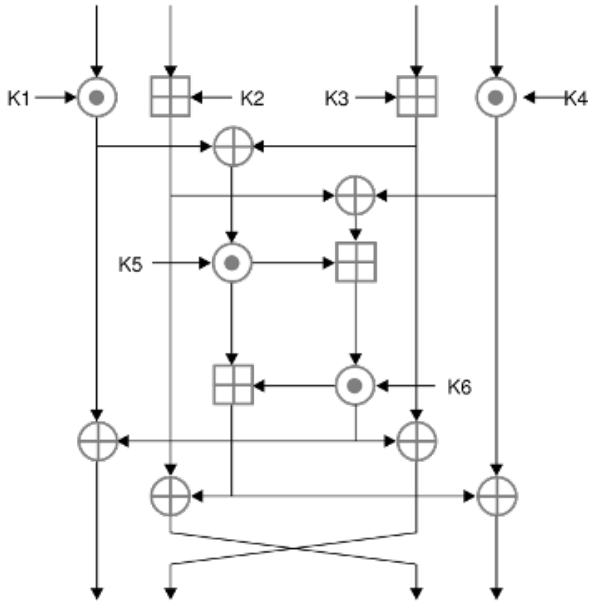

Abbildung 2.3: Ablaufdiagramm von IDEA, eine Runde, Quelle: [3].

IDEA ist in den USA und in Europa durch das Patentrecht geschützt, jedoch für eine nichtkomerzielle Nutzung frei Verfügbar. Die Popularität von IDEA entstand durch das Programm Pretty Good Privacy (PGP), welches für die Verschlüsselung und für Unterschriften von Dateien genutzt wird. Die Unterschrift einer Datei wird aus dem PGP-Schlüssel und der Datei erstellt. Ein Empfänger der Datei kann zusammen mit dem Schlüssel die Unterschrift berechnen. Sollte sich das Ergebnis beim Empfänger von der ursprünglichen Unterschrift unterscheiden, so wurde die Datei nachträglich auf dem Weg zum Empfänger manipuliert.

2.8 MD5

Die Abkürzung MD steht für **M**essage-**D**igest. Dieser Algorithmus stammt aus dem Jahr 1991 und wurde von dem amerikanischen Kryptologen und Mathematiker Ronald L. Rivest entwickelt. Der MD5-Algorithmus erstellt aus einem beliebig langen Text einen 128 Bit langen Hashwert. Mit dem dadurch erstellten ”Fingerabdruck” kann überprüft werden, ob die Nachricht verändert wurde. Sollte dieses geschehen sein, so wird aus der Nachricht ein anderer Hashwert ermittelt und die Integrität der Nachricht ist nicht mehr gewährleistet.
Bei dem Algorithmus wird die Nachricht auf eine Länge von 448 gebracht, indem zuerst eine Eins angehängt und die verbleibende Längendifferenz mit Nullen aufgefüllt wird. Anschließend wird eine 64 Bit lange Zahl angehängt, welche die Länge der ursprünglichen Nachricht angibt. Mit diesem erstellten Block werden verschiedene Runden durchlaufen, welche unterschiedliche Funktionen haben. Die Grundfunktionen sind in der Gleichung

(2.3) abgebildet, hierbei stehen $\wedge$ für AND, $\vee$ für OR, $\neg$ für NOT und $\oplus$ für XOR in der Digitaltechnik.

$$\begin{aligned} F(X,Y,Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\ G(X,Y,Z) &= (X \wedge Z) \vee (Y \wedge \neg Z) \\ H(X,Y,Z) &= X \oplus Y \oplus Z \\ I(X,Y,Z) &= Y \oplus (X \vee \neg Z) \end{aligned} \tag{2.3}$$

Der Algorithmus gilt nicht mehr als sicher, weil schon mehrfach erfolgreiche Kollisionsangriffe durchgeführt wurden. Eine Kollision bedeutet, dass zwei unterschiedliche Nachrichten den selben Hashwert erzeugen. Damit ist es möglich, passend zu einem Hashwert eine falsche Nachricht zu erstellen und diese gegen die Originalnachricht auszutauschen. Eine andere Angriffsart sind so genannte Regenbogentabellen. In diesen Tabellen sind zu unterschiedlichen Hashwerten Nachrichten abgespeichert. Anhand von so einer Tabelle kann versucht werden, den passenden Text zu einem vorhandenen Hashwert zu finden. Dieser Angriff ist besonders bei verschlüsselten Passwörtern effektiv.
MD5 ist immernoch weit verbreitet. Eine häufige Anwendung ist die Kontrolle eines Datei-Downloads. Nachdem eine Datei heruntergeladen wurde, kann ein Hashwert erstellt werden. Wenn dieser Hashwert mit dem der originalen Datei übereinstimmt, dann war der Download erfolgreich. Ansonsten ist die heruntergeladene Datei beschädigt oder verändert worden. Unter dem Betriebssystem Mac OS X gibt es das Kommando `md5` für die Kommandozeile, für Linux-betriebssysteme lautet der Befehl `md5sum`. Ein Memo von R. Rivest zum MD5-Algorithmus gibt es unter [8].

3 Entwicklung einer Kryptanalyse

3.1 Auswahl des Algorithmus

Für die Untersuchung in dieser Diplomarbeit wurde AES (**A**dvanced **E**ncryption **S**tandard) gewählt. Der Algorithmus ist zum Zeitpunkt dieser Arbeit das bedeutendste symmetrische Verschlüsselungsverfahren und ein Standard für viele Anwendungen. Neben einer hohen Sicherheit ist der Algorithmus auch leicht in Hardware und Software zu realisieren. Das Verfahren ist nicht patentiert und kann von jedem verwendet werden.
Es soll nun untersucht werden, mit welchem Hardwareaufwand ein Angriff in einer absehbaren Zeit abgeschlossen werden kann und wie sich Softwarelösungen im Vergleich dazu verhalten. Dadurch kann letztendlich auch überprüft werden, wie sicher dieser Algorithmus noch ist.

3.2 Beschreibung des Algorithmus

Der Advanced Encryption Standard ist das Ergebnis eines offenem Wettbewerbs, welcher im Jahr 1997 von dem US-amerikanischen National Institute of Standards and Technology (NIST) ausgeschrieben wurde. Das Ziel des Wettbewerbs war es, einen Nachfolgealgorithmus für DES zu finden, da dieser Algorithmus nicht mehr den Sicherheitsansprüchen der damaligen Zeit genügte. Folgende Anforderungen wurden an den neuen Algorithmus gestellt:

- Der Algorithmus soll ein symmetischer Blockchiffre sein.
- Es sollen die Schlüssellängen 128, 192 und 256 Bit verwendet werden können.
- Der Algorithmus soll leicht in Hard- und Software implementiert werden können.
- Für die Verschlüsselung soll eine Blocklänge von 128 Bit gelten.
- Der Algorithmus muss frei von Patenten sein.
- Die Verschlüsselung soll schnell zu berechnen sein.

- Der Algorithmus muss den bekannten Instrumenten der Kryptanalyse widerstehen können.
- Der Algorithmus soll ressourcensparend sein, um eine Realisierung auf Speicherkarten durchführen zu können.

Insgesamt waren 15 verschiedene Algorithmen zur Teilnahme zugelassen. Diese Algorithmen wurden öffentlich diskutiert und im April 1999 waren noch fünf Verfahren in der engeren Auswahl verblieben. Der Sieger wurde am 2. Oktober 2000 bekannt gegeben und trägt den Namen **Rijndael**. Der Rijndael-Algorithmus wurde von den beiden belgischen Entwicklern Vincent Rijmen und Joan Daemen entworfen. Ausschlaggebend für den Sieg des Wettbewerbs war die hohe Geschwindigkeit von der Hard- und Softwareimplementierung und die geringe Komplexität des Algorithmus.

3.2.1 Verschlüsselung

Der Rijndael-Algorithmus ist in vier Hauptfunktionen aufgeteilt welche jeweils mehrmals durchlaufen werden. Ein Durchlauf wird als Runde bezeichnet. Die vier Funktionen sind:

1. **SubBytes**
2. **ShiftRows**
3. **MixColumns**
4. **AddRoundKey**

Bei einer Verschlüsselung wird als Vorrunde einmal die Funktion AddRoundKey aufgerufen. Danach folgen mehrere Durchläufe in der Reihenfolge der obigen Auflistung. In der letzten Runde wird MixColumns nicht mehr verwendet. Die Anzahl der Runden ist abhängig von der Schlüssellänge und kann der folgenden Tabelle 3.1 entnommen werden. Die Tabelle gilt nur für AES, da der eigentliche Rijndael-Algorithmus für auch für andere Schlüssel- und Blocklängen entwickelt wurde. Die Vorrunde wird nicht mitgezählt.

Schlüssellänge	Anzahl der Runden
128 Bit	10
192 Bit	12
256 Bit	14

Tabelle 3.1: Anzahl der Runden bei AES.

Im Rahmen dieser Diplomarbeit wird nur die Schlüssellänge von 128 Bit betrachtet. Jede Runde wird auf eine 4x4 Matrix angewandt. Diese Matrix setzt sich aus den Bytes des Klartextes zusammen. Dadurch ergibt sich die geforderte Blocklänge von 128 Bit, da ein Byte acht Bit enthält und die Matrix aus 16 Elementen besteht. Die Bytes des Klartextes werden spaltenweise in die Matrix geschrieben. So ergibt sich aus den Bytes B eines Klartextes K die Matrix (3.1). Sobald die erste Operation des Algorithmus auf den Klartext angewendet wird, werden die Elemente mit $S_{r,c}$ angesprochen. Das S steht für State (dt. Lage, Zustand), r ist die jeweilige Zeile und c die Spalte des Elements in der Matrix. Es gilt: $0 \leq r,c \leq 3$. Diese Schreibweise wird in der Fachliteratur häufig verwendet und kommt auch in dieser Arbeit zur Anwendung.

$$\begin{bmatrix} B_{K1} & B_{K5} & B_{K9} & B_{K13} \\ B_{K2} & B_{K6} & B_{K10} & B_{K14} \\ B_{K3} & B_{K7} & B_{K11} & B_{K15} \\ B_{K4} & B_{K8} & B_{K12} & B_{K16} \end{bmatrix} \text{ eine andere Schreibweise: } \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} \tag{3.1}$$

SubBytes

Diese Funktion ersetzt jedes Byte des Klartextes durch ein Byte aus einer statischen S-Box. Die Funktion ist eine nichtlineare Substitution. Das Ziel ist es, den Zusammenhang zwischen den Klartextzeichen und den Zeichen des Chiffrats zu verschleiern. Diese Vorgehensweise wird in der Kryptologie Konfusion genannt. In der Tabelle sind die Werte Null bis 255 enthalten. Diese Werte werden einmal berechnet und können dann für die Verschlüsselung verwendet werden. Für die Berechnung wird der endliche Körper GF(2^8) benötigt (Galois-Feld (2^8)). In diesem Feld können keine Zahlen größer als 255 entstehen, da die Rechenoperationen so definiert sind, dass sich die Ergebnisse immer in einem Byte abbilden lassen. Für einen Wert T in der Ersetzungstabelle und einem Byte b aus der Blockmatrix gilt folgende Gleichung (3.2):

$$T = A \cdot b^{-1} \oplus 01100011_b \tag{3.2}$$

Für A gilt die Matrix (3.3):

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{3.3}$$

Für b^{-1} ist zu beachten, dass das inverse Element im Galois Feld gesucht ist (3.5). Es sei also eine Zahl x gesucht, welche bei einer Multiplikation mit b und Modulo 283 eine Eins ergibt (3.6). Die Zahl 283 ist in dem Standard festgelegt, dahinter verbirgt sich das folgende irreproduzible Polynom:

$$m(x) = x^8 + x^4 + x^3 + x + 1 \tag{3.4}$$

Dieses Polynom wird so in einem Byte abgbildet, dass nur die Koeffizienten Null oder Eins gespeichert werden. Dadurch wird (3.4) binär gespeichert als 100011011_b. Das entspricht der Zahl 283 im Dezimalsystem. Eine Multiplikation bei AES ist immer eine Multiplikation zweier Zahlen modulo 283 und wird durch das Zeichen $\bullet$ als solche gekennzeichnet. Zu beachten ist außerdem, dass bei der Modulo-Operation die binäre Polynomdivision angewendet wird.

Für das inverse Element gilt folgendes:

$$b \bullet b^{-1} = 1 \tag{3.5}$$

$$b \cdot x = 1 \pmod{283} \tag{3.6}$$

$$x \equiv b^{-1} \tag{3.7}$$

Das inverse Element von Null ist definiert als Null. Die komplette S-Box ist im Anhang B.1 zu finden. Der Austausch erfolgt dadurch, dass bei einem gegebenen Byte das obere Nibble die Zeile und das untere Nibble die Spalte in der Tabelle angibt. Zusammen referenzieren Zeile und Spalte auf den neuen Wert.

ShiftRows

Bei dieser Funktion werden die Zeilen der Blockmatrix verschoben. Bei einer Schlüssellänge von 128 Bit wird die zweite Zeile um eins nach links rotiert, so dass das Byte, welches links herausfällt, auf der rechten Seite wieder in die Matrix reingeschoben wird. Analog dazu wird die dritte Zeile um zwei nach links geschoben und die vierte Zeile wiederum um drei nach links. Bei anderen Schlüssellängen sind die Verschiebungen geringfügig anders. Im Rahmen dieser Arbeit werden andere Schlüssellängen jedoch nicht betrachtet. Das Ziel ist die Beseitigung von statistischen Strukturen, welche in der Kryptologie Diffusion genannt werden. Eine statistische Struktur ist zum Beispiel die Buchstabenhäufigkeit, wie sie bei der Caesar-Verschlüsselung vorkommt.

MixColumns

Diese Funktion dient dazu, die Spalten zu vermischen. Das Ziel ist hier wieder eine Diffusion des Textes. Hierfür wird jede Spalte als Polynom dritten Grades betrachtet, wobei jeder Koeffizient ein Element aus GF(2^8) ist. Jede Spalte wird mit einem festen Polynom $a(x)$ (3.8) multipliziert. Damit das Ergebnis ebenfalls ein Polynom dritten Grades ist, wird für eine Modulorechnung das Polynom $m(x) = x^4 + 1$ verwendet.

$$a(x) = 3x^3 + 1x^2 + 1x + 2 \tag{3.8}$$

Für eine Spalte c lässt sich diese Multiplikation in Matrixform darstellen (3.9). Die Matrixelemente sind hexadezimal abgebildet.

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \tag{3.9}$$

Für jedes Element $S_{r,c}$ ergibt sich eine Gleichung, bei der die Elemente jeweils aus dem GF(2^8) sind.

AddRoundKey

Bei dem letzten Schritt einer Runde und bei der Vorrunde wird ein Rundenschlüssel mit den aktuellen Elementen in dem zu bearbeitenden Block mit XOR verknüpft. Der Rundenschlüssel wird vorher erzeugt und besteht aus einer 4x4 Matrix mit insgesamt 16 Byte.

3.2.2 Erzeugung des Rundenschlüssels

Bei einer Verschlüsselung mit 10 Runden und einem 128 Bit langem Schlüssel werden insgesamt 11 Teilschlüssel benötigt, da zusätzlich zu den Runden ein Teilschlüssel bei der Vorrunde benötigt wird. Diese Teilschlüssel, oder auch Rundenschlüssel genannt, haben die Form einer 4x4 Matrix mit insgesamt 16 Byte. Bei einem Schlüssel der Länge 128 Bit wird insgesamt ein Schlüssel der Länge 176 Byte verwendet.
Der gegebene 128 Bit lange Schlüssel wird in vier gleichgroße Teile zerlegt. Der erste Rundenschlüssel ergibt sich aus diesen Byte, welche zeilenweise in die Matrix w geschrieben werden. Jede Zeile ist fortlaufend nummeriert mit w_i und es gilt: $0 \leq i \leq 43$. Der erste Rundenschlüssel der Vorrunde ist somit $w_{0:3}$. Um die Zeile w_4 zu generieren, werden nacheinander die Funktionen RotWord und SubWord mit dem Wert von w_3 ausgeführt. Beide Funktionen sind weiter unten genauer beschrieben. Das Ergebnis wird mit dem konstanten Wert aus der Matrix Rcon mittels XOR verknüpft. Nach dieser Operation wird das Resultat mit der Zeile w_0 ein letztes Mal mit XOR verknüpft. Die folgenden drei Zeilen der Matrix sind XOR-Verknüpfungen aus der jeweiligen vorherigen Zeile und der Zeile, welche vier Indexzähler zurück liegt. Diese Vorgehensweise wiederholt sich zyklisch für jede Runde.
Zu beachten ist, dass die einzelnen Teilschlüssel zwar zeilenweise generiert werden, in der Blockmatrix werden die Eingangsbytes aber spaltenweise abgelegt. Somit muss der Rundenschlüssel transponiert werden, da in der Funktion AddRoundKey sonst die falschen Elemente miteinander verbunden werden. Die Abbildung 3.1 zeigt die Schlüsselerzeugung. Das Schaubild ist angelehnt an die Abbildung der Schlüsselerzeugung in [Sch09], wurde jedoch geringfügig verändert.

RotWord

Diese Funktion rotiert ein Byte einer Zeile der Matrix w_i nach links.

$$w_i = \begin{bmatrix} B_0 & B_1 & B_2 & B_3 \end{bmatrix} \quad \Rightarrow \quad w_i = \begin{bmatrix} B_1 & B_2 & B_3 & B_0 \end{bmatrix} \tag{3.10}$$

SubWord

Bei dieser Funktion wird jedes Byte durch ein neues Byte aus der S-Box (B.1) ausgetauscht. Bei einem Byte steht das obere Nibble für die Zeile und das untere Nibble für die Spalte in der S-Box. Dadurch lässt sich der neue Wert für das Byte in der S-Box finden.

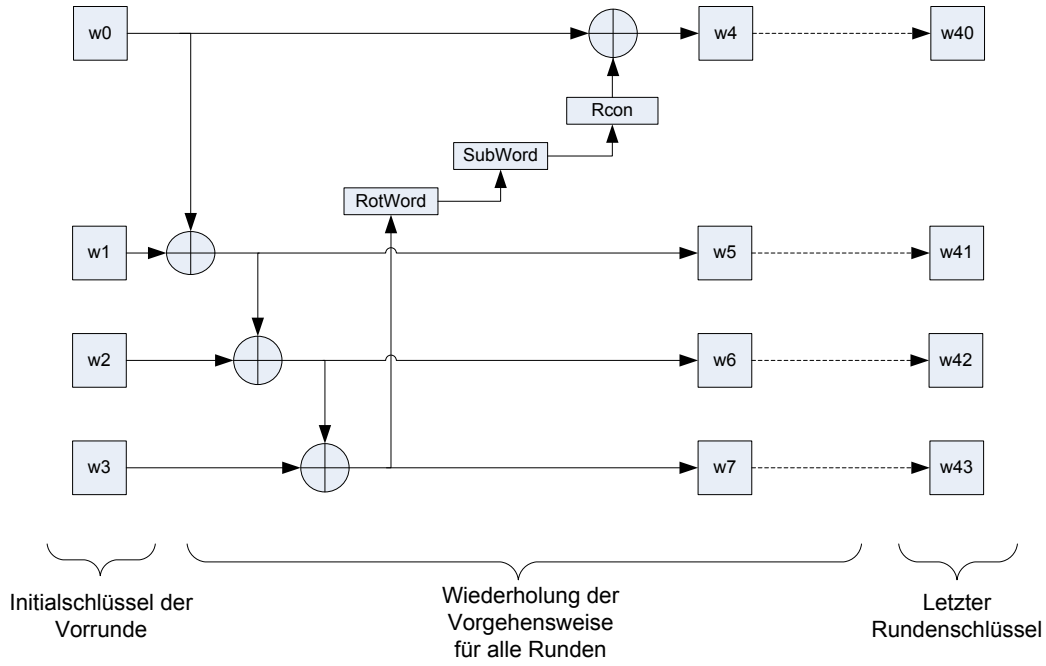

Abbildung 3.1: Erzeugung des Rundenschlüssels

Rcon

Rcon ist eine konstante Matrix der Größe 10x4 (3.11). Für jede Runde gibt es eine Zeile mit vier Bytes. Mit einer bitweisen XOR-Operation wird eine Zeile der Schlüsselmatrix w mit einer Zeile aus der Rcon-Matrix verknüpft. Hierbei entspricht die jeweilige Runde der Zeile in der Matrix. Die Werte in der Matrix sind hexadezimal.

$$Rcon = \begin{bmatrix} 01 & 00 & 00 & 00 \\ 02 & 00 & 00 & 00 \\ 04 & 00 & 00 & 00 \\ 08 & 00 & 00 & 00 \\ 10 & 00 & 00 & 00 \\ 20 & 00 & 00 & 00 \\ 40 & 00 & 00 & 00 \\ 80 & 00 & 00 & 00 \\ 1B & 00 & 00 & 00 \\ 36 & 00 & 00 & 00 \end{bmatrix} \tag{3.11}$$

3.2.3 Entschlüsselung

Die Entschlüsselung eines Geheimtextes erfolgt in umgekehrter Reihenfolge der Verschlüsselung. Des Weiteren müssen kleine Änderungen vorgenommen werden, da einzelne Operationen rückwärts durchgeführt werden müssen. Der allgemeine Ablauf pro Runde ist in der nachfolgenden Auflistung dargestellt.

1. **InvShiftRows**
2. **InvSubBytes**
3. **AddRoundKey**
4. **InvMixColumns**

Als Vorrunde wird einmal die Funktion AddRoundKey angewendet und beim letzten Rundendurchlauf wird die Funktion InvMixColumns nicht verwendet.

InvShiftRows

Diese Funktion schiebt die Zeilen einer 4x4 Matrix nach rechts. Die erste Matrixzeile bleibt unverändert. Die zweite Zeile wird um ein Element nach rechts geschoben. Das letzte Element der Zeile wird am Zeilenanfang wieder eingefügt. Die dritte Zeile wird um zwei Elemente verschoben und die letzte Zeile um drei Elemente.

InvSubBytes

Bei der inversen S-Box werden die Zahlen Null bis 255 in eine neue Matrix eingetragen. Die Werte der ursprünglichen S-Box sind die Feldindices der neuen, inversen S-Box. Ein Beispiel: Der Wert 63_h steht bei der S-Box an der Stelle 0_h. Dadurch ist bei der inversen S-Box an der Stelle 63_h der Wert 0_h. Für ein einfaches Verständnis ist der folgende Pseudocode (3.1) gegeben:

```
1 for i=0 to 255 do
      inv_S_box( s_box[i] ) = i
3 end for
```

Listing 3.1: Pseudocode zum Generieren der inversen S-Box

In der folgenden Tabelle 3.2 ist die inverse S-Box abgebildet. Der Austausch der Werte erfolgt dadurch, dass das obere Nibble des alten Wertes die Zeile und das untere Nibble die Spalte des neuen Wertes angibt.

oberes Nibble \ unteres Nibble	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Tabelle 3.2: Die inverse S-Box von AES. Alle Zahlen sind hexadezimal

AddRoundkey

Bei dieser Funktion wird nichts weiter verändert, da die XOR-Operation zweimal angewandt die Inversion ergibt. Zu beachten ist aber, dass alle Rundenschlüssel vor der Entschlüsselung erzeugt werden müssen, da diese in umgekehrter Reihenfolge verwendet werden müssen. Somit wird bei der Vorrunde der Rundenschlüssel $w_{40:43}$ verwendet.

InvMixColumns

Die Funktion ist das Inverse der MixColumns-Funktion. Bei dieser Funktion wird jede Spalte mit a^{-1} (3.12) multipliziert und mit dem Polynom $m(x) = x^4 + 1$ Modulo gerechnet. Die Koeffizienten aus dem Galois-Feld GF(2^8) sind hexadezimal abgebildet.

Für eine Spalte c lässt sich diese Multiplikation in Matrixform darstellen (3.13). Die Matrixelemente sind ebenfalls hexadezimal abgebildet.

$$a^{-1}(x) = 0Bx^3 + 0Dx^2 + 09x + 0E \tag{3.12}$$

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \cdot \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \tag{3.13}$$

3.2.4 Betriebsarten

Für jeden Blockalgorithmus gibt es verschiedene Betriebsarten von unterschiedlicher Sicherheit. Der Hauptgrund für die Notwendigkeit von mehreren Betriebsarten liegt darin, dass bei einem Blockalgorithmus eine Nachricht in mehrere Blöcke aufgespaltet wird. Blöcke werden benötigt, weil eine Nachricht im Normalfall zu lang ist, um den Algorithmus darauf anzuwenden. Nachfolgend wird nur eine kleine Auswahl an Möglichkeiten beschrieben, wie eine Nachricht auf mehrere Blöcke verteilt werden kann.

Electronic codebook (ECB)

Die ECB-Methode ist die einfachste Betriebsart eines Blockalgorithmus. Hierbei wird der Klartext vom Anfang an in die benötigte Blockgröße aufgespaltet und nacheinander verschlüsselt. Der große Nachteil an dieser Methode ist, dass identische Blöcke vom Klartext auch in identischen Geheimtexten resultieren. Diese Betriebsart wird aber im Rahmen dieser Diplomarbeit verwendet, weil nur ein Block verschlüsselt wird. Weitere Sicherheitsmaßnahmen werden nicht berücksichtig, da im Anschluss ein Geschwindigkeitsvergleich erfolgen soll und dafür die simpelste Betriebsart ausreichend ist.

Cipher-block chaining (CBC)

Diese Betriebsart wurde 1976 von IBM entwickelt. Hierbei wird jeder Klartextblock vor der Verschlüsselung mit dem vorherigen, schon verschlüsselten Block mit XOR verknüpft. Dadurch ist jeder Block vom vorherigen Block abhängig, gleiche Klartextblöcke resultieren aber nicht mehr in gleichen Blöcken von Geheimtext. Für den ersten Block wird ein Initialisierungsvektor (IV) benötigt.

Output feedback (OFB)

Bei dieser Methode wird der Blockalgorithmus in einen Stromalgorithmus umgewandelt. Als erster Block zur Verschlüsselung wird ein Initialisierungsvektor benötigt. Nach der Verschlüsselung wird das Ergebnis zum einen als Eingabeblock für die nächste Verschlüsselung benutzt, zum anderen wird damit ein Klartextblock gleicher Größe mit XOR verknüpft, um damit den Geheimtext zu erstellen. Besonders hierbei ist, dass bei der Entschlüsselung ebenfalls eine Verschlüsselung durchgeführt wird, jedoch wird der Geheimtext mit den jeweiligen Blöcken verknüpft, um damit dann den Klartext zu erstellen.

Cipher feedback (CFB)

Diese Betriebsart ist sehr ähnlich zu der vorherigen. Auch hierbei entsteht ein Stromchiffre. Bei der Verschlüsselung wird für den ersten Block ein Initialisierungsvektor benötigt. Nach der Verschlüsselung wird der Klartext mit dem Ergebnis per XOR-Operator verknüpft. Das Ergebnis ist zum einen der Geheimtext, zum anderen der Eingangsblock für die weitere Verschlüsselung. Bei der Entschlüsselung wird ebenfalls eine Verschlüsselung verwendet. Für den ersten Block wird wieder der Initialisierungsvektor benötigt. Um den Klartext zu erstellen, wird das Ergebnis dann mit XOR und dem Geheimtext berechnet. Der erste Geheimtextblock ist ebenfalls der Eingangsvektor für den zweiten Block der Entschlüsselung.

3.3 Kryptanalyse

Eine komplett erfolgreich durchgeführte Kryptanalyse gegen AES ist zu diesem Zeitpunkt nicht bekannt. Kritikpunkte an dem Algorithmus sind zum Beispiel die relativ geringe Rundenzahl und die einfach gehaltene S-Box. Es gibt eine Vielzahl von Ideen für einen erfolgreichen Angriff. Nachfolgend werden einige Beispiele genannt. Des Weiteren konnten schon erfolgreiche Angriffe mit einer reduzierten Rundenzahl durchgeführt werden.

3.3.1 Rijndael als Formel

Eine Möglichkeit ist es, den kompletten Algorithmus als eine mathematische Formel auszudrücken. Diese Formel enthält 2^{50} Komponenten und konnte bisher noch nicht nach dem Schlüssel aufgelöst werden. Es ist aber vorstellbar, dass mit zunehmender Rechenleistung

oder durch neue Erkenntnisse, welche diese Gleichung vereinfachen, dieser Angriff möglich und sogar erfolgreich sein könnte. Als kleines Beispiel ist nachfolgend die Gleichung für eine Runde gegeben, welche aus der Quelle [FSW01] entnommen wurde.

$$a_{i,j}^{r+1} = k_{i,j}^{(r)} + \sum_{\substack{e_r \in \mathcal{E}, \\ d_r \in D}} \frac{w_{i,e_r,d_r}}{(a_{e_r,e_r+j}^{(r)})^{2d_r}} \tag{3.14}$$

3.3.2 XSL

Eine andere Methode ist E**X**tended **S**parse **L**inearization (XSL). Das Verfahren wurde von Nicolas Courtois und Josef Pieprzyk weiterentwickelt, es beruht auf der heuristischen Methode e**X**tended **L**inearization (XL). Mit XL ist es möglich, große nichtlineare Gleichungssysteme zu lösen. In der Praxis scheiterte dieses Prinzip. Bei einem Angriff mit XSL kann der AES Algorithmus durch eine Vielzahl von quadratischen Gleichungen beschrieben werden. Der Rechenaufwand liegt jedoch bei 2^{200} Operationen für ein Gleichungssystem mit 8000 quadratischen Gleichungen und 1600 Variablen. Durch diesen hohen Aufwand wurde diese Methode noch nicht realisiert und es ist zum heutigen Zeitpunkt fraglich, ob ein Angriff überhaupt realisierbar ist. Interessant ist, dass eine Erhöhung der Rundenzahl bei AES keine exponentielle Rechenzeitsteigerung bei XSL erzwingt [CP].

3.4 Entwicklung des Angriffs gegen AES

Im Rahmen dieser Diplomarbeit wird eine komplette Schlüsselsuche als Angriff verwendet. Anhand dessen kann ein Vergleich zwischen der Realisierung auf einem FPGA und auf einem Computer erreicht werden. Der Entwurf des Angriffs wird so ausgelegt, dass das Ergebnis in einer relativ kurzen Zeit vorliegt. Das bedeutet, dass möglichst viele Operationen parallel abgearbeitet werden. Dadurch entsteht ein hoher Hardwareaufwand, da für eine parallele Umsetzung einzelne Komponenten mehrmals vorhanden sein müssen.

3.4.1 Übersicht über die Schaltung

Für eine vollständige Schlüsselsuche wird ein Zähler benötigt, welcher alle möglichen Schlüssel generiert. Mit allen Schlüsseln wird ein schon vorab verschlüsselter Text entschlüsselt. Wenn der entstehende Klartext manuell kontrolliert wird, so kann von einer Person eine Entscheidung getroffen werden, ob das Ergebnis der Entschlüsselung korrekt ist. Dieses funktioniert aber nur, wenn der entstandene Klartext für einen Menschen lesbar ist, oder wenn die kontrollierende Person ein bestimmtes Ergebnis erwartet. Der Vorteil einer manuellen Ergebniskontrolle ist, dass keine zusätzliche Berechnung für eine Auswertung

benötigt wird. Ferner kann ein Angriff sofort beendet werden, wenn das Ergebnis zufrieden stellend ist.
Eine manuelle Ergebniskontrolle hat zwei große Nachteile. Zum Ersten ist es anstrengend, eine Vielzahl von Ergebnissen manuell zu kontrollieren, wodurch sich auch die Wahrscheinlichkeit für Fehler erhöht. Zum Zweiten ist diese Art von Kontrolle sehr zeitintensiv, was dem Hauptkriterium einer schnellen Ergebnisfindung hinderlich ist. Aus den genannten Gründen wird am Ende der Ergebnistext analysiert und ausgewertet.

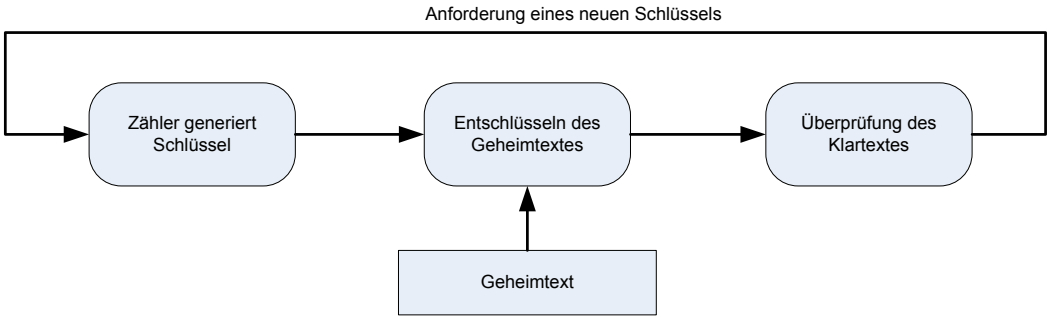

Abbildung 3.2: Übersicht über den Angriff

3.4.2 Realisierung der benötigten Komponenten

Da eine komplette Schlüsselsuche im Prinzip einer normalen Entschlüsselung entspricht, müssen die Komponenten realisiert werden, wie sie in 3.2.3 aufgeführt sind. Nachfolgend ist ein Ablaufplan für eine komplette Schlüsselsuche abgebildet. Der Plan zeigt nur eine beispielhafte Vorgehensweise, um den generellen Ablauf zu veranschaulichen.

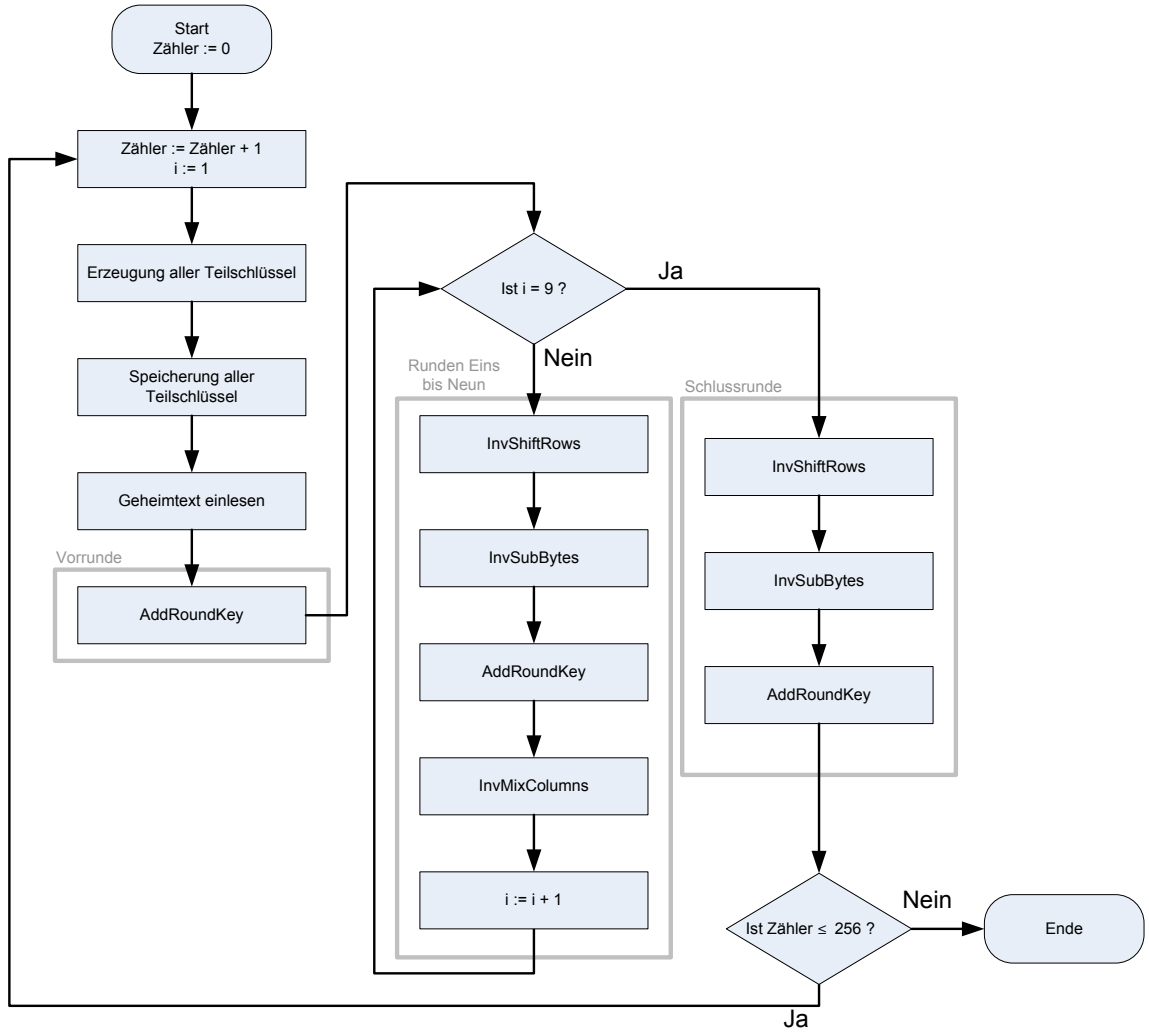

Abbildung 3.3: Ablaufplan einer kompletten Schlüsselsuche in Software.

InvShiftRows

Bei dieser Funktion erfolgt auf der Hardwareebene nur eine andere Verdrahtung der Eingangsbits auf den Ausgang. Anhand der folgenden Grafik 3.4 wird das Verfahren gut dargestellt. Die 32-Bit Leitungen entsprechen den Zeilen der Zustandsmatrix des Algorithmus. Deutlich zu erkennen ist, dass die erste Zeile nicht verändert wird. In der zweiten Zeile wird ein Byte nach rechts rotiert. Kenntlich gemacht wird dies, indem die 32-Bit Leitung in die einzelnen Bytes aufgeteilt wird, was an der Beschriftung zu erkennen ist. Die übrigen Matrixzeilen werden entsprechend der obigen Beschreibung ebenfalls rotiert.

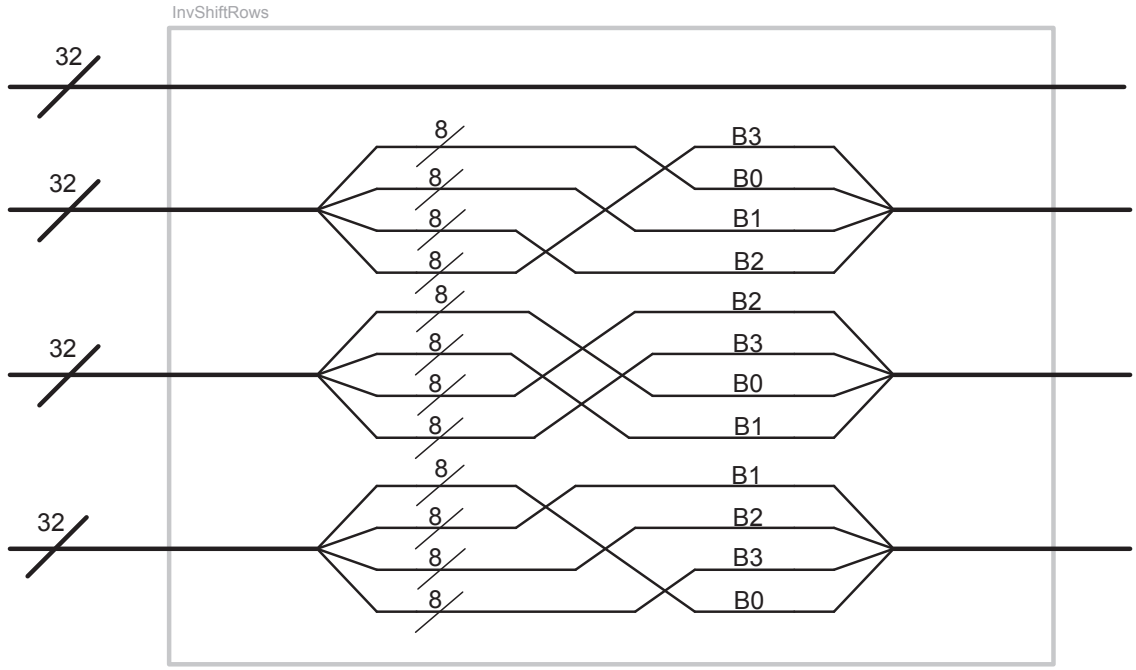

Abbildung 3.4: Blockschaltbild der Funktion InvShiftRows.

InvSubBytes

Die Realisierung dieser Funktion ist einfach. Wie schon vorher beschrieben, wird nur eine Zahl gegen eine andere ausgetauscht. Die inverse S-Box (3.2) wird als ROM (**R**ead **O**nly **M**emory) angelegt. Die Adresse des Speichers ist identisch mit der ursprünglichen Zahl, der Wert im Speicher entspricht dem Wert aus der Tabelle.

AddRoundkey

Da diese Funktion im Algorithmus nur eine byteweise XOR-Operation ist, ist dieses in der Hardware relativ leicht zu realisieren. Die einzelnen Bytes werden mit einem XOR-Gatter miteinander verbunden.

InvMixColumns

Die Realisierung dieser Funktion ist etwas schwieriger als die der anderen Funktionen. Hier muss die Matrixmultiplikation der Gleichung (3.13) umgesetzt werden. Nachfolgend ist eine Beispielrechnung angegeben (3.15). Die Koeffizienten sind in hexadezimaler Schreibweise dargestellt:

$$S'_{0,0} = 0E \cdot S_{0,0} + 0B \cdot S_{1,0} + 0D \cdot S_{2,0} + 09 \cdot S_{3,0} \tag{3.15}$$

$$S'_{0,0} = 0E \bullet S_{0,0} \oplus 0B \bullet S_{1,0} \oplus 0D \bullet S_{2,0} \oplus 09 \bullet S_{3,0} \tag{3.16}$$

Für eine schnelle, einfache und effiziente Realisierung der Berechnung werden die jeweiligen Multiplikationen zuerst einzeln durchgeführt und im Anschluss wird das Gesamtergebnis S' aus den Teilergebnissen berechnet. Die einzelnen Multiplikationen werden als Kombinationen aus Multiplikationen um den Faktor Zwei und dessen Vielfache betrachtet. Eine Multiplikation um den Faktor Zwei bedeutet auf Bitebene eine Verschiebung der Zahl um eine Stelle nach links. Diese Rechnung ist sehr schnell und einfach durchzuführen. Bei einer Multiplikation mit Vier wird eine Zahl zweimal um eine Stelle nach links geschoben. Zu beachten ist, dass bei einer Eins als MSB das Ergebnis mit der Zahl $1B_h$ mit XOR verknüpft werden muss. Die Zahl $1B_h$ ist eine verkürzte Form der Gleichung (3.4), welche für die Modulorechnung um Galois-Feld verwendet wird. Die eigentliche Repräsentation des Polynoms in hexadezimaler Schreibweise ist $11B_h$. Die führende Eins kann aber vernachlässigt werden, da diese bei der Modulorechnung zusammen mit der geschobenen Eins automatisch verschwindet, sowohl durch die XOR-Berechnung, als auch durch die Schiebeoperation. Sichergestellt ist dadurch aber, dass das Ergebnis korrekt im GF(2^8) berechnet wird. Deswegen muss eine Multiplikation mit Vier als Multiplikation mit $2 \cdot 2$ berechnet werden, da auf die Zwischenergebnisse nicht verzichtet werden kann. Jeder Koeffizient in der Matrix (3.13) lässt sich zerlegen in Vielfache der Zahl Zwei und gegebenenfalls noch eine Addition mit Eins. Addiert wird mit XOR, da dieses der Addition im GF(2^8) entspricht. Die folgende Tabelle 3.4.2 zeigt die Aufteilung der Werte aus der Matrix.

Multiplikation hexadezimal	Multiplikation binär	Multiplikation mit Vielfachen
$x \cdot 0E$	$x \cdot 00001110$	$x \cdot 8 \oplus x \cdot 4 \oplus x \cdot 2$
$x \cdot 0B$	$x \cdot 00001011$	$x \cdot 8 \oplus x \cdot 2 \oplus x \cdot 1$
$x \cdot 0D$	$x \cdot 00001101$	$x \cdot 8 \oplus x \cdot 4 \oplus x \cdot 1$
$x \cdot 09$	$x \cdot 00001001$	$x \cdot 8 \oplus x \cdot 1$

Tabelle 3.3: Zerlegung der Matrixwerte für InvMixColumns

In der digitalen Hardware wurde dazu ein Multiplizierer realisiert, welcher durch eine Schiebeoperation die jeweiligen Vielfachen erzeugt und gegebenenfalls die Modulo-Operation durchführt. Durch Angabe eines Operationscodes kann dann bestimmt werden, mit welchem Wert aus der Matrix die Zahl x multipliziert werden soll. Da diese Funktion die kritischste Funktion im Bezug auf die Laufzeit und Komplexität des Algorithmus ist, wird der Quellcode mit abgedruckt. Dieser befindet sich im Anhang C.

Schüsselerzeugung

Bei der üblichen Entschlüsselung werden alle Teilschlüssel vor der Benutzung erzeugt und gespeichert. Danach werden die Teilschlüssel in umgekehrter Reihenfolge verwendet, so wie es der Standard vorsieht. Um eine Speicherung der Teilschlüssel zu umgehen und dadurch Speicherplatz zu sparen ist es besser, wenn der jeweilige Teilschlüssel zeitnah vor der Verwendung generiert wird. Nach der Benutzung kann dieser Schlüssel dann verworfen werden und der Speicher wird somit freigegeben. Um das zu erreichen wird die oben beschriebene Vorgehensweise (3.2.3) etwas angepasst.
Die ersten drei Zeilen eines Teilschlüsselblocks entstehen aus der XOR-Verknüpfung der jeweils dritten und vierten vorherigen Zeile. Jede vierte Zeile wird dadurch generiert, dass auf dem neu generierten, ersten Schlüssel im Block nacheinander die Funktionen invRotWord, SubWord und Rcon angewendet werden. Bei Rcon ist zu beachten, dass die konstante Matrix mit der letzten Zeile zuerst verwendet wird. Das Ergebnis wird mit der vierten vorherigen Zeile mit XOR verknüpft und es entsteht der letzte Teilschlüssel im aktuellen Block. Die Grafik 3.5 zeigt den Ablauf der Schlüsselerzeugung.

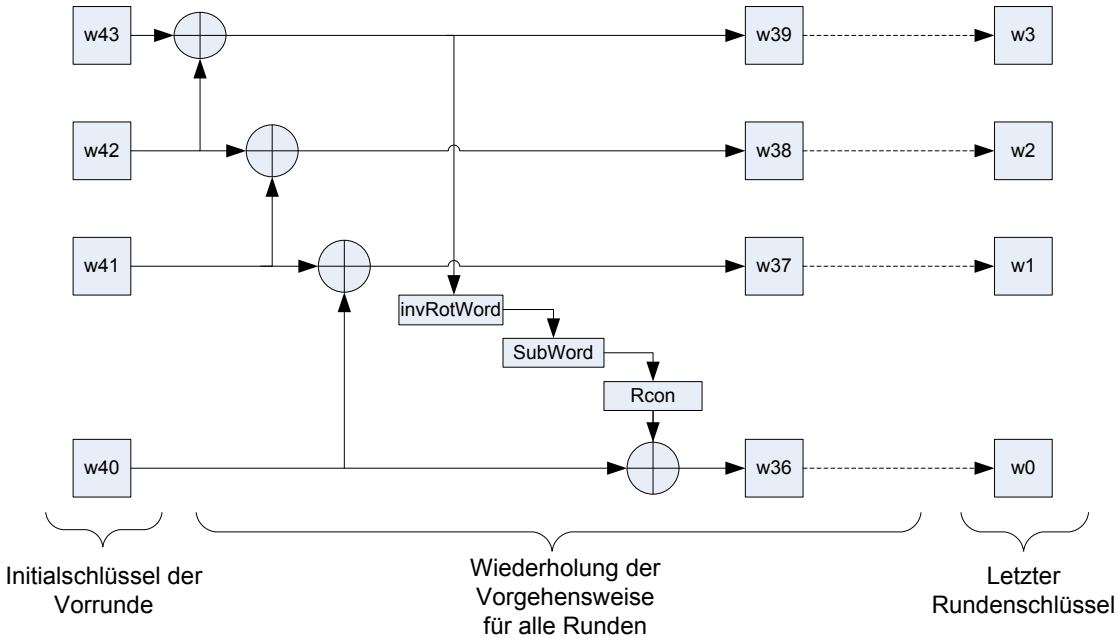

Abbildung 3.5: Schaubild zur Darstellung der Teilschlüsselerzeugung.

3.4.3 Simulationen der benötigten Komponenten

Nachdem alle benötigten Komponenten in der Hardwarebeschreibungssprache VHDL realisiert worden sind, muss überprüft werden, ob das Verhalten der Komponenten korrekt ist. Für sämtliche Simulationen wird das Programm ModelSim XE III in der Version 6 von der Firma Xilinx eingesetzt. Um zu prüfen, ob die Ergebnisse korrekt sind, werden diese mit Ergebnissen aus der AES Toolbox für MATLAB [9] verglichen. Der MATLAB-Quellcode wurde zur Erklärung des Algorithmus geschrieben und beinhaltet eine gute Dokumentation [Buc01]. Die Funktion der einfachen Komponenten wie AddRoundKey und InvShiftRows kann leicht nachvollzogen werden und soll hier nicht weiter erläutert werden. Nachfolgend werden die Ergebnisse der Überprüfungen von den übrigen Komponenten gezeigt.

Überprüfung von SubBytes und InvSubBytes

Die beiden Funktionen zur Bytesubstitution sind in ihrer Funktion zwar einfach nachzuvollziehen, jedoch können durch die Größe der Tabelle leicht Fehler entstehen. Um beide Funktionen schnell überprüfen zu können, werden sie hintereinander geschaltet. Dadurch wird der Eingangswert zweimal getauscht, jeweils einmal aus einer der Tabellen. Dadurch muss der Ausgangswert aus der zweiten Tabelle identisch mit dem Eingangswert der ersten Tabelle sein. Für den Eingang wird ein Zähler verwendet, welcher die Zahlen von Null

bis 255 hochzählt. Wie in der Abbildung 3.6 zu sehen, sind die Signale `counter` und `outbyte` identisch. Das Signal `tmp_out` ist der Ausgang von SubBytes und gleichzeitig der Eingang von InvSubBytes.

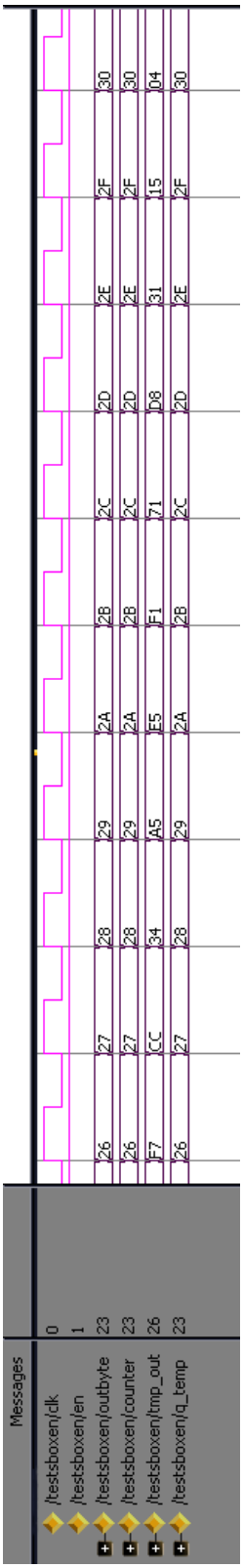

Abbildung 3.6: Simulation von SubBytes und InvSubBytes.

Überprüfung von InvMixColumns

Um die Funktion von InvMixColumns zu überprüfen wird hier ebenfalls die Komponente simuliert und mit Hilfe von MATLAB manuell kontrolliert. Die Grafik 3.7 zeigt ein mögliches Simulationsergebnis. Nachfolgende Gleichung (3.17) zeigt eine Beispielrechnung zum Verständnis. Alle Werte sind in hexadezimaler Schreibweise angegeben. Die Funktion *xtime* bewirkt eine Multiplikation mit Zwei, welche durch eine Schiebeoperation nach links ausgeführt wird. Auf das Resultat wird gegebenenfalls die XOR-Operation mit $1B$ angewendet, sofern das MSB des Eingangswertes eine Eins ist [Pub].

$$
\begin{aligned}
&\text{out\_0\_0} = 54 = && 49 \oplus FB \oplus 2B \oplus CD \\
&\text{Berechnung für 49:} && \\
&\text{in\_0\_0} \bullet 0E = && E9 \bullet 0E = E9 \bullet (02 \oplus 04 \oplus 08) \\
&\text{mit:} && \\
& && E9 \bullet 02 = xtime(E9) = C9 \\
& && E9 \bullet 04 = xtime(C9) = 89 \\
& && E9 \bullet 08 = xtime(89) = 9 \\
&\text{folgt:} && \\
& && E9 \bullet 0E = C9 \oplus 89 \oplus 9 = 49
\end{aligned}
\tag{3.17}
$$

Überprüfung der Schlüsselerzeugung

Die folgende Abbildung 3.8 zeigt eine Simulation der Schlüsselerzeugung. Nachfolgend wird eine Beispielrechnung (3.18) angegegeben, um zu zeigen, dass der Algorithmus funktioniert. Des Weiteren kann so das Schaubild zur Teilschlüsselerzeugung (3.5) besser nachvollzogen werden. Alle Werte sind hexadezimal. Zu beachten ist, dass die Teilschlüssel spaltenweise abgelegt sind. Die Eingangswerte in der Grafik sind die Teilschlüssel w40 bis w39. Die Schlüssel w37 bis w39 ergeben sich aus der XOR-Verknüpfung mit den jeweils drei bzw. vier vorherigen Schlüsseln. Auf den Schlüssel w39 werden nacheinander die Funktionen invRotWord, SubWord und Rcon angewendet. Das daraus resultierende Ergebnis wird mit dem Teilschlüssel w40 durch XOR verknüpft.

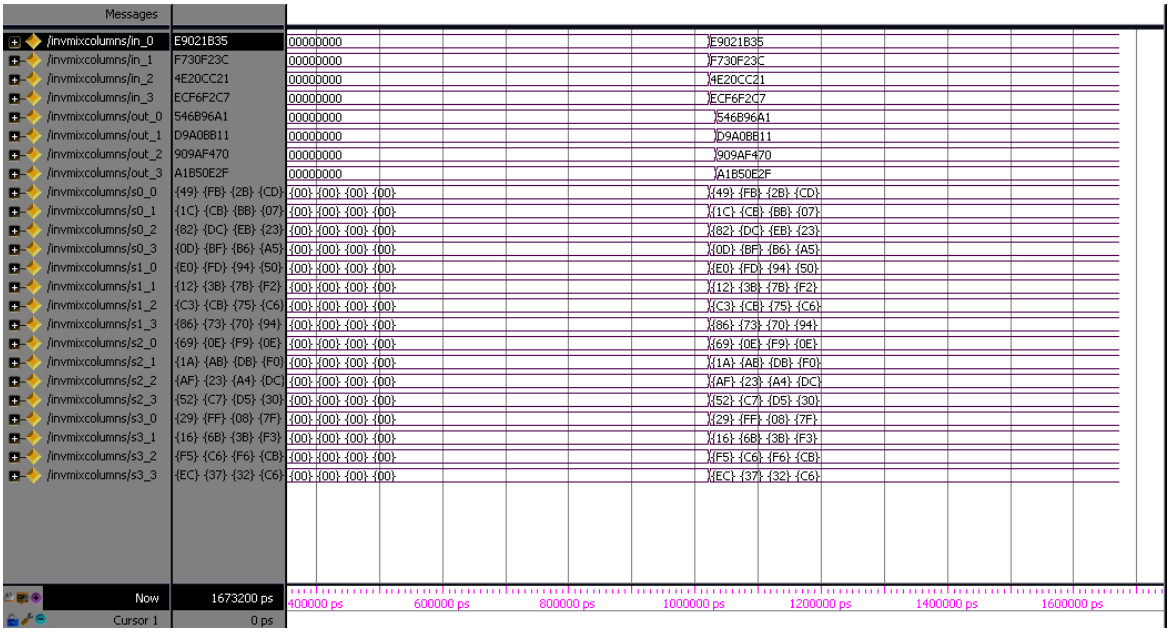

Abbildung 3.7: Funktionale Simulation von InvMixColumns.

Abbildung 3.8: Funktionale Simulation von der Rundenschlüsselerzeugung.

$$
\begin{aligned}
\text{w43:} &\quad 4D2B30C5 \\
\text{w42:} &\quad F307A78B \\
\text{w39 = w43 xor w33} &\quad BE2C974E \\
\text{Zwischenrechnung(Zw) für w36:} & \\
\text{w39:} &\quad BE2C974E \\
\text{invRotWord} &\quad 2C974EBE \\
\text{SubWord} &\quad 71882FAE \\
\text{Rcon} &\quad 36000000 \\
\text{Zw} &\quad 47882FAE \\
\text{w40} &\quad 13111D7F \\
\text{w36 = w40 xor Zw} &\quad 549932D1
\end{aligned}
\tag{3.18}
$$

Funktionale Simulation der Schaltung

Eine funktionale Simulation der gesamten Schaltung ist in der Abbildung 3.9 dargestellt. Hierbei sind keine Register verwendet worden, um die Schaltvorgänge in der Simulation sichtbar zu haben. Die Zeit für eine komplette Entschlüsselung ist abhängig von der symbolischen Laufzeit. Da jede Runde eine Laufzeit von 10 Nanosekunden hat und die Zuweisung an die Ausgänge ebenfalls mit einer Verzögerung von 10 Nanosekunden erfolgt, gilt für die zeitliche Berechnung T_{komb} die Gleichung (3.19). Zu beachten ist hierbei, dass jede Runde die vier gleichen Komponenten mit der gleichen Verzögerungszeit enthält. Streng genommen wird hier die Vorrunde mit der letzen Runde zusammengefasst, da die Vorrunde nur aus einer Komponente besteht und die letze Runde aus drei Komponenten.

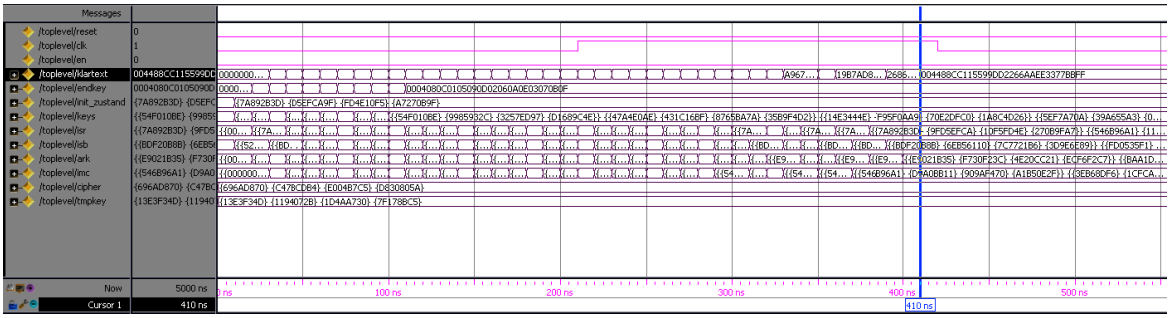

Abbildung 3.9: Funktionale Simulation einer kompletten Entschlüsselung.

$$T_{komb} = \text{Anzahl Runden} \cdot 4 \cdot 10\ \text{ns} + 10\ \text{ns} = 410\ \text{ns} \tag{3.19}$$

Bei der Schlüsselgenerierung werden 10 Komponenten mit jeweils 10 Nanosekunden Verzögerungszeit durchlaufen. Weitere 10 Nanosekunden vergehen durch die Zuweisung der Ergebnisse an den Ausgang. Dann ist der endgültige Schlüssel, welcher zur eigentlichen Verschlüsselung des Textes benutzt wurde, sichtbar am Ausgang vorhanden. Für die Simulation wurde der verschlüsselte Text und der Schlüssel fest vorgegeben. Die Matrizen (3.20) und (3.21) zeigen die Eingangs- und Ausgangsbytes in hexadezimaler Schreibweise, die für diese Simulation verwendet wurden. In der Simulation sind die Ergebnisse Vektoren, welche aus den Zeilen der Matrizen bestehen.

$$\text{verschlüsselter Text} = \begin{bmatrix} 69 & 6A & D8 & 70 \\ C4 & 7B & CD & B4 \\ E0 & 04 & B7 & C5 \\ D8 & 30 & 80 & 5A \end{bmatrix} \Rightarrow \text{Klartext} = \begin{bmatrix} 00 & 44 & 88 & CC \\ 11 & 55 & 99 & DD \\ 22 & 66 & AA & EE \\ 33 & 77 & BB & FF \end{bmatrix} \tag{3.20}$$

$$\text{1. Rundenschlüssel} = \begin{bmatrix} 13 & E3 & F3 & 4D \\ 11 & 94 & 07 & 2B \\ 1D & 4A & A7 & 30 \\ 7F & 17 & 8B & C5 \end{bmatrix} \Rightarrow \text{Schlüssel} = \begin{bmatrix} 00 & 04 & 08 & 0C \\ 01 & 05 & 09 & 0D \\ 02 & 06 & 0A & 0E \\ 03 & 07 & 0B & 0F \end{bmatrix} \tag{3.21}$$

3.5 Analyse des Klartextes

Um eine Entscheidung zu treffen, welcher Klartext und somit welcher Schlüssel der richtige ist, muss im Anschluss an jeden Entschlüsselungsvorgang der entstandene Klartext untersucht werden. Dazu wird die Entropie des Textes bestimmt. Die Entropie gibt den mittleren Informationsgehalt einer Nachricht pro Zeichen an. Je höher die Entropie ist, umso gleichmäßiger und zufälliger sind die einzelnen Zeichen im Text vorhanden. Um den richtigen Text identifizieren zu können, wird der Klartext mit der niedrigsten Entropie gesucht. Eine niedrige Entropie wird erreicht, wenn ein oder mehrere Zeichen häufig vorkommen. In einem normalen Satz wären dieses Leerzeichen und Vokale. Die Formel für die Entropie ist definiert als:

$$H = -\sum_{i=1}^{N} p_i \cdot \log_2 p_i \tag{3.22}$$

In der Formel (3.22) wird die Entropie H mit der Wahrscheinlichkeit für das Auftreten eines einzelnen Zeichens p_i über alle möglichen Zeichen N berechnet.

Für eine maximale Entropie gilt die Gleichung (3.23). Im Rahmen dieser Diplomarbeit werden 16 Byte zur Berechnung der Entropie verwendet. Somit ergibt sich für die maximale Entropie mit $N = 16$ der Wert $H_{max} = 4$.

$$\begin{aligned} H_{max} &= -\sum_{i=1}^{N} \tfrac{1}{N} \cdot \log_2 \tfrac{1}{N} \\ H_{max} &= \log_2 N \\ H_{max} &= \log_2 16 \\ H_{max} &= 4 \end{aligned} \tag{3.23}$$

3.5.1 Häufigkeiten einzelner Bytes im entschlüsselten Text

Bevor die Entropie errechnet werden kann, muss die Auftrittswahrscheinlichkeit der Bytes errechnet werden, welche in dem entschlüsselten Block vorkommen. Dazu muss lediglich überprüft werden, welches Byte wie oft vorkommt. Die Gesamtzahl an Bytes ändert sich nicht und ist konstant 16. Die Schwierigkeit hierbei ist, dass keine Bytes doppelt gezählt werden dürfen. Die Realisierung der Aufgaben erledigt eine Logik aus Komparatoren mit "Enable-"Eingängen. Da es für jedes Byte einen Komparator gibt, lässt sich über diese Eingänge der jeweilige Komparator abschalten, wenn das jeweilige Byte, für das der Komparator steht, schon gezählt wurde. Jeder Komparator liefert für alle nachfolgenden Komparatoren ein low-aktives Enable-Signal. Dadurch, dass diese Eingänge hintereinander geschaltet sind, lassen sich nachfolgende Komparatoren abschalten. Die Signale sind identisch mit dem Summenvektor exklusive dem MSB. Dadurch wird eine doppelte Zählung vermieden. Die nachfolgende Grafik 3.10 ist ein vereinfachtes Abbild, welches das Prinzip erläutert.

Hierbei hat der Eingangsblock vier Byte. Die Komparatoren enthalten absteigend ebenfalls die gleichen Bytes, sowie jeweils ein Byte zum Vergleichen. Der erste Komparator vergleicht das erste Byte mit den restlichen Bytes im Block. Als Ergebnis entsteht ein Vektor, welcher angibt, wie oft der Vergleich positiv ausgefallen ist. Im Ergebnisvektor sind drei Einsen, was bedeutet, dass das Byte `01` dreimal vorhanden ist. Ferner zeigen die Einsen die Stellen an, an denen das jeweilige Byte im Eingangsblock auftritt. Mit dieser Information werden die nachfolgenden Komparatoren abgeschaltet, um eine doppelte Zählung zu vermeiden. Der Ergebnisvektor exklusive dem MSB lautet `011`, somit ist das Enablesignal für den zweiten Komparator Null. Der zweite Komparator bleibt aktiv. Die Komparatoren drei und vier werden abgeschaltet, da hier eine Eins als Signal an den Enableeingängen anliegt. Die Summenergebnisse beider ausgeschalteter Komparatoren ist jeweils ein Nullvektor.
Der Summenvektor enthält zwar die richtige Anzahl an Einsen, jedoch nicht den richtigen Zahlenwert des Ergebnisses. Die Zahl im Ergebnisvektor ist 11, jedoch ist Drei das richtige Ergebnis. Deswegen werden die Einsen im Vektor mit Hilfe von Addierern zu dem rich-

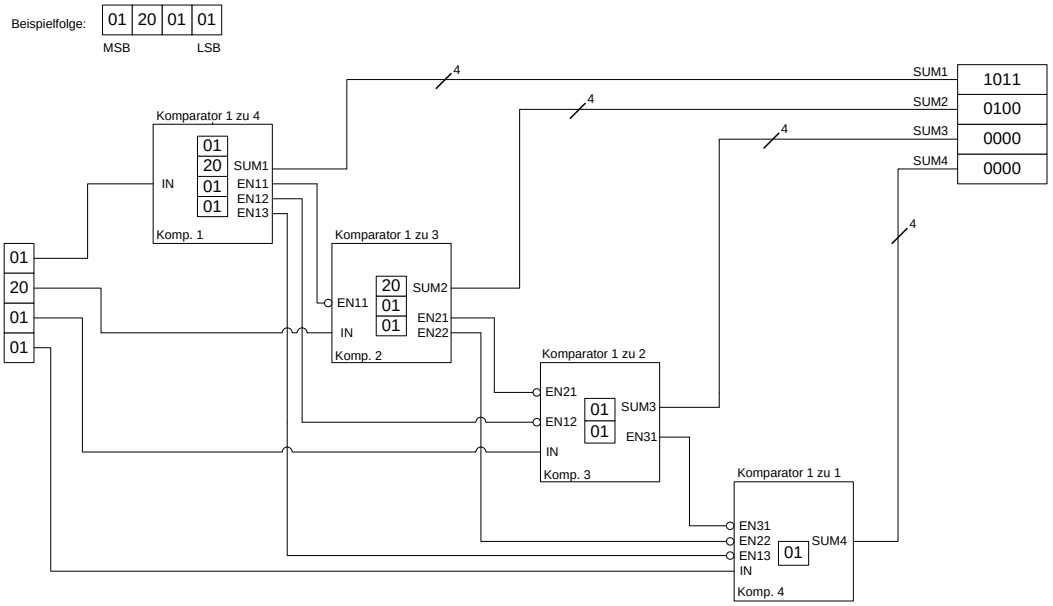

Abbildung 3.10: Kaskadierte Komparatoren zur Zählung von Bytes.

tigen Ergebnis zusammen gezählt. Die folgende Abbildung 3.11 zeigt einen vereinfachten Aufbau, um das Prinzip zu verdeutlichen.

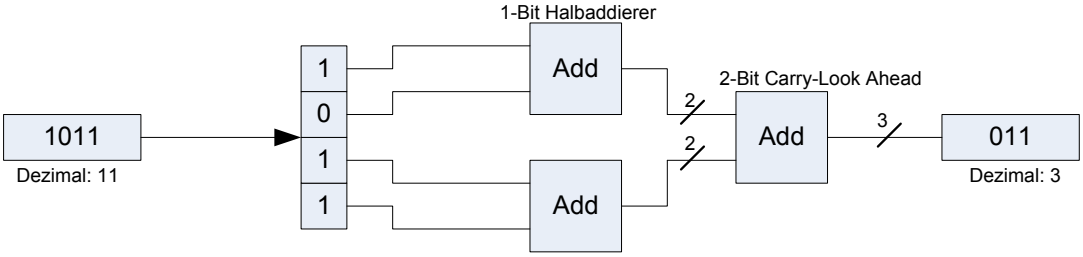

Abbildung 3.11: Addiererlogik zum Umwandeln des Zahlenwertes vom Komparator.

3.5.2 Entropieberechnung

Für eine Berechnung der Entropie werden insgesamt 16 Byte verwendet. Damit das Ergebnis möglichst schnell berechnet werden kann, wird eine Tabelle mit den passenden Ergebniswerten erzeugt. Für alle möglichen Wahrscheinlichkeiten wird der Logarithmus berechnet

und das Ergebnis in einem Speicher abgelegt. Hierbei wird auch gleichzeitig das Minuszeichen berücksichtigt. Die Ergebnisse befinden sich zwischen vier und $|0,0872900666|$. Für eine hohe Genauigkeit wird eine Berechnung auf vier Nachkommastellen angestrebt. Für die Darstellung der Zahlen als Gleitkommazahlen wird das Q-Format verwendet. Für den oben genannten Zahlenbereich ergibt sich ein Q3.15-Format, ein Vorzeichenbit wird nicht verwendet. Jedes Entropieergebnis hat somit eine Breite von 18 Bit. Die nachfolgende Tabelle 3.4 zeigt eine Übersicht von allen Werten.

p_i	$-p_i \cdot \log_2 p_i$ (gerundet)	$-p_i \cdot \log_2 p_i$ (Q-Format)	$-p_i \cdot \log_2 p_i$ (hexadezimal)
1/16	0,2500	000010000000000000	02000
2/16	0,3750	000011000000000000	03000
3/16	0,4528	000011100111110110	039F6
4/16	0,5000	000100000000000000	04000
5/16	0,5244	000100001100100000	04320
6/16	0,5306	000100001111101100	043EC
7/16	0,5218	000100001011001010	42CA
8/16	0,5000	000100000000000000	04000
9/16	0,4669	000011101111000100	03BC4
10/16	0,4238	000011011000111111	0363F
11/16	0,3716	000010111110010010	02F92
12/16	0,3113	000010011111011000	027D8
13/16	0,2434	000001111100101000	01F28
14/16	0,1686	000001010110010100	01594
15/16	0,0873	000000101100101101	00B2D
16/16	4,0000	100000000000000000	20000

Tabelle 3.4: Logarithmuswerte für die auftretenden Wahrscheinlichkeiten.

3.5.3 Funktionale Simulation der Entropieberechnung

Nachfolgend wird gezeigt, dass die Schaltung zur Entropieberechnung die korrekten Ergebniswerte liefert. Die Abbildung 3.12 zeigt eine funktionale Simulation der Schaltung zur Entropieberechnung. Der Eingangsvektor ist eine Folge von 16 Byte in hexadezimaler Schreibweise (3.2). Die Zwischenergebnisse sind in der Tabelle 3.5 aufgelistet. Die erste Spalte zeigt den jeweiligen Komparator, der das Ergebnis in der zweiten und dritten Spalte

liefert. In der vierten Spalte sind die umgewandelten Summenwerte der jeweiligen Komparatoren zu sehen. Zeilenweise muss diese Zahl mit der Anzahl der Einsen aus der zweiten Spalte übereinstimmen. Dies ist hier gegeben. In der fünften Spalte stehen die jeweiligen Ergebniswerte aus der Logarithmustabelle gemäß der Tabelle 3.4. In der zweiten Zeile der Tabellenüberschrift stehen die jeweiligen Signalnamen, wie sie in der Simulation 3.12 wiedergefunden werden können.

```
1 inbytes = F00FF00FF0000000000000000000AAAA
```

Listing 3.2: Eingangsvektor zum Testen.

Komparator	Ergebnis binär	Ergebnis hex toadd	Anzahl Bytes tologtable	Logarithmus log2
1	1010100000000000	A800	03	039F6
2	0101000000000000	0500	02	03000
3	0000000000000000	0000	00	00000
4	0000000000000000	0000	00	00000
5	0000000000000000	0000	00	00000
6	0000011111111100	07FC	09	03BC4
7	0000000000000000	0000	00	00000
8	0000000000000000	0000	00	00000
9	0000000000000000	0000	00	00000
10	0000000000000000	0000	00	00000
11	0000000000000000	0000	00	00000
12	0000000000000000	0000	00	00000
13	0000000000000000	0000	00	00000
14	0000000000000000	0000	00	00000
15	0000000000000011	0003	02	03000
16	0000000000000000	0000	00	00000

Tabelle 3.5: Ergebniswerte der funktionalen Simulation der Entropieberechnung.

Aus der letzen Spalte in Tabelle 3.5 wird das Endergebnis berechnet. Hierfür werden die Logarithmuswerte aus der Tabelle addiert. Das Ergebnis zeigt die Berechnung (3.24).

$$\begin{aligned} H &= 039F6_h + 03000_h + 03BC4_h + 03000_h \\ H &= D5BA_h \\ H &= 0011010101101110 10_b \end{aligned} \tag{3.24}$$

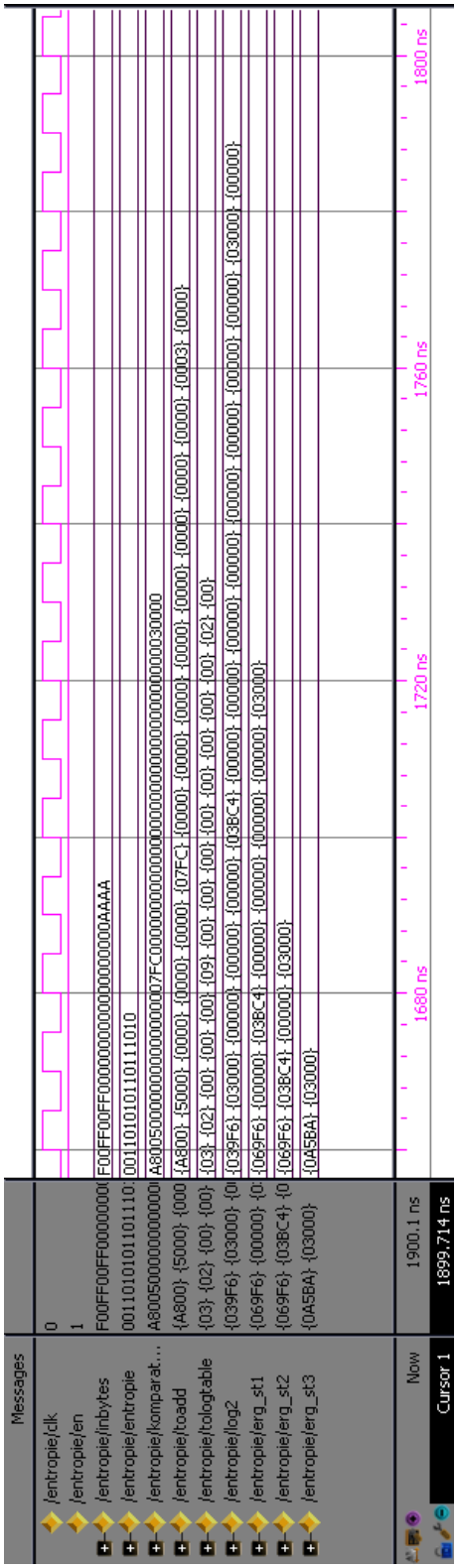

Abbildung 3.12: Funktionale Simulation der Entropieberechnung.

Abschließend muss das Ergebnis aus (3.24) vom Q-Format in eine dezimale Darstellung umgewandelt werden. Dieses ist in der Rechnung (3.25) zu sehen.

$$\begin{aligned} H = \;& 001101010110111010_b \\ H = \;& 2^0+2^{-1}+2^{-3}+2^{-5}+2^{-7}+2^{-8}+2^{-10}+2^{-11}+2^{-12}+2^{-14} \\ H = \;& 1+0,5+0,125+0,03125+0,0078125+0,00390625+0,0009765625 \\ & +0,00048828125+0,000244140625+0,00006103515625 \\ H \approx \;& \underline{\underline{1,6697}} \end{aligned} \tag{3.25}$$

Zur Kontrolle muss nun überprüft werden, ob das Ergebnis aus der Berechnung (3.25) identisch mit dem Ergebnis einer herkömmlichen Rechnung ist. Deswegen wird die Entropie mit den gleichen Eingangswerten und der Formel 3.22 berechnet. Nachfolgend ist die Berechnung gegeben (3.26) und die Ergebnisse sind identisch.

$$\begin{aligned} H = \;& -\sum_{i=1}^{N} p_i \cdot \log_2 p_i \\ H = \;& -(\tfrac{3}{16} \cdot \log_2 \tfrac{3}{16} + \tfrac{2}{16} \cdot \log_2 \tfrac{2}{16} + \tfrac{9}{16} \cdot \log_2 \tfrac{9}{16} + \tfrac{2}{16} \cdot \log_2 \tfrac{2}{16}) \\ H = \;& -(-0,4528-0,375-0,4669-0,375) \\ H \approx \;& \underline{\underline{1,6697}} \end{aligned} \tag{3.26}$$

3.6 Grafische Ausgabe der Ergebnisse

Um Ergebnisse auch manuell überprüfen zu können, werden diese auf einem LCD dargestellt. Die Ansteuerung für die Anzeige stammt von Benjamin Krill aus der Quelle [10]. Als Schnittstelle dient ein einfacher Automat, welcher die Ergebniswerte der Entschlüsselung für die Darstellung auf dem Display gegebenenfalls konvertiert und diese auf der Anzeige ausgibt.

3.6.1 Umrechnung des Q-Formates

Um die Entropie auf dem Display anzeigen zu können, muss dieser Zahlenwert vom Q-Format zurück in eine dezimale Darstellung konvertiert werden. Für die Stellen vor dem Komma ändert sich nichts, da diese in der normalen Form binär abgespeichert sind. Die Nachkommastellen müssen neu berechnet werden. Hierfür wird jeder binären Stelle eine Wertigkeit zugewiesen. In Abhängigkeit von den binären Stellen wird dann eine neue Zahl aus den vorhandenen Wertigkeiten addiert. Die folgende Tabelle 3.6 zeigt die Zuteilung

der neuen Werte zu den jeweiligen Stellen in der zu wandelnden Zahl. Der Maximalwert für das MSB ist festgelegt als 0,5. Damit jeder binären Stelle ein Wert zugewiesen werden kann, wird der Maximalwert immer durch zwei dividiert, gerundet und so die nachfolgenden Wertigkeiten bestimmt. Da binär keine Kommawerte gespeichert werden können, sind die Zahlen mit 100000 multipliziert worden, um so als natürliche Zahlen vorzuliegen. Auf dem LCD wird die Entropie dann zusammengesetzt aus den Vorkommastellen, einem Komma und den Nachkommastellen.

binäre Stelle	Wertigkeit dezimal	Wertigkeit hexadezimal
MSB	50000	0C350
2	25000	061A8
3	12500	030D4
4	06250	0186A
5	03125	00C35
6	01563	0061A
7	00781	0030D
8	00391	00186
9	00196	000C3
10	00098	00061
11	00049	00030
12	00024	00018
13	00012	0000C
14	00006	00006
LSB	00003	00003

Tabelle 3.6: Konvertierung vom Q-Format ins Dezimalsystem.

Um die Vorgehensweise zu verdeutlichen, wird das Beispiel (3.24) mit der Tabelle berechnet. Die Beispielrechnung zeigt (3.25). Hier ist zu sehen, dass das Entropieergebnis auf vier Nachkommastellen genau berechnet werden kann.

$$\begin{aligned} H &= 001101010110111010_b \\ H &= 001,101010110111010_b \\ H &= 1,(50000+12500+3125+781+391+98+49+24+6) \\ H &= 1,66974 \\ H &\approx \underline{\underline{1,6697}} \end{aligned} \tag{3.27}$$

3.6.2 Konvertierung der Zahlen

Alle Zahlen sind in der binären Darstellung gespeichert. Wenn diese Zahlen ausgegeben werden, so wird ein Offset von 48 dazu addiert, damit die Zahlen mit der Codierung im ASCII-Format übereinstimmen. Wenn der Ergebniswert auf dem Display ausgegeben wird, erscheint die gewünscht Zahl in hexadezimaler Darstellung. Für eine dezimale Anzeige müssen die Zahlen zuvor umgewandelt werden. Hierfür gibt es einen Algorithmus, welcher aus einer Schiebeoperation und einer Addition mit drei besteht. Hierbei wird die Zahl um ihre binäre Anzahl an Stellen sequenziell um eins nach links geschoben und nach jeder Schiebeoperation wird für jedes Nibble eine drei addiert, wenn der Wert des Nibbles größer als vier ist. Am Ende ergibt sich daraus dann eine BCD-Zahl [11]. Der Quellcode dazu befindet sich im Anhang D. Das besondere an diesem ist, dass die Anzahl der zu wandelnden Ziffern als Parameter übergeben wird.

3.7 Gesamtübersicht der Schaltung

Die Grafik 3.13 zeigt eine Übersicht über die komplette Schaltung. Abgebildet ist nur der Datenpfad des Systems. Die dazugehörigen Steuersignale werden anschließend beschrieben. Die ovalen Symbole repräsentieren Elemente aus kombinatorischer Logik. Die Quadrate stehen für Komponenten, welche mindestens ein Register enthalten und dadurch vom Takt abhängig sind. Der Geheimtext ist fest im System gespeichert. Mit einem Steuersignal zählt der Zähler von Null bis zum maximalen Wert hoch, welcher von der eingestellten Bitbreite abhängt. Dadurch werden die benötigten Schlüssel generiert. Mit jedem Schlüssel wird eine komplette AES-Entschlüsselung durchgeführt, wie sie oben beschrieben ist. Nach der Entschlüsselung wird das Ergebnis in ein Schieberegister mit variabler Speicherbreite übernommen. Das Schieberegister ist nötig, um festzustellen, wann eine Entschlüsselung beendet ist. Da die Entschlüsselung nicht getaktet ist, ändert sich das Ergebnis permanent, bis ein Endzustand erreicht wird. Um diesen Zustand zu erfassen, wird das Ergebnis der Entschlüsselung in das Schieberegister übernommen. Wenn alle Werte im Register identisch sind, dann wird von einem stabilen Endzustand ausgegangen und ein Signal wird gesetzt, welches diesen Zustand anzeigt. Parallel dazu wird die Entropie von dem zeitlich gerade vorliegenden Ergebnis der Entschlüsselung berechnet. Da an dieser Stelle ein Eingangsregister die Werte übernimmt, werden nur Ergebnisse übernommen, welche mit der aktuellen steigenden Taktflanke anliegen. Wichtig ist hierbei, dass die Entropieberechnung schneller erfolgen muss, als das Schieberegister mit gleichen Werten gefüllt sein kann. Dadurch ist sichergestellt, dass zu dem Zeitpunkt, an dem alle Werte im Schieberegister gleich sind, die Entropie von diesem Ergebnis berechnet ist. Zwei Komparatoren überprüfen zum einen, ob jedes Zeichen des entschlüsselten Textes innerhalb der ASCII-Tabelle liegt, zum anderen, ob das Ergebnis der Entropieberechnung kleiner ist als das gespeicherte Ergebnis einer vorhe-

rigen Entschlüsselung. Wenn beide Fälle zutreffen, dann werden der Klartext, der Schlüssel und der Entropiewert jeweils in einem Register gespeichert.

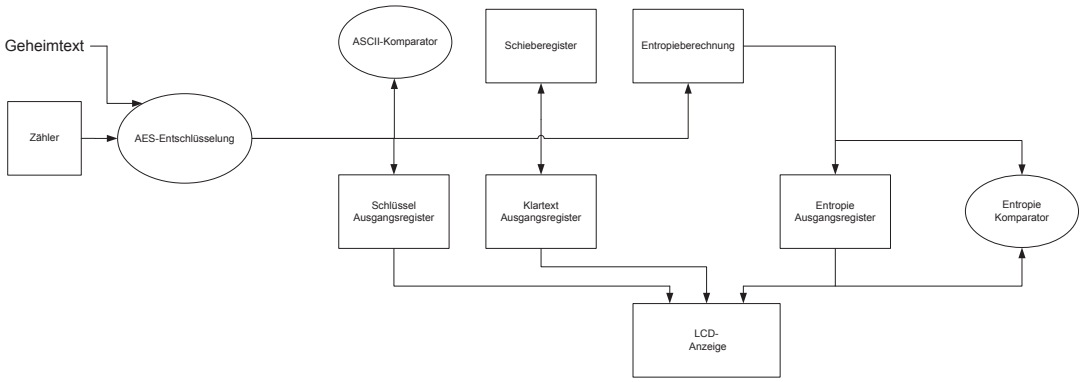

Abbildung 3.13: Der Datenpfad der kompletten Schaltung.

3.7.1 Steuerautomat der Schaltung

Damit eine komplette Schlüsselsuche realisiert werden kann, wird eine Steuereinheit benötigt. Diese steuert alle Komponenten der Schaltung, damit die korrekten Ergebnisse erzielt werden. Hierfür wird ein Zustandsautomat realisiert. Da der Automat zur Steuerung von internen Signalen verwendet wird und keine zeitkritischen externen Eingänge verarbeiten muss, wird ein Moore-Automat verwendet. Bei diesem Automatenmodell sind die Steuersignale nur vom jeweiligen Zustand des Automaten abhängig, nicht aber von Eingangssignalen. Dadurch sind alle Steuersignale synchron zum Takt. Im Quelltext wurde eine Zwei-Prozess-Darstellung nach der Huffman-Normalform (vgl. [RS03] Seite 131) für den Automaten gewählt. Hierbei wird ein taktsynchroner Prozess für den Zustandsspeicher und ein weiterer Prozess für die Berechnung von Folgezuständen und Ausgangssignalen verwendet. Eine umfassende Beschreibung des Automaten ist in dem Zustandsdiagramm 3.14 abgebildet. Zusätzlich zu dem Zustandsdiagramm zeigen die Tabellen 3.7 und 3.8 eine Übersicht aller Signale.

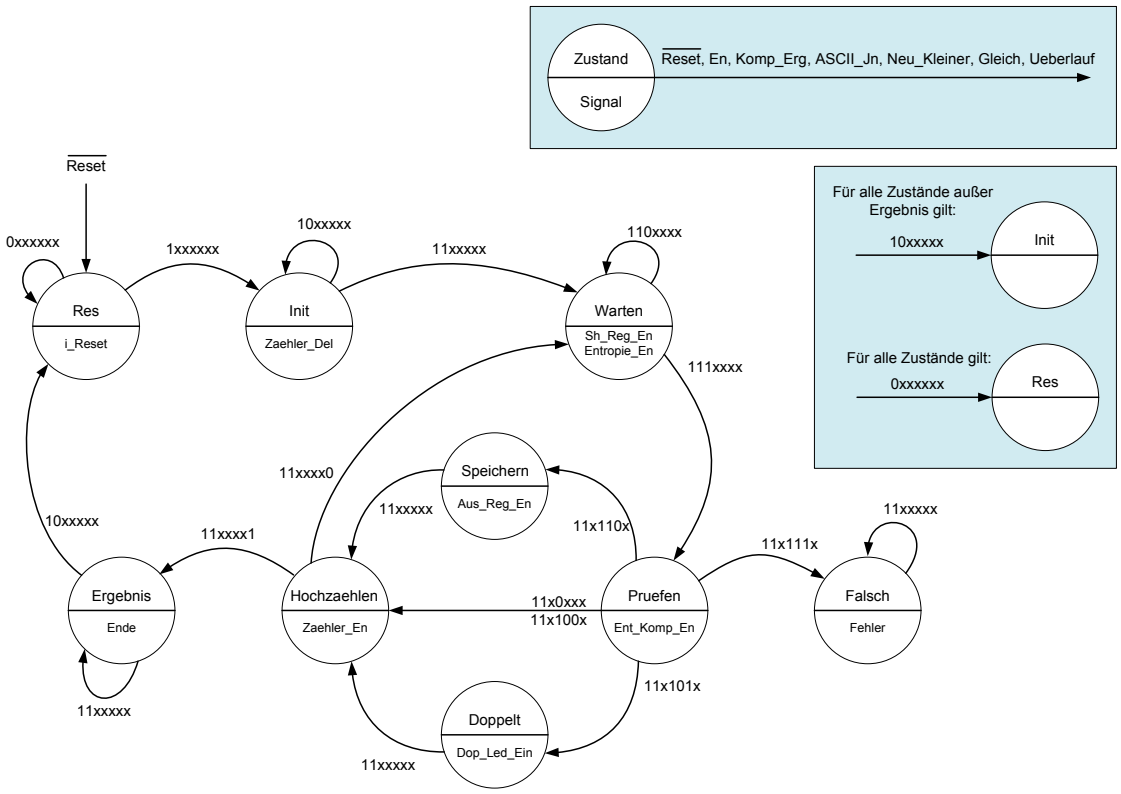

Abbildung 3.14: Das Zustandsdiagramm des Steuerautomaten.

Signalname im Quellcode	Funktion
$\overline{\text{RESET}}$	asynchroner Reset, gültig bei einer Null als Signal
EN	schaltet das System ein
KOMP_ERG	zeigt an, wann im Schieberegister alle Werte gleich sind
ASCII_JN	zeigt an, ob alle Zeichen im Wertebereich 0..127 liegen
NEU_KLEINER	zeigt an, ob der aktuelle Entropiewert kleiner als der bereits gespeicherte Wert ist
GLEICH	zeigt an, ob der aktuelle Entropiewert identisch mit dem gespeichertem Wert ist
UEBERLAUF	Das Signal zeigt den Überlauf des Zählers an, alle Schlüssel wurden dann ausprobiert

Tabelle 3.7: Alle Eingangssignale des Automaten.

Signalname im Quellcode	Funktion
i_RESET	synchroner Reset, gültig bei einer Null als Signal
ZAEHLER_DEL	löscht den aktuellen Zählstand und setzt den Zähler zurück
SH_REG_EN	schaltet das Schieberegister ein oder aus
ENTROPIE_EN	schaltet die Entropieberechnung ein oder aus
ENT_KOMP_EN	schaltet den Entropiekomparator zum Vergleichen ein oder aus
FEHLER	schaltet eine LED auf dem Entwicklungsboard ein
AUS_REG_EN	wenn das Bit gesetzt ist, dann werden der aktuelle Klartext, Schlüssel und die zugehörige Entropie gespeichert
DOP_LED_EIN	wenn zwei Entropiewerte gleich sind, dann wird mit dem Signal eine LED eingeschaltet.
ZAEHLER_EN	aktiviert den Zähler zur Schlüsselgenerierung
ENDE	schaltet eine LED ein, um die Beendigung des Angriffs anzuzeigen

Tabelle 3.8: Alle Ausgangssignale des Automaten.

Bei der Schaltung wird der asynchrone Reset vom Entwicklungsboard durch den Automaten in einen synchronen Reset umgewandelt. Dieser synchrone Reset ist für alle Komponenten gültig. Dadurch wird sichergestellt, dass alle Komponenten den Reset zur gleichen Zeit erhalten und kein asynchrones Verhalten auftritt.

3.8 Probleme

Bei der Entwicklung des Systems ist besonders darauf zu achten, dass die Vektoren spaltenweise eingegeben werden, alle Operationen werden hingegen zeilenweise ausgeführt. Des Weiteren wird die Schlüsselerzeugung ebenfalls zeilenweise abgearbeitet. Eine Vertauschung von Zeilen und Spalten ist zu vermeiden. Bei der Schlüsselerzeugung ist außerdem zu beachten, dass die Generierung der Rundenschlüssel in umgekehrter Reihenfolge erfolgt.

4 Durchführung der Kryptanalyse

4.1 Übersicht über das Entwicklungsboard

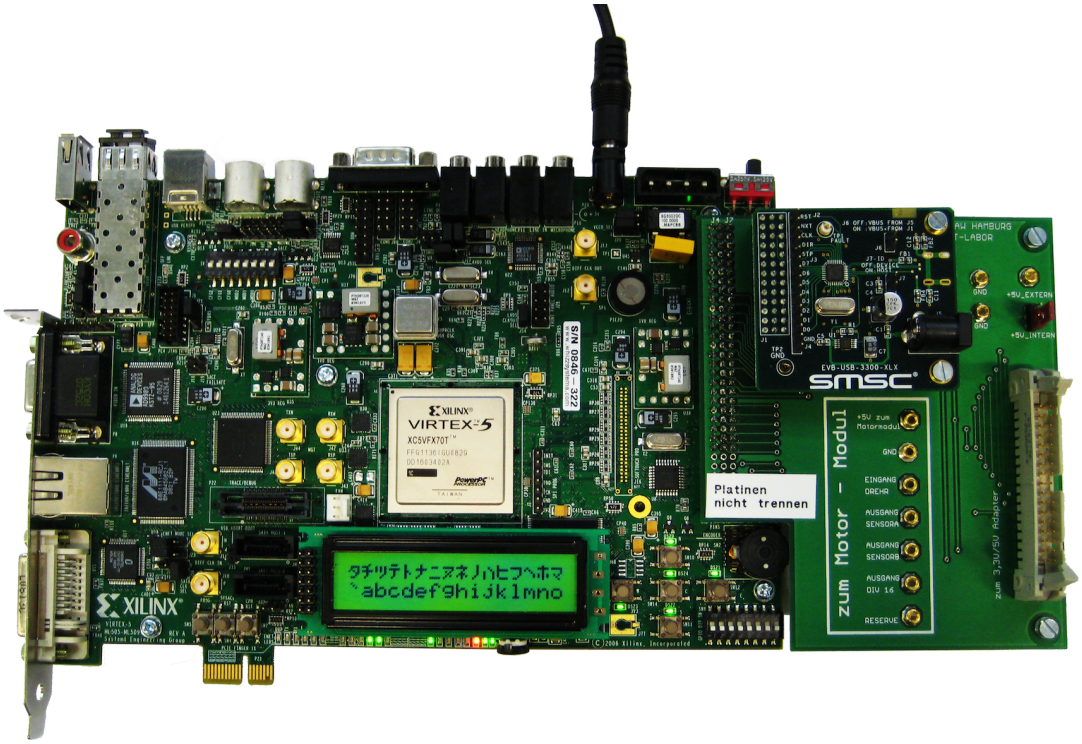

Abbildung 4.1: Das Entwicklungsboard ML507

Für die Implementierung in der digitalen Hardware wird ein Entwicklungsboard vom Typ ML507 der Firma Xilinx verwendet. Der Kern des Boards ist ein FPGA vom Typ Virtex-5, welcher für diese Diplomarbeit genutzt wird. Die genaue Bezeichnung des FPGAs ist XC5VFX70T. Die Gehäuseform ist vom Typ FFG1136 und der speed grade ist -1. Des Weiteren ist das Entwicklungsboard mit einer Vielzahl von Schnittstellen bestückt, welche hier nicht näher benannt werden sollen. Neben dem FPGA wird das zweizeilige LC-Display

sowie diverse LEDs und DIP-Schalter verwendet. Die Grafik 4.1 zeigt die verwendete Hardware. An der linken Seite ist eine Erweiterungsplatine angebracht, welche im Labor für Digitaltechnik während des Praktikums benötigt wird. Über die Anschlüsse wird im folgenden Abschnitt eine Kommunikation mit einem weiteren Board realisiert ([Xil09a]).

4.2 Implementierung

Das fertige Design wird über die JTAG-Schnittstelle auf das Entwicklungsboard geladen. Für eine erste Funktionsüberprüfung wurde der Großteil des Schlüssels vorgegeben und die unteren Bits wurden mit dem Zähler generiert. Um eine beliebige Schaltung mit einer höheren Frequenz takten zu können, werden lange kombinatorische Pfade mit Registern aufgespalten. Statt eines Pfades mit einer relativ langen Laufzeit entsteht dann ein Pfad, welcher aus mehreren Registern und kurzer Laufzeit zwischen den Registern besteht. Da nun mehrere Register in dem Pfad vorhanden sind, werden auch mehrere Taktzyklen benötigt, bis der gewünschte Pegel über die Leitung übertragen ist. Allerdings ist die Laufzeit zwischen den Registern deutlich geringer als die Laufzeit ohne Register, somit kann nun ein höherer Takt verwendet werden.
Dieses Prinzip wird auch bei der hier entworfenen Schaltung angewendet, um die Taktfrequenz deutlich zu erhöhen. Dazu werden Register zwischen jede Runde des AES-Algorithmus und bei der Entropieberechnung eingesetzt. Dadurch kann ein deutlich höherer Takt gewählt werden. Der Takt der Schaltung wurde von 27 MHz auf 200 MHz erhöht. Die nachfolgende Tabelle 4.2 zeigt einen Überblick über die verschiedenen Zeiten bei unterschiedlicher Schlüssellänge. Für die Zeitmessung ist eine Sekundenuhr entwickelt worden, deren aktuelle Zeit auf dem LC-Display angezeigt werden kann. Wenn eine komplette Schlüsselsuche beendet ist, dann wird die Uhr vom Steuerautomaten angehalten und die Laufzeit kann abgelesen werden. Für die Entschlüsselung, sowohl in Hardware als auch in Software, wird die Einstellung aus der Tabelle 4.1 verwendet.

Testvektor und Schlüssel:	
Eingangstext:	Das ist ein Text
Schlüssel (hex):	12 34 56 78 99 87 65 43 21 12 34 56 78 99 87 65
Geheimtext (hex):	FB 0B 5A 59 FD 25 C3 7A F5 93 F3 A7 2A 60 F2 C5
Entropie Klartext:	3,2028

Tabelle 4.1: Einstellungen für den Geschwindigkeitstest.

	Implementierung mit langen Pfaden	Implementierung mit Zwischenregistern
24 Bit	3 Sekunden	2 Sekunden
28 Bit	42 Sekunden	36 Sekunden
32 Bit	666 Sekunden	594 Sekunden
36 Bit	11067 Sekunden	9003 Sekunden

Tabelle 4.2: Verschiedene Laufzeiten im Vergleich.

Bei der Implementierung treten einige Warnungen auf, welche in der nachfolgenden Tabelle 4.3 zusammengefasst werden.

Warnung	Erläuterung
Index value(s) does not match array range, simulation mismatch.	Für das Display wird ein Array mit insgesamt 42 Elementen verwendet. Ein Pointer kann die Elemente direkt adressieren. Um die Adresse anzugeben, hat der Pointer eine größe von 8 Bit. Damit können theoretisch 256 Elemente adressiert werden. Da das Array aber nur 42 Elemente hat, wird eine Warnung angezeigt.
Signal <Q_DEZ_BCD<27:20> is assigned but never used.	Bei der Wandlung der Entropie von einer Binärzahl zu einer BCD-Zahl erhöht sich der Speicherbedarf für diese Zahl um acht Bit. Diese Ergeben sich aus den Überträgen. Da der Zahlenbereich der Entropie aber begrenzt ist, werden diese zusätzlichen Stellen nicht benötigt und sind somit auch nicht angeschlossen.
FF/Latch <alle_runden[0].RUNDE0.-REG_0/AUS_1_12> (without init value) has a constant value of 0 in block <toplevel>.	Dadurch, dass große Teile des Schlüssels vorgegeben sind und somit auch einen festen Wert haben, ist das jeweilige Flipflop für das Signal überflüssig. Diese Warnung zeigt dies an und das Flipflop verschwindet während des Optimierungsprozesses. Dieses Beispiel ist repräsentativ für andere Signalnamen.
clock net ENT/LOG2TABLES[0].LOG_REG/-RESET_inv with clock driver ENT/LOG2TABLES[0].LOG_REG/-RESET_inv_BUFG drives no clock pins	Es scheint, als ob hier ein Signal auf einen Pfad gelegt wird, welcher normal für einen Takt vorgesehen ist. Da das Signal selbst aber kein Taktsignal ist, erscheint diese Warnung.

Tabelle 4.3: Verschiedene Warnungen während der Synthese

4.3 Auswahl der Software

Für die angestrebte Verifikation des Geschwindigkeitsvorteils wird eine kleine Auswahl an unterschiedlichen Softwarelösungen verwendet. Hierbei wurde der Schwerpunkt auf hardwarenahe Programmiersprachen gelegt, da diese für die meisten Anwendungen einen schnelleren Programmablauf als Skriptsprachen haben [12].

4.3.1 Implementierung von Codeplanet

Der hier verwendete Quellcode stammt von dem Internetportal Codeplanet [13]. Dabei handelt es sich um eine Seite für Softwareentwickler, welche diverse Tutorials und Informationen enthält. Auf der Webseite wird der AES-Algorithmus umfassend vorgestellt und erläutert. Der Quellcode ist ebenfalls gut beschrieben. Nachteilig ist, dass sich zum einen Fehler eingeschlichen haben, zum anderen wird die Bibliothek stdint.h verwendet, welche unter Umständen Probleme machen kann, da diese nicht zum Standard von allen Entwicklungsumgebungen gehört. Diese Bibliothek enthält Datentypen, welche unabhängig vom Betriebssystem immer die gleiche Größe haben sollen. Die folgende Grafik 4.2 zeigt eine Analyse des erstellten Programms unter dem Aspekt der Laufzeit. Durchgeführt wurde dieses auf einem Macbook Pro mit einem Core 2 Duo Prozessor bei einem Takt von 2,2 GHz. Das besondere bei dieser Grafik ist, dass klar erkennbar ist, welcher Teil des Programms am häufigsten durchlaufen wird, nämlich die Matrixmultiplikation für die inverse MixColumns-Operation. Durchgeführt wurde hier eine Kryptanalyse mit einem Teilschlüssel von 16 Bit. Die Laufzeit für die Entschlüsselung beträgt ungefähr eine Sekunde. Dieser Wert ist in diesem Fall ungenau, da die zeitliche Auflösung sekundengenau ist.

4.3.2 Implementierung von Cryptool

Cryptool ist eine Lernsoftware für Kryptographie. Hiermit können verschiedene Algorithmen nachvollzogen und analysiert werden. Das Programm Cryptool kann schon als fertige ausführbare Datei heruntergeladen werden und muss somit nicht extra kompiliert werden [14]. Allerdings ist es nur für das Betriebssystem Windows verfügbar. In dieser Arbeit wurde die Version 1.4.30 verwendet, welche in C++ geschrieben wurde. Der Quelltext ist frei verfügbar. Das Programm wurde ausgewählt, weil alle benötigten Funktionen vorhanden sind und der Quelltext ebenfalls übersichtlich ist.

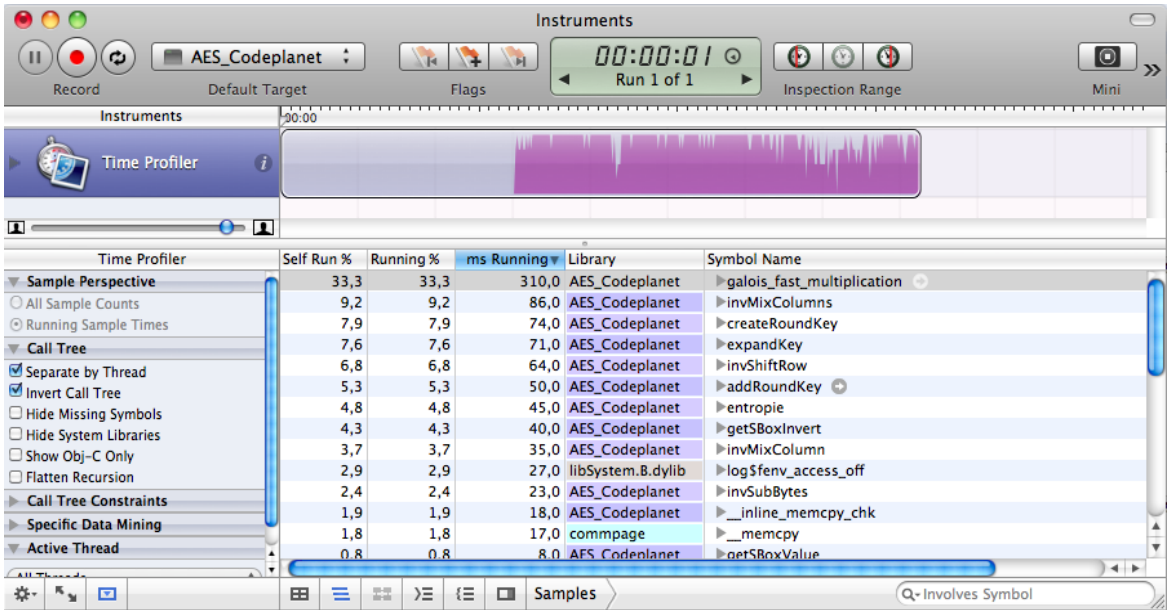

Abbildung 4.2: Laufzeitanalyse bei einem 16 Bit Schlüssel mit dem Programm Codeplanet.

4.3.3 Programmcode von Brian Gladman

Der hier verwendete Code stammt von dem britischen Mathematiker, Physiker und Doktor der Elektrotechnik Brian Gladman [15]. Das Interessante an seiner Implementierung ist, dass der Quelltext fast nur aus Präprozessoranweisungen besteht. Das erweckt den Eindruck, dass der Programmablauf im Gegensatz zu anderen Implementierungen sehr schnell ist, da bei dieser Implementierung auf Schleifendurchläufe nahezu verzichtet wurde.

4.4 Vergleich der Laufzeiten von den Implementierungen

Alle drei Softwarelösungen wurden für den Vergleich auf einem Computer mit dem Betriebssystem Windows Vista Business mit Service Pack 2 ausgeführt. Das Betriebssystem hat eine 64 Bit Architektur, passend zum verwendeten Prozessor. Der Prozessor ist ein Intel®Core™2 Duo E8400 mit zwei Kernen, welche jeweils mit drei Gigahertz getaktet sind. Die Programme werden mit Hilfe einer Entwicklungsumgebung erstellt und dann manuell ausgeführt. Die Ausnahme ist Cryptool, da dieses schon als fertiges Programm vorhanden ist.
Für die Messung der Zeiten wird aus der Bibliothek `time.h` die Funktion `clock()` verwendet. Die Funktion gibt als Rückgabewert die Anzahl an Taktzyklen zurück, welche seit dem Start des Programms vergangen sind. Für eine Umrechnung in Sekunden wird dieser

Wert durch die Konstante CLOCKS_PER_SEC, welche die Anzahl an Taktzyklen pro Sekunde enthält, dividiert. Nachfolgender C-Code 4.1 zeigt die Zeitmessung für die Software von Gladman und von Codeplanet. Beim Cryptool ist eine Uhrzeitfunktion vorhanden.

```
1  // Start des Programms
   // Initialisierungen
3  clock_t startwert,endwert;

5   startwert = clock();
       {
7          // Entschluesselung fuer alle moeglichen Schluessel
       }
9  endwert = clock();

11 // Ergebnisse ausgeben
   printf('Sekunden: %d', (endwert-startwert)/CLOCKS_PER_SEC);
```

Listing 4.1: Beispielcode zum Messen der Ausführungszeit.

Um die Ausführungsgeschwindigkeit eines Programms noch mehr zu erhöhen, kann die Priorität des Prozesses von einem Programm erhöht werden. Dieses bringt dann einen Geschwindigkeitsvorteil im unteren einstelligen Prozentbereich. Je größer die Schlüssel werden, desto geringer wird allerdings der Vorteil durch eine höhere Priorität des Programms. Die Tabelle 4.4 zeigt nun die verschiedenen Laufzeiten in Sekunden für die unterschiedlichen Implementierungen bei unterschiedlichen Schlüssellängen.

	Hardware	Codeplanet	Cryptool	Gladman
16 Bit	<1	< 1	<1	< 1
20 Bit	<1	4	2	< 1
24 Bit	2	60	21	7
28 Bit	36	964	288	111
32 Bit	594	15477	4620	1793

Tabelle 4.4: Verschiedene Laufzeiten im Vergleich, alle Werte in Sekunden.

Die folgende Grafik 4.3 zeigt die Werte aus der Tabelle 4.4. Die Anzahl an verwendeten Bits im Schlüssel ist auf der x-Achse aufgetragen und die Zeit in Sekunden befindet sich auf der y-Achse. Des Weiteren wurde die y-Achse logaritmisch skaliert. Dadurch werden die Kurven linear dargestellt. Das bedeutet, dass die Laufzeit exponentiell wächst, was nicht

sonderlich überrascht, da mit jedem Bit, welches zur Schlüssellänge hinzukommt, die Gesamtanzahl an möglichen Schlüsseln exponentiell steigt. Die schwarzen Linien sind die jeweiligen Trendlinien zur Messung und zeigen die theoretischen Werte für weitere Schlüssellängen an. Es ist zu erkennen, dass die Implementierung in der digitalen Hardware am schnellsten ist. Der Code von Codeplanet hat die längste Laufzeit.

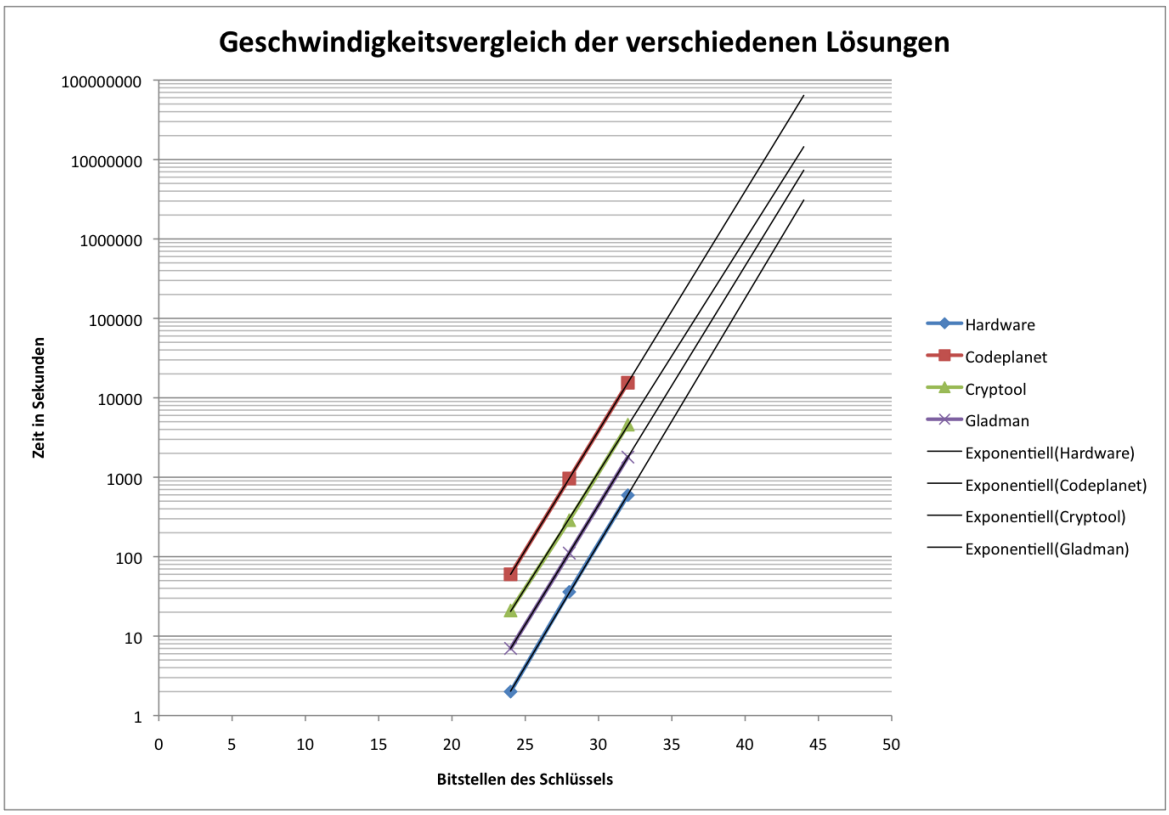

Abbildung 4.3: Logarithmische Darstellung der Laufzeiten in Abhänigkeit von der Schlüssellänge.

4.5 Verifikation eines Parallelbetriebs

Um die Geschwindigkeit weiter zu erhöhen, kann versucht werden, die einzelnen Aufgaben zu verteilen. In der digitalen Hardware bedeutet dies, dass ein weiteres Entwicklungsboard verwendet wird. Auf beiden FPGAs werden die gleichen Operationen ausgeführt, allerdings werden auf dem einen Entwicklungsboard nur alle geraden Schlüssel zum entschlüsseln verwendet und auf dem anderen kommen nur die ungeraden Schlüssel zur Verwendung. Am Ende müssen die Ergebnisse an einer zentralen Stelle gesammelt und verglichen werden. Für Softwarelösungen kann ein ähnliches Verfahren angewendet werden. Hierbei werden

dann zwei Prozesse erzeugt, welche nebeneinander ablaufen. Diese Parallelität hängt vom Betriebssystem ab, da dieses die Ressourcenzuteilung an die aktiven Prozesse regelt.

4.5.1 Realisierung des Parallelbetriebs für die Hardware

Für einen parallelen Betrieb werden zusätzlich zu den verwendeten Komponenten für die Kryptanalyse weitere Module benötigt. Die zwei wichtigsten Neuerungen sind ein Sender und ein Empfänger. Der Sender enthält eine zusätzliche Logik, um das entschlüsselte Ergebnis an den Empfänger zu senden. Der Empfänger benötigt eine Komponente, um mit dem Sender zu kommunizieren und die Ergebnisse in Empfang zu nehmen. Des Weiteren wird eine Logik benötigt, welche die Entropien der entschlüsselten Texte vergleicht und die Ergebnisse mit der kleineren Entropie auf dem Display ausgibt. Bei dem Sender werden keine Informationen auf dem Display ausgegeben und die Sekundenuhr wird ebenfalls nicht verwendet. Die Uhrzeit wird nur noch im Empfänger gemessen. Somit muss die Kryptanalyse auf beiden Entwicklungsboards gleichzeitig gestartet werden. Die Messung ist erst beendet, sobald der Sender seine Daten zum Empfänger übertragen hat. Das ist nötig, weil die Laufzeiten auf jedem Entwicklungsboard unterschiedlich sein kann. Mit diesem Verhalten wird erreicht, dass die Messung die Laufzeiten beider Entwicklungsboards beinhaltet. Wenn der Sender seine Daten senden kann, wird eine LED eingeschaltet. Bei dem Empfängerboard zeigen zwei LEDs an, ob das Board bereit ist, Daten zu empfangen und ob der Empfang abgeschlossen wurde. Wenn Daten empfangen wurden, dann wird die erste LED ausgeschaltet und die zweite LED wird eingeschaltet. Nachdem die Übertragung beendet ist, ist automatisch auch der Angriff beendet.
Die Dateiübertragung erfolgt mit einem Vier-Phasen-Handshake (dt. Handschlag) für jedes Bit. Insgesamt werden 274 Bits auf einem Kanal übertragen. Da jedes Bit mit einem Handshake übertragen wird, werden keine Start- oder Stopbits benötigt. Der Empfänger muss lediglich die Anzahl an empfangenen Bits abzählen um zu entscheiden, wann die Übertragung abgeschlossen ist. Der Empfangene Bitstrom wird nach dem Empfang aufgeteilt in die Bestandteile Schlüssel, Klartext und Entropie.

4.5.2 Realisierung des Parallelbetriebs für die Software

Zum Vergleich wird die Implementierung von Brian Gladman ausgewählt, da diese sich als die schnellste Implementierung unter den ausgewählten Algorithmen herausgestellt hat. Um hier einen parallelen Betrieb zu realisieren, wird die Entschlüsselung auf zwei Threads ausgelagert. Der eine Thread durchläuft alle geraden Schlüssel, der andere verarbeitet parallel dazu alle ungeraden Schlüssel. Das Hauptprogramm steuert die Zeitmessung und die Ausgabe der Ergebnisse auf dem Bildschirm. Die Zeitmessung wird vor dem Start beider Threads gestartet und erst gestoppt, wenn beide Threads beendet sind.

4.5.3 Geschwindigkeitsvergleich im Parallelbetrieb

Die nachfolgende Tabelle 4.5 zeigt die Ergebnisse der Implementierungen in digitaler Hardware und Software. Die Abbildung 4.4 zeigt die Meßwerte. Hierbei ist zu erkennen, dass sich die Laufzeiten bei beiden Implementierungen nahezu halbiert haben. Das Verhältnis der Laufzeiten zueinander ist nahezu unverändert und liegt bei einem Faktor von drei.

	Hardware	Gladman
24 Bit	1	4
28 Bit	18	57
32 Bit	293	892
36 Bit	4923	14330

Tabelle 4.5: Verschiedene Laufzeiten im Vergleich beim Parallelbetrieb, alle Werte in Sekunden.

4.6 Probleme

Ein kleines Problem ist die Entropieberechnung. Hierbei ist das Ergebnis nicht immer eindeutig, weil die kurze Textfolge von nur 16 Bytes dieses nicht zulässt. So passiert es, dass das richtige Ergebnis nicht gefunden wird, da ein anderes Ergebnis eine noch kleinere Entropie besitzt. Aus diesem Grund wird bei jedem Text verglichen, ob sich das Ergebnis aus zulässigen Zeichen zusammensetzt. Diese Zeichen sind Groß- und Kleinbuchstaben sowie Zahlen und Leerzeichen. Sollte das entschlüsselte Ergebnis nicht ausschließlich aus diesen Zeichen bestehen, so wird es nicht gespeichert und der nächste Schlüssel wird ausprobiert. Dieses Verhalten beschleunigt die Laufzeit einer Kryptanalyse, da dadurch nicht für alle Ergebnisse die Entropie berechnet werden muss. Es wurde sowohl für die digitale Hardware, als auch für die Software realisiert.
Das größte Problem ist, dass das LCD mit einem sehr niedrigen Takt betrieben wird. Der Ansteuerungsautomat ist mit 27 MHz getaktet, das Display selbst nur mit 27KHz. Dadurch, dass die übrige Schaltung mit 200 MHz getaktet ist, ist das Display nicht synchron mit der gesamten Schaltung. Die Register, welche die Ausgangssignale zum LCD bereitstellen, werden relativ zum Displaytakt sehr schnell beschrieben. Außerdem sorgt ein Signal dafür, dass die Werte nicht vom Display übernommen werden, wenn die Register gerade beschrieben werden. Da dieses Signal aber auch nur für einen Takt mit der Frequenz 200 MHz anliegt, kann es trotzdem vorkommen, dass die Ausgangsregister der Schaltung gleichzeitig geladen und gelesen werden. Dadurch kommt es zu einem unvorhersehbaren Verhalten der

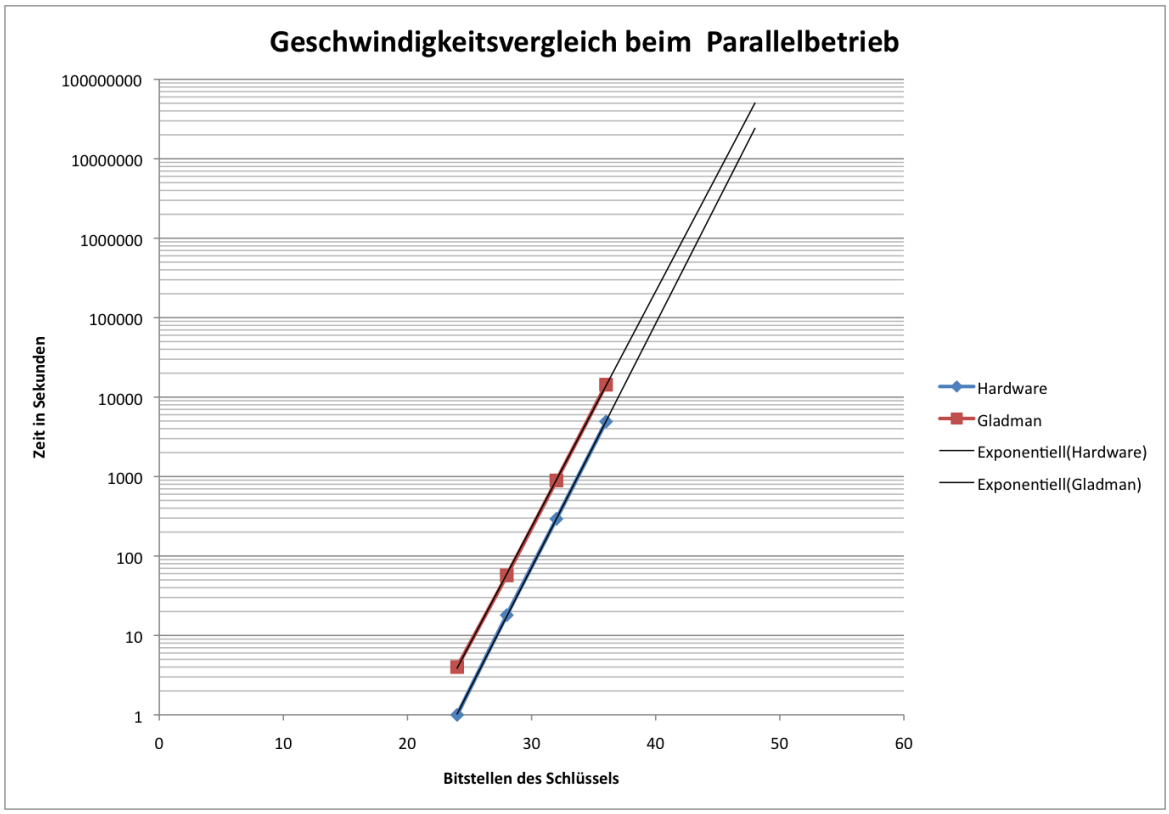

Abbildung 4.4: Logarithmische Darstellung der Laufzeiten beim Parallelbetrieb.

Schaltung. Eine Synchronisation ist zu empfehlen. Im Rahmen dieser Arbeit wurde keine explizite Synchronisation erstellt, um die Geschwindigkeit der Schaltung nicht zu verlangsamen.

5 Fazit

5.1 Verbesserungen und Ausblick zu dieser Arbeit

Nachfolgend werden noch Vorschläge aufgeführt, welche für diese Arbeit ebenfalls interessante Aspekte enthalten.

5.1.1 Verbesserungen für die digitale Hardware

Für weitere Untersuchungen ist es wünschenswert, wenn der Geheimtext und der Schlüssel beziehungsweise Teile des Schlüssels über eine Tastatur eingegeben werden können. Die Ergebnisausgabe kann außerdem auf einem Monitor ausgegeben werden. Die benötigten Hardwarekomponenten stehen auf dem Entwicklungsboard zur Verfügung. Die Ein- und Ausgabe kann über den PowerPC laufen.

5.1.2 Kryptanalyse mit einem Grafikprozessor

Die Entwicklung von Grafikkarten hat in den letzten Jahren große Fortschritte gemacht. Aufgrund der hohen Leistungsanforderungen von Computerspielen werden die Grafikprozessoren immer höher getaktet und die Architektur dahingehend verändert, dass eine parallele Verarbeitung gefördert wird. Aufgrund dessen gibt es immer mehr Bestrebungen, wissenschaftliche Berechnungen von Grafikprozessoren durchführen zu lassen. Hierfür hat der Grafikkartenhersteller NVIDIA die CUDA-Technik (**C**ompute **U**nified **D**evice **A**rchitecture) entwickelt. Aufgrund der speziellen Architektur des Grafikprozessors ist es möglich, Software auf diesem ausführen zu lassen. Alle benötigten Treiber und Informationen sind frei verfügbar [16]. Ein Geschwindigkeitsvergleich zwischen einer Softwareimplementierung, welche die Kryptanalyse auf dem Grafikprozessor ausführt, und einer Implementierung in digitaler Hardware ist durchaus interessant.

5.1.3 Realisierung einer Pipelinestruktur

Eine andere Art von Design für die Schaltung, wie sie hier verwendet wird, wäre eine Pipelinestruktur. Das bedeutet, dass zwischen jedem Bearbeitungsschritt ein Register vorhanden ist. Der Aufwand für die Steuerung erhöht sich zwar, jedoch werden durch die vielen Register die Signalpfade weiter verkürzt und der Takt kann noch weiter erhöht werden. Des weiteren sollte diese Struktur so aufgebaut sein, dass nach einer gewissen Anzahl von Takten zur Initialisierung mit jedem Takt eine Entschlüsselung am Ausgang vorhanden ist.

5.2 Abschließende Bemerkungen

Diese Arbeit kann nachweisen, dass eine Implementierung einer Kryptanalyse auf einem FPGA schneller ist als die Lösungen in Software auf einem Computer. Der Faktor liegt zwischen drei und 26, abhängig von der benutzen Software. Der Begriff Schnelligkeit bezieht sich in diesem Fall allerdings nur auf die Laufzeit, welche für den Brute-Force-Angriff benötigt wurde. Es gibt noch weitere Aspekte zu beachten. Zum einen wäre da die Entwicklungszeit, also die Zeit, die vergangen ist, um den Algorithmus umzusetzen und funktionsfähig zu bekommen. Da die Software nicht selbst entwickelt wurde, lässt sich hierfür nur schwer eine Aussage treffen. Fakt ist aber, dass das Kompilieren und Linken auf einem Computer schneller abläuft, als die Synthese und das Routing für den FPGA. Für das reine Programmieren des Algorithmus, beziehungsweise das Beschreiben der Hardware dürfte in diesem Fall die Umsetzung in digitaler Hardware schneller gewesen sein. Die Umsetzung des Algorithmus konnte relativ schnell durchgeführt werden, da der AES-Algorithmus sehr nahe an der Hardware entwickelt wurde. Die Berechnung der Entropie und das Zusammenspiel mit dem LCD-Display haben verhältnismäßig viel Zeit in Anspruch genommen. Nicht unerwäht lassen möhte ich den preislichen Aspekt. Das verwendete Entwicklungsboard gibt es auf der Herstellerseite für 1195 US$ zu kaufen (April 2010). Der verwendete Prozessor kostet ungefähr 140 €, zusätzlich wird dazu noch ein Mainboard benötigt, welches den Prozessor optimal ausnutzt. Da ein einzelner FPGA ohne Entwicklungsboard in großen Stückzahlen einen geringeren Preis haben als das Entwicklungsboard, so können diese für einen komerziellen Einsatz gut verwendet werden. Der AES-Algorithmus kann unter diesen Umständen immernoch als sicher eingestuft werden. Eine komplette Schlüsselsuche ist mit einem Brute-Force-Angriff so nicht zu machen. Allerdings hat diese Arbeit auch gezeigt, dass eine Kryptanalyse in digitaler Hardware einfach zu realisieren ist und deren Laufzeit sehr schnell ist, schneller als die schnellste Softwareimplementierung.

Literaturverzeichnis

[Beu09] BEUTELSPACHER, Albrecht: *Kryptologie*. 9. Auflage. Vieweg+Teubner, 2009

[BSW99] BEUTELSPACHER, Albrecht ; SCHWENK, Jörg ; WOLFENSTETTER, Klaus-Dieter: *Moderne Verfahren der Kryptographie*. 3. Auflage. Vieweg, 1999

[Buc01] BUCHHOLZ, Jörg J.: *MATLAB Implementation of the Advanced Encryption Standard*, 2001

[Buc08] BUCHMANN, Johannes: *Einführung in die Kryptographie*. 4.Auflage. Springer, 2008

[CP] COURTOIS, Nicolas T. ; PIEPRZYK, Josef: *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*

[Ert07] ERTEL, Wolfgang: *Angewandte Kryptographie*. 3. Auflage. Hanser Verlag, 2007

[FKL+00] FERGUSON, Niels ; KELSEY, John ; LUCKS, Stefan ; SCHNEIDER, Bruce ; STAY, Mike ; WAGNER, David ; WHITING, Doug: *Improved Cryptanalysis of Rijndael*. 2000

[FSW01] FERGUSON, Niels ; SCHROEPPL, Richard ; WHITING, Doug: *A Simple Algebraic Representation of Rijndael*. Springer, 2001

[gen] *Grundlagen der Kryptographie*. http://www.fh-wuerzburg.de/fh/fb/all/personal/interper/WSCHNELL/Grundla.pdf, Abruf: 07.09.2009

[KPP+] KAISER, Ulrich ; PAAR, Christof ; PELZL, Jan ; RAPPE, Dörte ; SCHINDLER, Werner ; WEIMERSKIRCH, André ; WOLLINGER, Thomas: *Auswahlkriterien für kryptographische Algorithmen bei Low-Cost-RFID-Systemen*. http://weimerskirch.org/papers/Weimerskirch_AuswahlkriterienRFID.PDF, Abruf: 15.09.2009

[Mar] MARTINI, Prof. Dr. N.: *Vorlesungsskript Kryptografie*. http://www.mt.haw-hamburg.de/home/martini/kryptografie_ms4.pdf, Abruf: 07.09.2009

[Osw02] OSWALD, Elisabeth: *AES - Eine Analyse der Sicherheit des Rijndael-Algorithmus*. Oktober 2002

[Pub] PUBLICATIONS, Federal Information Processing S.: *Offizielle Spezifikation von AES*. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, Abruf: 01.09.2009

[RS03] REICHARDT, Jürgen ; SCHWARZ, Bernd: *VHDL-Synthese*. 3. Auflage. Oldenbourg-Verlag, 2003

[Sch96] SCHNEIDER, Bruce: *Angewandte Kryptographie*. Addison-Wesley, 1996

[Sch09] SCHMEH, Klaus: *Kryptografie*. 4.Auflage. dpunkt.verlag, 2009

[SSP08] SWOBODA, Joachim ; SPITZ, Stephan ; PRAMATEFTAKIS, Michael: *Kryptographie und IT-Sicherheit*. Vieweg+Teubner, 2008

[Swe08] SWENSON, Christopher: *Modern Cryptanalysis*. Wiley Publishing, 2008

[Tia] TIANMA (Hrsg.): *Datenblatt vom LCD*. Tianma, http://www.xilinx.com/products/boards/ml505/datasheets/TM162VCA6_SPEC.pdf

[Wät08] WÄTJEN, Dietmar: *Kryptographie*. 2.Auflage. Spektrum, 2008

[Wil08] WILLEMS, Wolfgang: *Codierungstheorie und Kryptographie*. Birkhäuser, 2008

[Xil09a] XILINX (Hrsg.): *ML505/ML506/ML507 Evaluation Platform User Guide*. v3.1.1. Xilinx, Oktober 2009. http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf

[Xil09b] XILINX (Hrsg.): *Virtex 5 Datenblatt*. v5.2. Xilinx, Juni 2009. http://www.xilinx.com/support/documentation/data_sheets/ds202.pdf

Webliteratur

[1] *Kryptologie - Sicherheit für alle*. http://www.mathematik.de/spudema/spudema_beitraege/beitraege/galan/kryptmed.htm, Abruf: 29.09.2009

[2] *Einstieg in die Kryptologie*. http://www.regenechsen.de/phpwcms/index.php?krypto, Abruf: 22.09.2009

[3] *Grundlagen und Verfahren der Informationsübertragung*. http://www.informationsuebertragung.ch/indexAlgorithmen.html, Abruf: 16.02.2010

[4] KERCKHOFFS, Auguste: *La Cryptographie Militaire*. http://www.petitcolas.net/fabien/kerckhoffs/index.html, Abruf: 28.08.2009

[5] *Zeitungsartikel über den Schüssel vom One-Time-Pad*. http://www.nytimes.com/2001/02/20/science/20CODE.html, Abruf: 20.09.2009

[6] *Originaler Text der RC4 Veröffentlichung*. http://groups.google.com/group/comp.security.misc/msg/10a300c9d21afca0, Abruf: 02.10.2009

[7] *Informationen zur DES Challenge III*. http://www.rsa.com/rsalabs/node.asp?id=2108, Abruf: 22.01.2010

[8] *The MD5 Message-Digest Algorithm*. http://tools.ietf.org/html/rfc1321, Abruf: 02.10.2009

[9] BUCHHOLZ, Jörg J.: *AES Toolbox für Matlab*. http://www.mathworks.com/matlabcentral/fileexchange/1190, Abruf: 25.11.2009

[10] KRILL, Benjamin: *Quelle des LCD-Codes*. http://un.codiert.org/2008/08/first-fun-with-the-v5osdk-aka-ml505-fpga/, Abruf: 03.02.2010

[11] *Binär zu BCD Algorithmus*. http://www.engr.udayton.edu/faculty/jloomis/ece314/notes/devices/binary_to_BCD/bin_to_BCD.html, Abruf: 01.03.2010

[12] *Leistungsübersicht von Programmsprachen*. http://shootout.alioth.debian.org/u32q/code-used-time-used-shapes.php, Abruf: 22.03.2010

[13] *Programmcode über AES bei codeplanet*. http://www.codeplanet.eu/tutorials/cpp/3-cpp/51-advanced-encryption-standard.html, Abruf: 22.03.2010

[14] *Homepage von Cryptool*. http://www.cryptool.de/, Abruf: 25.03.2010

[15] *AES-Webseite von Brian Gladman*. http://gladman.plushost.co.uk/oldsite/AES/index.php, Abruf: 25.03.2010

[16] *NVIDIA Cuda Homepage*. http://www.nvidia.de/object/cuda_home_new_de.html, Abruf: 21.04.2010

[17] *Kryptographische Protokolle*. http://www.staff.uni-mainz.de/pommeren/DSVorlesung96/Protokolle.html, Abruf: 26.08.2009

[18] COURTOIS, Nicolas T.: *Eine umfangreiche Materialsammlung zu AES*. http://www.cryptosystem.net/aes/, Abruf: 01.09.2009

[19] *Eine Einführung in die Kryptographie*. http://www.tu-chemnitz.de/urz/lehre/rs/rs02/krypto/terms.htm, Abruf: 07.09.2009

[20] *Kryptographie – Chancen und Risiken*. http://www.cdc.informatik.tu-darmstadt.de/reports/TR/TI-03-06.thema_Forschung.pdf, Abruf: 07.09.2009

[21] *Public Key Kryptanalyse Skript*. http://www.informatik.tu-darmstadt.de/KP/lehre/ws0506/vl/skript/skript13.pdf, Abruf: 16.09.2009

[22] *Archiv von Kryptographischen Werken*. http://eprint.iacr.org/index.html, Abruf: 16.09.2009

[23] *Blog zu Kryptologie*. http://www.mitternachtshacking.de/blog/, Abruf: 16.09.2009

[24] *Kryptanalyse von Crypto-1 / Mifare*. http://www.cs.virginia.edu/~kn5f/Mifare.Cryptanalysis.htm, Abruf: 15.09.2009

[25] *Fachbericht Kryptologie*. http://andreas-romeyke.de/Projekt1/fachbericht_komplett_html/fachbericht_komplett.html, Abruf: 22.09.2009

[26] *Geschichte der Kryptographie*. http://krypto.informatik.fh-augsburg.de/geschichte.htm, Abruf: 22.09.2009

Glossar

ASCII **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange ist eine sieben Bit Zeichenkodierung aus dem Jahr 1963, welche die Grundlage weiterer Zeichenkodierungen bildet. Durch diese Kodierung lassen sich die lateinischen Buchstaben, arabischen Ziffern sowie einige Satz- und Steuerzeichen darstellen.

BCD **B**inary **C**oded **D**ecimal ist ein Code zur Darstellung von Dezimalzahlen in einer binären Repräsentation. Für jede Zahl werden vier Bit benötigt. Der Unterschied zu der binären Zahl ist, dass beim BCD-Code jede Stelle einer Zahl einzeln codiert ist und nicht die komplette Zahl.

Byte Ein Byte ist eine Speichereinheit. Sie umfasst acht Bits, welche jeweils den Zustand Null oder Eins annehmen können.

DIP Die Abkürzung steht für **d**ual **i**n-line **p**ackage und kennzeichnet eine Bauform, bei der die Anschlüsse in zwei parallel liegenden Reihen angeordnet sind.

FPGA Ein **F**ield **P**rogrammable **G**ate **A**rray ist ein programmierbare integrierte Schaltung, welche im Kern aus mehreren Flipflops und LUTs besteht.

HTTPS **H**yper **T**ext **T**ransfer **P**rotokoll **S**ecure, ein Protokoll zum verschlüsselten Austausch von Daten über as Internet, das ohne eine zusätzliche Software verwendet werden kann.

IEEE Das **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers ist der weltweit größte Verband von Ingenieuren der Elektrotechnik und Informatik. Der Verband veröffentlicht Fachzeitschriften, organisiert Fachkonferenzen und bildet Gremien für Standardisierungen aus dem Bereich der Informations- und Elektrotechnik.

JTAG **J**oint **T**est **A**ction **G**roup ist der Name für den IEEE-Standard 1149.1, welcher verschiedene Verfahren zum Debuggen und Testen von Hardwarekomponenten beschreibt, die schon in einer Schaltung verbaut sind. Eine integrierte Schaltung wird im Normalfall nicht beeinflusst, kann aber über ein Schieberegister, welches die JTAG-Schnittstelle realisiert, beeinflusst werden.

LCD Ein **L**iquid **C**rystal **D**isplay, beziehungsweise eine Flüssigkristallanzeige ist ein Bildschirm bestehend aus mehreren Segmenten mit Flüssigkristallen. Durch eine elektrische Spannung können diese Kristalle die Durchlässigkeit von einer Hintergrundbeleuchtung beeinflussen.

LED Eine **L**ight **E**mitting **D**iode oder zu deutsch Leuchtdiode oder Lumineszenz-Diode ist ein elektronisches Bauteil, welches den durchfließenden Strom in Licht umwandelt.

LUT Eine **L**ook**u**p-**T**able ist eine einfache Tabelle, welche einem Eingangswert einen Ausgangswert zuweist. So können zum Beispiel komplexe Berechnungen vorab berechnet werden und das Ergebnis von jedem möglichen Eingangswert wird in einer Tabelle erfasst.

Moore-Automat Bei einem Moore-Automaten ist das Ausgangssignal nur vom jeweiligen Zustand des Automaten abhängig. Die Eingangssignale wirken sich nur auf die Zustände des Automaten aus, nicht aber auf den Ausgang des Automaten.

Nibble Vier Bits zusammengefasst ergeben ein Nibble. In hexadezimaler Schreibweise kann ein Nibble leicht durch ein Zeichen von 0 bis F geschrieben werden. Ein Byte besteht aus zwei Nibble.

PGP Pretty Good Privacy ist ein populäres Computerprogramm, welches für eine digitale Unterschrift von Dateien benutzt werden kann. Es wurde von Phil Zimmermann entwickelt und die erste Version erschien im Jahr 1991.

Q-Format Das Q-Format ist eine Darstellung von Gleitkommazahlen für Festkommaplattformen. Hierbei wird angegeben, wie viele Bits Vor- und Nachkommastellen ausmachen. Nachkommastellen werden mit 2^{-x} berechnet, wobei x für das jeweilige Bit steht. Ein Beispiel: Q7 gibt an, dass sieben Bits ausschliesslich als Nachkommastellen verwendet werden. Zusätzlich ist immer ein Bit als Vorzeichenbit vereinbart. Die gesamte Vektorlänge beträgt somit acht Bit.

ROM **R**ead **O**nly **M**emory). Elektronischer Speicher, von dem nur gelesen, aber nicht neu geschrieben werden kann.

S-Box Eine S-Box (englisch: substitution box) ist eine Matrix, bei der Bits des Schlüssels oder des Klartextes durch andere Bits ersetzt werden. Bei einer statischen S-Box ist der Inhalt der Matrix fest vorgegeben. Bei einer dynamischen S-Box wird der Inhalt der Matrix in Abhängigkeit von dem Schlüssel generiert.

speed grade Der speed grade ist ein Maß für die Leistungsstärke des FPGAs. Bei der Virtex-5-Familie existieren die speed grades -1, -2, und -3, wobei der letzte speed grade der höchsten Leistung entspricht. [Xil09b]

TAN **T**rans**a**ktions**n**ummer, ein Einmalpasswort, welches üblicherweise aus einer Ziffernfolge besteht.

Thread Ein Thread ist ein Teil eines Prozesses, welcher Handlungsabläufe parallel neben dem Prozess realisieren kann.

Usenet Unix **Use**r **Net**work, ein elektronisches, weltweites Netzwerk, welches als Diskussionsplattform und zum Austausch von Nachrichten genutzt wird.

VHDL **V**ery **H**igh Speed Integrated Circuit Hardware **D**escription **L**anguage ist eine Hardwarebeschreibungssprache mit der es möglich ist, digitale Schaltungen zu entwerfen.

WEP **W**ired **E**quivalent **P**rivacy ist ein ehemaliger Standard für das Verschlüsseln eines drahtlosen Netzwerkes(WLAN).

WPA **W**i-Fi **P**rotected **A**ccess ist ein Standardprotokoll für die Verschlüsselung eines drahtlosen Netzwerkes.

XOR Ein mathematisches Verfahren der Digitaltechnik. Es gilt:
$0 \oplus 0 = 0$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$
$0 \oplus 1 = 1$

A Tabelle Vignère Quadrat

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

Tabelle A.1: Das Vignère-Quadrat

B AES S-Box

		unteres Nibble														
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
oberes Nibble	**0**	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Tabelle B.1: Die S-Box von AES

C Quellcode Multiplikator

```
1  ----------------------------------------------------------
2  Multiplikator fuer InvMixColumns
3  ----------------------------------------------------------
4
5  --! ------------------------------------------------------
6  --!  OPCODE = Steuersignal
7  --!           = 00 => Multiplikation mit 0E
8  --!           = 01 => Multiplikation mit 0B
9  --!           = 10 => Multiplikation mit 0D
10 --!           = 11 => Multiplikation mit 09
11 --! ------------------------------------------------------
12
13 entity multiplikator_IMC is
14         Port ( IN_0 : in  bit_vector (7 downto 0);
                      --! Eingangsbyte
                   OPCODE : in bit_vector (1 downto 0);    --!
                      Operationscode
16                 OUT_0 : out  bit_vector (7 downto 0)); --!
                      Ausgangsbyte
   end multiplikator_IMC;
18
19 --! Fuehrt eine Matrixmultiplikation fuer die InvMixColumns-
      Funktion aus.
20 --! Das Eingangsbyte wird mehrmals mit Zwei multipliziert.
      Hierfuer wird das
21 --! Byte mehrmals nach links verschoben und gegebenenfalls
      mit 0x1B modulo gerechnet.
22 --! In Abhaengigkeit des OPCODES wird das Ergebnis dann so
      mit XOR verknuepft, dass das Ergebnis
23 --! eine Multiplikation mit dem jeweiligen Faktor ist.
24
25 architecture mult_IMC_Verhalten of multiplikator_IMC is
26 signal Z1_1, Z1_2, Z1_3 : bit_vector (7 downto 0);
```

```
28 begin
   --! Multiplikation mit dem Vielfachen von 2
30 Multiplikation: process(IN_0,Z1_1,Z1_2,Z1_3)
   begin
32         if IN_0(7) = '1' then
                   Z1_1 <= IN_0(6 downto 0) & '0' xor x"1B";
34         else
                   Z1_1 <= IN_0(6 downto 0) & '0';
36         end if;

38         if Z1_1(7) = '1' then
                   Z1_2 <= Z1_1(6 downto 0) & '0' xor x"1B";
40         else
                   Z1_2 <= Z1_1(6 downto 0) & '0';
42         end if;

44         if Z1_2(7) = '1' then
                   Z1_3 <= Z1_2(6 downto 0) & '0' xor x"1B";
46         else
                   Z1_3 <= Z1_2(6 downto 0) & '0';
48         end if;
   end process Multiplikation;
50
   --! Berechnung des Ausgangs in Abhaengigkeit von OPCODE
52 Ausgang: process(Z1_1,Z1_2,Z1_3,IN_0,OPCODE)
   begin
54         if OPCODE = "00" then
                   OUT_0 <= Z1_1 xor Z1_2 xor Z1_3;
56         elsif OPCODE = "01" then
                   OUT_0 <= IN_0 xor Z1_1 xor Z1_3;
58         elsif OPCODE = "10" then
                   OUT_0 <= IN_0 xor Z1_2 xor Z1_3;
60         else
                   OUT_0 <= IN_0 xor Z1_3;
62 end if;

64 end process Ausgang;

66 end mult_IMC_Verhalten;
```

D Quellcode BCD-Konverter

```
1  library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;

5  entity bin2bcd is
   generic( BITS_IN  : natural := 20;      --! Anzahl an
      Eingangsbits
7                  BITS_OUT  : natural := 28); --! Anzahl an
                      Ausgangsbits
   Port ( CLK : in  bit;   --! Takt-Eingang
9            RESET : in  bit; --! Reset-Eingang
             LADEN :  in bit; --! Laden-Eingang
11           EINGANG : in bit_vector(BITS_IN-1 downto 0);
                      --! Die Zahl als Vektor - Eingang
             AUSGANG : out bit_vector(BITS_OUT-1 downto 0));
                     --! Die BCD-Zahl - Ausgang
13 end bin2bcd;

15 --! Fuer die Umwandlung wird der "Shift and Add-3 Algorithm"
      verwendet.
   --! Hierbei wird die Zahl mit dem MSB zuerst in einen neuen
      Speicher geschoben. Vier Bits werden
17 --! zu einer Spalte zusammengefasst und es wird ueberprueft,
      ob die Zahl in der Spalte groesser als 4 ist.
   --! Falls ja, wird eine drei hinzu addiert. Wenn die
      komplette Zahl in den Speicher geschoben wurde, ist
19 --! die Umwandlung beendet.

21 architecture bin2bcd_Verhalten of bin2bcd is

23 component add3 is
   generic(ANZ_STELLEN     : natural := 5);                  --
      Anzahl der Stellen-Parameter
```

```
25 Port ( EINGANG : in  bit_vector (ANZ_STELLEN*4-1 downto 0);
     -- Zahl-Eingang
   AUSGANG : out bit_vector (ANZ_STELLEN*4-1 downto 0)); --
     Zahl-Ausgang
27 end component;


29

   signal SPEICHER, SP_ADD : bit_vector(BITS_OUT-1 downto 0);
31 signal EINGANG_TMP : bit_vector(BITS_IN-1 downto 0);
   signal SCHIEBE : bit;
33 signal Z_TEMP : std_logic_vector(5 downto 0);
   signal EN : bit;
35

   begin
37

   --! Entweder wird die Zahl geschoben, oder der addierte Wert
       wird geladen.
39 schieben: process(CLK,RESET)
   begin
41         if RESET = '0' then
                   SPEICHER <= (others => '0');
43                 EINGANG_TMP <= (others => '0');
           elsif CLK='1' and CLK'event then
45                 if EN = '1' then
                           if SCHIEBE = '1' then
47                                 EINGANG_TMP <= EINGANG_TMP(
                                       BITS_IN-2 downto 0) &
                                       '0';
                                   SPEICHER <= SPEICHER(
                                       BITS_OUT-2 downto 0) &
                                       EINGANG_TMP(BITS_IN-1);
49                         elsif SCHIEBE = '0' then
                                   SPEICHER <= SP_ADD;
51                         end if;
                   else
53                         if LADEN = '1' then
                                   EINGANG_TMP <= EINGANG;
55                                 SPEICHER <= (others => '0');
                           end if;
57                 end if;
           end if;
59 end process schieben;
```

```
61 AUSGANG <= SPEICHER;

63 --! Addiere eine Drei pro Spalte dazu.
   add3_k : add3 generic map(ANZ_STELLEN=>BITS_OUT/4)
65       port map(EINGANG=>SPEICHER, AUSGANG=>SP_ADD);

67 --! Der Prozess zaehlt ab, wie oft die Zahl noch geschoben
       werden muss.
   zaehler: process(CLK,RESET)
69 begin
          if RESET = '0' then
71                SCHIEBE <= '0';
                  Z_TEMP <= (others => '0');
73                EN <= '0';
          elsif CLK='1' and CLK'event then
75                if LADEN='1' then
                        Z_TEMP <= (others => '0');
77                      EN <= '0';
                        SCHIEBE <= '1';
79                else
                        SCHIEBE <= not SCHIEBE;
81                      Z_TEMP <= Z_TEMP + 1;
                        EN <= '1';
83                      if Z_TEMP = (BITS_IN*2) then
                                EN <= '0';
85                              Z_TEMP <= Z_TEMP;
                                SCHIEBE <= '1';
87                      end if;
                  end if;
89        end if;
   end process zaehler;
91

93 end bin2bcd_Verhalten;
```

E Datenträger

Zu dieser Diplomarbeit gehört ein Datenträger mit zusätzlichen Materialien. Der Datenträger kann bei Herrn Prof. Dr.-Ing. Fitz eingesehen werden. Sollte der Datenträger fehlen, wenden Sie sich bitte an *svenkapitza@yahoo.de*.
Dieser Datenträger enthält:

- die Diplomarbeit im pdf-Format
- alle verwendeten Quellen
- sämtliche Quelltexte
- eine ausführliche Dokumentation der vhdl-Dateien
- ... und vieles mehr.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §25(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 28. April 2010	
Ort, Datum	Unterschrift