

# Entwicklung eines autarken Chiffrierungsadapter

Moritz Beller

24. Januar 2006

In der vorliegenden Arbeit wird eine Verschlüsselungslösung auf Basis des „American Encryption Standard“ in Chiphardware entwickelt. Ziel ist neben dem Verständnis der grundlegenden mathematischen Probleme der Transfer dieser in eine für die Chipherstellung verwendbare Form, so dass schlussendlich entsprechende Prototypen gefertigt werden können.

Eine aktuelle Version ist unter <http://www.4momo.de/forschung/jufo-chip-mbeller.pdf> abrufbar.

## Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>2</b>	<b>Vorüberlegung und Spezifikation</b>	<b>2</b>
2.1	Faktensammlung . . . . .	2
2.2	Zusammenfassung . . . . .	3
2.3	Entscheidungsbegründung für AES . . . . .	3
2.4	Überblick über den grundlegenden Aufbau von AES . . . . .	4
<b>3</b>	<b>Mathematische Grundlagen</b>	<b>5</b>
3.1	Das Finite Feld . . . . .	5
3.2	Rechenoperationen in $GF(2^8)$ . . . . .	6
<b>4</b>	<b>Detailanalyse von AES</b>	<b>8</b>
4.1	Input und Output . . . . .	8
4.2	Die Rundentransformation . . . . .	9
4.3	Problemstellungen des Chipdesigns . . . . .	12
4.4	Chipentwicklung . . . . .	14
<b>A</b>	<b>Anhang: AES-Mindmap</b>	<b>16</b>
<b>B</b>	<b>Anhang: AES-Tabellen</b>	<b>16</b>
<b>C</b>	<b>Anhang: Danksagung</b>	<b>18</b>

## 1 Einleitung

*Das Informationszeitalter stellt nach der Agrargesellschaft und dem Industriezeitalter die dritte (Wirtschafts- und) Gesellschaftsform dar. Gekennzeichnet ist diese Phase von der zentralen Bedeutung von Information als Rohstoff und Ware, was erst durch die elektronische Datenverarbeitung und globale Informationsflüsse möglich wurde. Die Bedeutung der Produktion von materiellen Gütern geht stattdessen immer mehr verloren, entscheidend für die Wertschöpfung ist stattdessen die Intelligenz, die in ein Gut gesteckt wird. Der Übergang vom Industriezeitalter ins Informationszeitalter begann in den 1970er und 1980er Jahren mit der immer größeren Bedeutung kommunikativer Netzwerke.<sup>[6]</sup>*

Im ersten Jahrzehnt des dritten Jahrtausend leben wir endgültig nicht mehr im Industriezeitalter, sondern im Zeitalter der Information. In einem Zeitalter, in dem die Portabilität von Informationen einen so hohen Stellenwert wie noch nie in der Geschichte der Menschheit erlangt hat. Gleichzeitig wird viel zu selten hinterfragt, wer überhaupt die Sicherheit des Datenmaterials, der Informationen, gewährleistet. Die Kapazitäten der mobilen Informationsträger steigen zwar exponentiell, die Sicherheitsfragen aber werden nach wie vor vernachlässigt.

Wie können sensible Daten auf einem externen Massenspeicher zwar einerseits schnell zugreifbar sein, andererseits aber auch im Falle des Verlusts oder beim Diebstahl des Geräts vor den Augen Dritter geschützt gelagert werden?

Die Antwort scheint zunächst offensichtlich: Nur mittels Verschlüsselung. Da sie jedoch ausschließlich mit etlichem Fachwissen in den Chiffrierungstechniken softwareseitig möglich ist, wird sie, von Privatanwendern ohnehin, nicht genutzt. Begründungen: Zu unsicher, zu viel Aufwand, zu teuer.

Die erstrebenswerte Lösung muss möglichst sicher vor äußeren Angriffen, und möglichst autark arbeiten. Ziel der vorliegenden Arbeit ist es, eine entsprechende Lösung zu entwickeln.

Ich habe versucht, die Arbeit für einen Mathematiker wie für einen Informatiker gleichermaßen verständlich aufzubauen. Wo es mir nötig schien, habe ich grundlegende Begriffe unter Bezugnahme auf andere Literatur kurz erläutert. Die meisten Abbildungen in diesem Werk gehen auf [1] zurück. Dort finden sich auch zahlreiche programmiertechnische Details, auf die ich im Rahmen der gegebenen Seitenzahl in diesem Dokument nicht eingehen konnte. Für das Verständnis der vorliegenden Arbeit ist die Lektüre von [1] zwar nicht erforderlich, sie hilft jedoch die Struktur des Quellcodes eines Chipdesigns zu verstehen. Dabei wird auch deutlich, wie aufwändig die Realisierung selbst kleinerer Aufgaben in der „Hardware Description Language“ Verilog ist.

## 2 Vorüberlegung und Spezifikation

Im Folgenden werden verschiedene Aspekte der Informatik im Bezug auf das gestellte Verschlüsselungsproblem aufgegriffen und teilweise kurz erklärt. Dies dient als erster Grundbaustein der vorliegenden Arbeit, in dem wichtige Grundsatzentscheidungen getroffen werden.

### 2.1 Faktensammlung

- Der Begriff des „Massenspeicher“ wird in [7] wie folgt definiert:

*Als Massenspeicher werden im IT-Bereich Speichermedien bezeichnet, die dauerhaft große Mengen an Daten speichern. (...)*

*Die größte Bedeutung als Massenspeicher haben seit über zwanzig Jahren Festplatten. (...)*

*Je nach Größendefinition sind Medien wie (...) CD-ROM und DVD-ROM, (...) und Flash-Speicher-Medien wie USB-Sticks und CompactFlash-Karten ebenfalls als Massenspeicher zu bezeichnen. Ihr Vorteil ist die einfache Mobilität und Verwendbarkeit als Wechselmedium in jedem beliebigen Gerät mit entsprechendem Laufwerk, Steckplatz oder passender Schnittstelle.*

- Die heutzutage am weitesten verbreitete Verbindungsschnittstelle zwischen externen Massenspeichern und Computer ist der USB-Port.
- Eine Verschlüsselung ist zunächst als Software denkbar, die auf dem Computer abläuft und dort vom Endanwender bedient wird. Dies hat eine Reihe von Vorzügen:
  - Die Software kann ständigen Verbesserungen unterliegen.
  - Programmcode kann leicht an unterschiedliche Bedürfnisse angepasst werden.
  - Softwareentwicklung ist billig und kann mit einfachen Mitteln durchgeführt werden.

Nachteile sind:

- Software ist immer abhängig vom Betriebssystem. Erstrebenswerte plattformübergreifende Lösungen sind aufwändig.
- Der Prozess des Verschlüsseln ist sehr rechenintensiv und softwareseitige Umsetzungen müssen komplett von der CPU bearbeitet werden. Somit ist ein störungsfreies Arbeiten beim Verwenden der Verschlüsselung unter Umständen nicht gegeben.
- Software ist fehleranfällig und kann vergleichsweise leicht gehackt werden, was bei sicherheitsrelevanten Anwendungen umso riskanter ist.

Einer Reihe von Vorteilen einer Softwarelösung des Verschlüsselungsproblem stehen also eine große Zahl an gewichtigen Contra-Argumenten gegenüber. Die Schlussfolgerung, die Verschlüsselung nicht softwareseitig, sondern hardwareseitig vorzunehmen, liegt nahe:

- Da die Hardware für diesen einen speziellen Zweck (und auch nur diesen!) entwickelt wurde, vermag sie ihre Aufgabe sehr schnell zu erfüllen, mitunter besser als der generische Prozessor eines Computers selbst.
- Hardware in Form von Mikrochips ist – sobald einmal entwickelt – solide und weiter weniger fehleranfällig als Software<sup>1</sup>. Ein Angriffsversuch wird ungleich schwerer.
- Der Verschlüsselungschip ist genauso mobil wie der Informationsträger selbst. Software ist das nicht, da sie immer an eine bestimmte Architektur gebunden ist.
- Die Herausforderung und damit verbunden der Aufwand, einen Chip zu entwerfen, ist wesentlich höher als ein „einfaches Stück Software“ zu schreiben.

## 2.2 Zusammenfassung

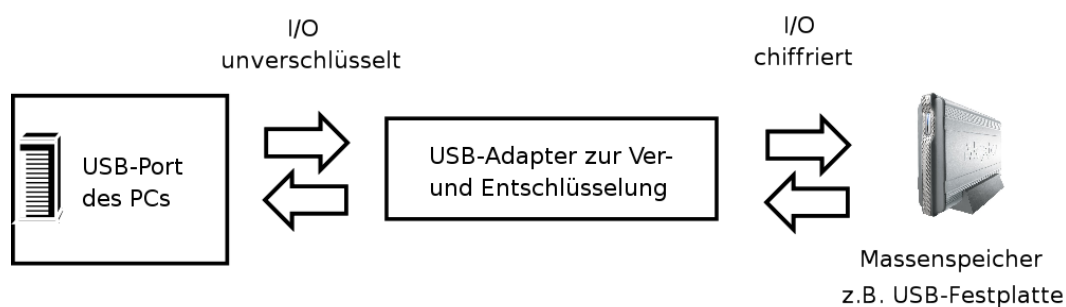


Abbildung 1: Anschluss der Chiphardware.

Die Anforderungen an die Chipentwicklung lauten also, die algorithmischen Verfahren der Verschlüsselung in einen USB-Adapter zu integrieren, der physikalisch als Bindeglied zwischen USB-Port des Rechners und dem benutzten Endgerät, also zum Beispiel der USB-Festplatte, fungiert. Dieser soll den Benutzer während des Mount-Prozesses<sup>2</sup> nach einem Passwort fragen und dann, zwecks einfacher Benutzung, in den Hintergrund treten.

## 2.3 Entscheidungsbegründung für AES

AES (Akronym für „American Encryption Standard“) ist ein symmetrisches Verschlüsselungssystem, das seinen überholten Vorgänger DES („Data Encryption Standard“) sowie dessen Derivat

<sup>1</sup>Nicht umsonst kommen Chips und nicht ein Mikrocontroller für die kritischen Sicherheitssysteme „ABS“, „ESP“ und „ASR“ moderner Autos zum Einsatz.

<sup>2</sup>Als „Mounten“ bezeichnet man das Einbinden eines Datenträgers in das System.

3DES im Jahr 2000 offiziell ablöste.

- Bei symmetrischen Verschlüsselungsverfahren wird derselbe Key für Ver- und Entschlüsselung benutzt.
- AES ist frei, dh. seine Benutzung wird nicht durch Patente restriktiert, wie etwa bei „IDEA“ (International Data Encryption Algorithm).
- Im Gegensatz zu anderen Verschlüsselungsverfahren wie RSA sind noch keine praktischen Schwächen von AES bekannt. Ein theoretischer Schwachpunkt AES' ist aber seine sehr algebraische Struktur, die es erlaubt, ihn als eine einzige komplexe Formel zu schreiben, wie in [5] gezeigt. Solange niemand ein Verfahren findet, mit dem man effizient 8000 quadratische Gleichungen mit 1600 Variablen lösen kann, muss man sich bei diesem Ansatz über die Sicherheit keine weiteren Gedanken machen. Shamir et al. entwickelten zwar in einem auf der „Eurocrypt 2000“ veröffentlichten Paper einen entsprechenden Algorithmus, dieser war aber in der Praxis nicht zu gebrauchen (siehe [2]).
- AES verzichtet auf platz- und taktintensive Divisionen, im Gegensatz zu RSA, das auf der Annahme beruht, dass es zwar recht leicht ist, Primzahlen zu finden, aber sehr schwer, sie zu Primfaktoren zu zerlegen.  
Daher ist AES für die Chipherstellung geeignet.
- Mit variablen Cipher- und Blocklängen ist AES sehr flexibel einzusetzen. Der Autor hat sich für 256-Bit entschieden, der besten Verschlüsselungstiefe, die AES bieten kann.

## 2.4 Überblick über den grundlegenden Aufbau von AES

Obwohl in Kapitel 4 noch ausführlicher darauf eingegangen wird, soll hier zunächst ein grober Überblick über den Aufbau von AES geboten werden.

### 2.4.1 Rudimentärer Ablauf (abstrakt)

AES gehört zu den Block-orientierten Verschlüsselungsalgorithmen. Es arbeitet daher immer nur mit vollständigen Blöcken einer wohldefinierten Anzahl an Bytes. (Dies hat einige Implikationen auf zusätzliche Gestaltungsmerkmale des Chips, siehe hierzu auch.)

Ein vollständiger AES-Verschlüsselungsprozess besteht stets aus  $10 \leq c \leq 14$  Runden, wobei  $c$  abhängig ist von der gewünschten Breite der Blöcke und der Breite des Verschlüsselungskeys. Die Breite der Blöcke, welche in Bits angegeben wird, bestimmt wie viele Daten mit einem Verschlüsselungsprozess bearbeitet werden können. Für Anwendungen mit geringem Datendurchsatz ist es sinnvoller, kleinere Blockgrößen von zum Beispiel 128 Bit zu wählen. Die maximale Größe beträgt 256 Bit.

Jede Runde ist definiert durch das lineare Abarbeiten einer sequentiellen Folge von Unterschritten. Nachdem alle Schritte in der vorgegebenen Reihenfolge ausgeführt sind und die letzte Runde beendet worden ist, wird das Ergebnis des Algorithmus ausgegeben und ein neuer Block von Daten kann bearbeitet werden.

Wie wir später noch sehen werden, entspricht die Entschlüsselung genau dem umgekehrten Ausführen der Reihenfolge beim Verschlüsseln.

### 2.4.2 Ablaufbeispiel

Ein Datenblock sei ein Feld aus  $4 \times 4$  Bytes beliebigen Inhaltes, dh. wir verwenden eine 128-Bit-Verschlüsselung. Gemäß der AES-Spezifikation ist es aber nicht notwendig, dass der Key diesselben Eigenschaften erfüllt wie der Ciphystate. Dieser soll mit dem gespeicherten „Passwort“, der Keyphrase, verschlüsselt werden. Dazu werden zunächst die Subalgorithmen **SubBytes**, dann **ShiftRows**, **MixColumns** und schließlich der **AddRoundKey**-Schritt ausgeführt. Nach  $c = 10$  wird per definitionem beim letzten Rundendurchlauf der **MixColumns**-Schritt ausgelassen.

## 3 Mathematische Grundlagen

### 3.1 Das Finite Feld

Sämtliche mathematische Operationen in AES werden in einem finiten Feld, genauer in einem Galois-Feld  $GF(2^8)$  durchgeführt. Die Arithmetik dafür ist eine andere als Standardarithmetik, da alle Ergebnisse innerhalb des Definitionsbereichs  $2^8$  liegen müssen. Obwohl es also nur eine bestimmte endliche Anzahl an Ziffern innerhalb des finiten Feldes gibt, existieren unendliche viele finite Felder.

**Definition:** Ein Byte bestehe aus 8 Bit. Also kann ein Byte  $b$  polynomial geschrieben werden als

$$b_7 \cdot x^7 + b_6 \cdot x^6 + \dots + b_1 \cdot x + b_0, \quad (1)$$

wobei  $b_z \in \{0, 1\}$ .

10011001 entspricht gemäß (1) in Polynomschreibweise  $x^7 + x^4 + x^3 + 1$ .

**Definition:** Um nicht in Verwechslungsgefahr zu geraten, definieren wir folgende Symbole:

⊕ Additionsoperator, auch XOR-Operator; in der Informatik oft  $\wedge$

⊗ Multiplikationsoperator (Den  $\oslash$  Divisionsoperator verwenden wir nur einamtl.)

## 3.2 Rechenoperationen in $GF(2^8)$

### 3.2.1 Addition

Die Addition zweier Summanden  $b$  und  $c$  ist gegeben durch

$$b \cdot x^z + c \cdot x^z = (b + c) \bmod 2 \cdot x^z. \quad (2)$$

Aus (1) und (2) folgt

$$\begin{aligned} 1 + 1 &= & (3) \\ 1 \cdot x^0 + 1 \cdot x^0 &= \\ (1 + 1) \bmod 2 \cdot x^0 &= \\ 0 \cdot x^0 &= 0 \end{aligned}$$

Dieses Ergebnis darf nicht verwirren, ist doch  $1 \oplus 1 = 0$ . Bei der Addition in  $GF(2^8)$  handelt es sich also nach Anwenden der Vereinfachung um nichts anderes als eine XOR-Verknüpfung der beiden Summanden.

$$x^7 + x^4 + x^3 + x^2 + 1 \oplus x^6 + x^3 + 1 = x^7 + x^6 + x^4 + x^2$$

$$10011101 \oplus 01001001 = 11010100$$

Mit normaler Algebra lautete das Ergebnis  $x^7 + x^6 + x^4 + 2x^3 + x^2 + 2$ , was aber natürlich nicht mehr in  $GF$  läge. Dies darf gemäß der Definition in Abschnitt 3.1 aber nicht sein.

Das neutrale Element der Addition ist 0.

Wie das Beispiel zeigt, ist sie mit binärer Logik, wie sie bei der Chipherstellung zur Verfügung steht, leicht umzusetzen.

### 3.2.2 Subtraktion

Da die XOR-Verknüpfung ihre eigene Umkehrung darstellt, ist in  $GF(256)$  die Addition gleich der Subtraktion. Jedes Element ist sein eigenes additives Invers.

$$1 \oplus 1 \oplus 1 = 1 \oplus (1 \oplus 1)$$

lässt sich nach (3) umformen in

$$1 \oplus 0 = 1.$$

QED.

### 3.2.3 Multiplikation

In Polynomschreibweise ist die Multiplikation das Produkt der beiden Polynome  $p_1(x)$  und  $p_2(x)$  modulo ein irreduzibles Polynom achten Grades. Dies ist notwendig, um das resultierende Ergebnisse in  $GF$  halten zu können. Das irreduzible Polynom für AES lautet  $x^8 + x^4 + x^3 + x + 1$  und wird mit  $m(x)$  bezeichnet.<sup>3</sup> Irreduzibel bedeutet, dass das gewählte Polynom keine Teiler außer 1 und sich selbst besitzt.

$$\begin{aligned} (x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) &= x^{13} + x^{11} + x^9 + x^8 + x^7 + \\ & x^7 + x^5 + x^3 + x^2 + x + \\ & x^6 + x^4 + x^2 + x + 1 \end{aligned}$$

wird nach Vereinfachung zu

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1.$$

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \bmod m(x) = x^7 + x^6 + 1 \quad (4)$$

Wird (4) hexadezimal geschrieben, so ergibt sich daraus  $57 \otimes 83 = C1$ .

Es ist evident, dass das neutrale Element dieser Multiplikation 1 ist:

$$\begin{aligned} (x^7 + x^2)(1) &= x^7 + x^2 \\ x^7 + x^2 \bmod m(x) &= x^7 + x^2 \end{aligned}$$

$b(x)$  sei ein Polynom höchstens siebten Grades. Gemäß des erweiterten euklidischen Algorithmus kann man zwei Polynome  $p_1(x)$  und  $p_2(x)$  berechnen, für die gilt

$$b(x)p_1(x) + m(x)p_2(x) = 1. \quad (5)$$

Aus (5) folgt

$$\begin{aligned} p_1(x) \otimes b(x) \bmod m(x) &= 1 \\ b(x) &= \frac{1}{p_1(x) \otimes b(x) \bmod m(x)}. \end{aligned}$$

QED.

Wenn man die Multiplikation und Addition wie oben beschrieben definiert, so folgt daraus, dass alle 256 möglichen Werte die Struktur unseres  $GF$  haben.

<sup>3</sup>Die Schreibweise „von x“ soll verdeutlichen, dass es sich hierbei um die Polynomdarstellung handelt.

Für die Multiplikation existiert keine einfache Operation auf Bit-Ebene wie noch bei der Addition, so dass sie in Chiphardware sehr schwer umzusetzen ist.

## 4 Detailanalyse von AES

### 4.1 Input und Output

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Abbildung 2: `cipherstate` mit  $Nb = 6$  und `keyphrase` mit  $Nk = 4$ . Bild entnommen aus [1].

AES arbeitet grundsätzlich mit nur zwei Variablen: Dem `cipherstate`  $a$  und der `keyphrase`  $k$ . `cipherstate` wird zunächst der zu verschlüsselnde Datenblock im Plaintext genannt, das heißt in der Reinform, in der er übergeben wird. `keyphrase` ist der Cipher, also der Schlüssel, mit dem der `cipherstate` verschlüsselt wird.

`cipherstate` und `keyphrase` werden als Arrays mit vier Reihen betrachtet. Die Anzahl der Spalten  $Nb$  und  $Nk$  ist abhängig von der gewählten Blockgröße 4 (128 Bit), 6 (192 Bit) oder 8 (256 Bit). Jede Zelle repräsentiert ein Byte aus 8 Bit.

Der externe Input und Output kann natürlich nicht als solches Array erfolgen, zumal es sich bei den meisten Übertragungswegen um serielle Schnittstellen handelt. Die eindimensionale Eingabe des Inputs  $I$  wird dementsprechend bytewise in das Array übertragen nach der Formel

$$a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{0,1}, a_{1,1}, a_{2,1}, a_{3,1}, \dots \quad (6)$$

Formel (6) gilt analog natürlich auch für `keyphrase`. In den noch sehr rudimentären Programmiersprachen der Chipentwicklung sind multidimensionale Arrays oft nicht umsetzbar.  $n$  sei der eindimensionale Index eines Bytes innerhalb des Blocks. Die zweidimensionale Repräsentation laute  $a_{i,j}$ . Die folgenden Formeln werden dann in dem Programmtext zum Chipdesign zum Umrechnen verwendet.

$$i = n \bmod 4 \quad (7)$$

$$j = n/4 \quad (8)$$

$$n = 4j + i \quad (9)$$

Für den Output des Programms werden natürlich dieselben Transformationen bezüglich des `cipherstate` ausgeführt, lediglich in umgekehrter Reihenfolge von Array in eindimensionale Folge.

## 4.2 Die Rundentransformation

Wie bereits in Abschnitt 2.4.2 angesprochen, besteht der eigentlich AES-Prozess aus vier Transformationsschritten, die  $c$  Runden oft ausgeführt werden. Jeder der vier Transformationsschritte ändert den `cipherstate` mithilfe verschiedener mathematischer Operationen, die im Folgenden erläutert werden. Als Ergebnis, also wenn der Algorithmus terminiert, wird die oben beschriebene Outputoperation durchgeführt und `cipherstate` in entsprechender Reihenfolge sequentiell ausgegeben. Bestimmte Variablen innerhalb der Transformationsschritte nehmen für unterschiedliche  $Nk$  und  $Nb$  divergente Werte an. Für die umzusetzende Verschlüsselung hat sich der Autor für  $Nk = Nb = 8$  entschieden, so dass die maximale Verschlüsselungstiefe von 256 Bits ausgenutzt wird. Die anderen Werte können in [3] nachgeschlagen werden.

### 4.2.1 SubBytes-Transformation

1. Berechne das multiplikative Invers (siehe 3.2.3 auf Seite 7). **Definition:** 00 bleibe dabei 00.
2. Wende anschließend eine affine Transformation über die Bits des Bytes an, welche definiert ist durch:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Das nähere Berechnungsverfahren der affinen Transformation, das eine Addition der Komponenten in einer Reihe vorsieht, soll an dieser Stelle nicht weiter erklärt werden (für das Verfahren siehe aber z.B. [4] S. 256ff). Wesentlich ist, dass die hier gezeigt affine Transformation für einen Chip bei weitem zu langsam wäre. Eine äquivalente, aber weitaus geeignetere Methode wird in Abschnitt 4.3 entwickelt.

### 4.2.2 ShiftRows-Transformation

Im Anschluss wird ein zyklischer Shift über alle  $a_{\{1,2,3\},j}$  des `cipherstate` durchgeführt. Der Vektor  $\vec{v}$  für  $a_{1,j}$  lautet  $(0, 1)$ , für  $a_{2,j}$   $(0, 3)$  und für  $a_{3,j}$   $(0, 4)$ . Das heißt beispielsweise, dass das Byte,

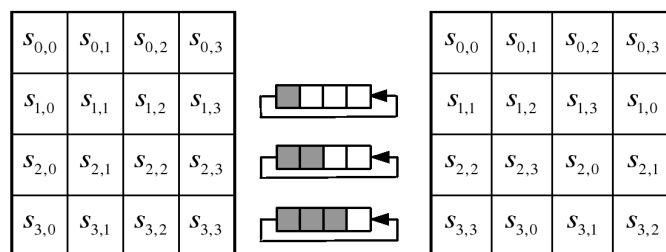


Abbildung 3: Visualisierung der ShiftRows-Prozedur am Beispiel eines  $4 \times 4$  Byte cipherstate.

welches an der Stelle  $(1, 0)$  sitzt, zur Position  $(1, 1)$  wechselt. Zyklisch deswegen, weil  $(1, 3)$  zu  $(1, 0)$  wird. Es findet also kein Datenverlust oder ähnliches statt, sondern lediglich eine Umverteilung der Bytes innerhalb des cipherstate (siehe zur besseren graphischen Visualisierung auch Abbildung 2 auf Seite 8).

Die Entschlüsselung wird durch den Gegenvektor gegeben,  $\bar{v}$  ändert sich also nicht.

#### 4.2.3 MixColumns-Transformation

Die MixColumn-Transformation arbeitet spaltenweise über den cipherstate, indem sie jeweils vier Bytes aus den Spalten  $a_i, 0$  bis  $a_i, 3$  ( $i = Nb$ ) mit einem festen Polynom  $e(x) = 03 x^3 + 01 x^2 + 01 x + 02$  multipliziert.

$$\begin{bmatrix} a_{0neu} \\ a_{1neu} \\ a_{2neu} \\ a_{3neu} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Das heißt, konsequenterweise muss für die Entschlüsselung mittels  $g(x)$  gemäß Abschnitt 3.2.3 gelten:

$$\begin{aligned} e(x) \otimes g(x) &= 1 & (10) \\ 03 x^3 + 01 x^2 + 01 x + 02 \otimes g(x) &= 1 \\ \Rightarrow g(x) &= 0B x^3 + 0D x^2 + 00 x + 0E \end{aligned}$$

#### 4.2.4 AddRoundKey-Transformation

Die anspruchsvollste Transformation stellt das Vermengen des aktuellen Rundenschlüssel mit dem cipherstate dar. Im ursprünglichen Sinne ist dies die eigentliche Ver- bzw. Entschlüsselung. Das Addieren erfolgt gemäß den mathematischen Regeln in Gleichung (2) über eine XOR-Verknüpfung.

Schwierig ist hingegen das Bestimmen des aktuellen Runden-Schlüssel, denn es gibt für jede Runde einen aus dem ursprünglichen Key berechneten Runden-Schlüssel. Prinzipiell existieren zwei Möglichkeiten für die Handhabung der Rundenschlüssel: Zum einen kann man den jeweiligen, momentan aktuellen Rundenschlüssel auf Basis des vorhergehenden „on the fly“ neu berechnen; zum anderen alle Rundenschlüssel schon beim Initialisieren des Schlüssels vorausberechnen und in ein Register `keyring` abspeichern. Aus diesem kann mittels entsprechender Rundenindizes dann der jeweils geeignete Rundenkey bezogen werden. In Betracht der Tatsache, dass die Verschlüssel genau umgekehrt stattfindet, also „von hinten“ beginnt, ist dies sicherlich die sinnvollste Lösung, da bei der Entschlüsselung für die erste Runde der Rundenkey der letzten Runde der Verschlüsselung benötigt wird. Diesen kann man aber nur dann berechnen, wenn man bereits alle vorherigen ermittelt hat.

Der  $n$ -te Rundenkey wird wie folgt berechnet:

1. Teile den  $(n - 1)$ -ten Rundenkey in 4 Byte breite Mengen auf.<sup>4</sup>
2. Jede dieser Mengen enthält Bytes in der Form  $M = \{a, b, c, d\}$ . Vertausche die Reihenfolge dieser, so dass  $M' = \{b, c, d, a\}$ . Diese Prozedur soll `RotByte` genannt werden.
3. Die ersten  $Nk$  Wörter (das heißt die ersten  $4 \cdot Nk$  Bytes) des neuen Rundenkey werden mit denen des vorherigen Keys gefüllt.

Für die folgenden Wörter  $W_i$  gilt:

$$W_i = W_{i-1} \oplus W_{i-Nk} \quad (11)$$

Wenn allerdings  $i$  ein Vielfaches von  $Nk$  ist, so ist statt Formel (11)

$$W_i = \text{Sbox}(\text{RotByte}(W_{i-1})) \oplus \text{rcon}_{\binom{i}{Nk}} \quad (12)$$

anzuwenden.

4. In der vorigen Gleichung ist noch die Variable `rcon` unbekannt. Bei `rcon` handelt es sich um ein fixes, in AES fest vorgegebenes Array. Die ersten Bytes der vier bytigen RoundConstants lauten: 00, 01, 02, 04, 08, 10, 20, 40, 80, 1b, 36, 6c, d8, ab, 4d Die restlichen drei Bytes sind immer 00. Berechnet werden kann der Wert `rcon[i]` über

$$\text{rcon}_i = x^{(i-1)} = x \otimes \text{rcon}_{i-1} \quad (13)$$

`rcon`<sub>1</sub> ist festgesetzt als 1. Alle anderen `rcon` <sub>$i$</sub>  leiten sich entsprechend daraus ab.

---

<sup>4</sup>Daemen und Rijmen verwenden hierfür im englischen Original den Ausdruck „word“.

## 4.3 Problemstellungen des Chipdesigns

### 4.3.1 Vereinfachung des SubBytes-Schrittes

Die Vereinfachung des SubBytes-Schrittes ist durch folgende Überlegung möglich: Der SubBytes-Schritt gemäß der AES-Spezifikation (Erklärung siehe 4.2.1) ist eine eindeutig bestimmte Funktion  $f(x)$ , die für jedes  $x$  genau einen Wert  $y$  liefert.

Damit lässt sich also bereits im Voraus eine Wertetabelle anlegen, die man fest in das Programm „verdrahten kann.“ Mit wenigen geschickten Formulierungen innerhalb des Quellcodes kann man dann erzwingen, dass keine platzfressenden Register<sup>5</sup> plaziert werden, sondern Logikgatter verwendet werden. Die dafür nötige so genannte „S-Box“<sup>6</sup> ist im Anhang B abgedruckt.

### 4.3.2 Vereinfachung des MixColumn-Schrittes

Neben den bereits durch die Anpassung der Multiplikation für das Chipdesign erreichten Verbesserungen, können hier beim Verschlüsseln pro Reihe und Byte zwei Multiplikationen gespart werden: Wie in 3.2.3 gezeigt, ist 01 (bzw. 1) das neutrale Element der Multiplikation. Statt Platz auf der Chipoberfläche für Logikelemente zu vergeuden, deren Ergebnis schon im Voraus klar ist, kann hier direkt der ursprüngliche Bytewert zum Addieren benutzt werden.

### 4.3.3 Vereinfachung der Multiplikation

Wie bereits in Abschnitt 3.2.3 erläutert, unterliegt die Multiplikation wie sie im AES-Entwurf ursprünglich vorgesehen war einem sehr aufwändigen Verfahren, das für das Chipdesign unbrauchbar ist. Wir wollen daher im Folgenden an einer schnelleren und platzsparenderen Lösung für das Chipdesign erarbeiten und bedienen uns eines eigentlich schon vergessenen Hilfsmittel: Der Logarithmustafel.

Mit einigen Werten innerhalb von AES' Galois Feld erhält man durch fortgesetzte Multiplizierung alle möglichen Werte (außer 0). Nach genau 255 Exponentierungen gelangt man zur ursprünglichen Zahl zurück. Dazu wurde im Anhang die entsprechende Tabelle B mit dem Generator 05 erstellt. Die Auswahl erfolgte willkürlich (da alle natürlich letzten Endes natürlich zum selben Ergebnis führen), insgesamt gibt es 128 mögliche Generatoren. Der  $n$ -te Tabelleneintrag wurde wie folgt angelegt:

$$a_n = (05)^n$$

Nach  $n \bmod 16 = 0$  wurde jeweils ein Zeilenumbruch durchgeführt. So kann man schnell die benötigten Werte ablesen.

In der zugehörigen Logarithmustafel wird ersichtlich, wie oft wir 05 mit sich selbst multiplizieren müssen, um eine Zahl  $x \in \{1, 2, 3, \dots, 254, 255\}$  zu erhalten. Es lässt sich schnell überprüfen, ob die

<sup>5</sup>Flipflops; teurer Datenspeicher wie etwa im Ram

<sup>6</sup>Substitutionsbox

Tabellen zusammengehören, denn es muss gelten

$$\log_{05} 05 = 1. \quad (14)$$

Mit der Logarithmstabelle lässt sich wie folgt arbeiten: In der Beschriftung der Reihen auf der linken Seite ist die erste Ziffer der hexadezimalen Zahl zu suchen. Die „Einer“ findet sich in der horizontalen Beschriftung der Spalten. Möchte man also wissen, ob Gleichung (14) korrekt ist (bzw. vielmehr ob die Tafel auch wirklich stimmt), so sucht man zuerst den Wert 00 (wegen 00 auf der vertikalen Beschriftung. Dann, nachdem die Reihe feststeht, in welcher sich der gewünschte Logarithmus befindet, muss man nur noch die richtige Spalte auswählen: 05.

Die beiden Tafeln können wir nun dazu verwenden, um Multiplikationen der Form  $x \otimes y$  durchzuführen.

1. Gehe zur Logarithmstabelle und suche  $\log_g x$  ( $g$  steht für den gewählten Generator.)
2. Gehe zur Logarithmstabelle und suche  $\log_g y$
3.  $s = \log_g x + \log_g y$  (Man beachte  $+$ , nicht  $\oplus$ ! Es wird folglich die reguläre Addition angewandt.)
4. Schaue in der Exponentierungstabelle nach, welcher Wert  $e$  für  $s$  hinterlegt ist.

$$e = x \otimes y$$

#### 4.3.4 Das multiplikative Inverse

Das multiplikative Invers einer Zahl  $a$  ist definiert durch  $\frac{1}{a}$ . Wir wollen aber, wie bereits in der Faktensammlung angesprochen, Divisionen um jeden Preis bei Computerchips vermeiden.

1. Finde mithilfe der Logarithmstabelle den Logarithmus für  $a$ .
2.  $b = a - 255$
3. Finde mithilfe der Exponentierungstabelle den für  $b$  hinterlegten Wert. Dieser ist  $a^{-1}$ .

#### 4.3.5 Division

1. Gehe zur Logarithmstabelle und suche  $\log_g x$
2. Gehe zur Logarithmstabelle und suche  $\log_g y$
3.  $x \oslash y = \log_g x - \log_g y \text{ mod } 255$

Trotz dieser Vereinfachung können wir die Division immer noch nicht in einem Chipdesign verwenden, da mod 255 dort nicht definiert ist. Wir müssen also einen Weg finden, wie wir den Modulooperator ausführen können ohne eine echte Division mit extrem langwierigem und fehleranfälligen Übertrag und Bitshifting programmieren zu müssen.

### 4.3.6 Äquivalent für $x \bmod 255$

Die Vorschrift zur Rundenschlüssel-Bildung setzt das Vorhandensein eines Modulo 255-Operators voraus. In der Theorie ist dies kein Problem; beim Chipdesign hingegen stehen nur Modulooperatoren für 2er-Potenzen zur Verfügung. Gehen wir also von

$$\bmod 255 \approx \bmod 256$$

aus. In  $\mathbb{N}$  wäre diese Approximation natürlich recht nutzlos. Wir müssen uns aber vor Augen führen, dass wir uns im eingeschränkten Werte von  $GF(2^8)$  befinden. Hier ist der Gedanke nämlich sehr wohl brauchbar, da sich  $x \bmod 255$  und  $x \bmod 256$  nur für  $x \in \{255, 256\}$  unterscheiden. Es ergibt sich:

$$f(x) := \begin{cases} 0 & \text{für } x = 255 \\ 1 & \text{für } x = 256 \\ x \bmod 256 & \text{für } x < 255 \end{cases} \quad (15)$$

$f(x)$  ist äquivalent zu  $x \bmod 255$  in  $GF(2^8)$ .

## 4.4 Chipentwicklung

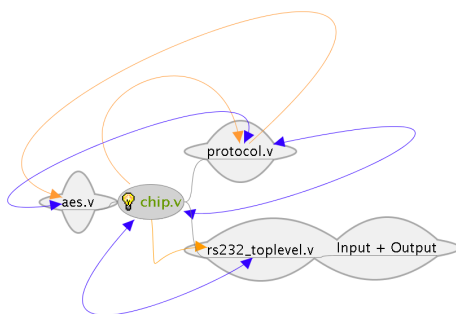


Abbildung 4: Aufteilung der Chipstruktur in einzelne Module. Pfeilen zeigen Abhängigkeiten und Kommunikation untereinander an (aus [1]).

Bei der Chipentwicklung wird der sog. „Top-Down-Level“-Ansatz angewandt. Das heißt, man versucht zunächst den Gesamtaufwand des Projektes mittels einfacher „Blasendiagramme“ einzuschätzen. Dadurch werden einzelne Teilprobleme ersichtlich, in Module aufgeteilt und die einzelnen Aufgaben des jeweiligen Moduls näher spezifiziert.

Beim Verschlüsselungsadapter bedeutet dies, dass nicht nur ein AES-Modul entwickelt werden musste, sondern auch eine Kommunikationsfunktion mit der Außenwelt bereitgestellt werden muss-

te. Dieses Interface sollte ursprünglich über eine USB-Schnittstelle verfügen. Im Verlauf der Chipentwicklung zeigte sich jedoch, dass eine USB-Verbindung nicht nur unvertretbar teurer würde, sondern auch mit erheblichem Mehraufwand (und entsprechenden Risiken bei fehlerhafter Programmierung) behaftet wäre. Aus diesem Grund entschied ich mich für die serielle Schnittstelle RS-232.

Der Programmtext für den Chipentwurf umfasst circa 4000 Zeilen, die anderen Module zusammen etwa 500. Dieser Programmtext – in der Entwurfsautomatisierung spricht man auch vom „funktionellen Entwurf“ oder schlicht dem „Design“ – kann dann wie jedes andere Programm auch *kompiliert* werden, womit man ihn in einer *Simulation* auf dem Rechner testen kann. Die Ergebnisse dieser Simulation genügen aber bei weitem noch nicht zu garantieren, dass der Entwurf auf dem fertigen Chip auch wirklich funktionieren wird. In der nächsten Stufe der Designverifikation werden die kompilierten Dateien zu so genannten Gatternetzlisten *synthetisiert*. Diese stellen einen Bauplan für die Logik und Speicherelemente des Chips dar, die man auf speziell dafür ausgelegter, externer Hardware emulieren kann. Verwendete dazu die FPSLIC-Einheit<sup>7</sup> AT94K von Atmel.

Leider unterlag die Chipherstellung gewissen Kosten- und daher Größenbeschränkungen. Zum einen wurde eine mittlerweile überholte Strukturgröße von 600 nm in der Fabrik benutzt<sup>8</sup>, zum anderen betrug die zur Verfügung stehende Fläche für den Chip nur 3 mm<sup>2</sup>.<sup>9</sup> So können insgesamt etwa 200-mal größere Entwürfe als bei diesem Projekt zur Verfügung standen problemlos als Chips gefertigt werden. Leider wiesen meine Betreuer erst sehr spät daraufhin, dass der funktionale Entwurf mit AES bei weitem zu groß für die gegebenen Grenzen war, nämlich zu einem Zeitpunkt, als er nicht nur in der Simulation funktionierte, sondern auch fehlerfrei in der Synthese. Im Anschluss musste nach der eigentlichen Deadline der Programmcode nochmals komplett durchgegangen und die AES-Verschlüsselung für den in der Fabrik zu fertigenden Chip herausgenommen werden. Die tatsächliche Funktionsweise des AES-Chips kann also nur am Computer gezeigt, nicht an einem echten Modell demonstriert werden. Dieses enthält nur eine 32-bittige XOR-Verschlüsselung von Key und Datenblock.

---

<sup>7</sup>FPSLICs sind kleine motherboard-ähnliche Geräte, auf denen das Verhalten eines Mikrochipentwurfs getestet werden kann, ohne ihn tatsächlich kostspielig in Hardware umzusetzen.

<sup>8</sup>Üblich sind bei alltäglichen Chips < 100 nm. Die im Jahr 2005 eingeweihte neue Chipfabrik des Herstellers AMD in Dresden produziert mit 65 nm-Technik.

<sup>9</sup>Usus sind bei modernen Chips über 100 mm<sup>2</sup>

### A Anhang: AES-Mindmap

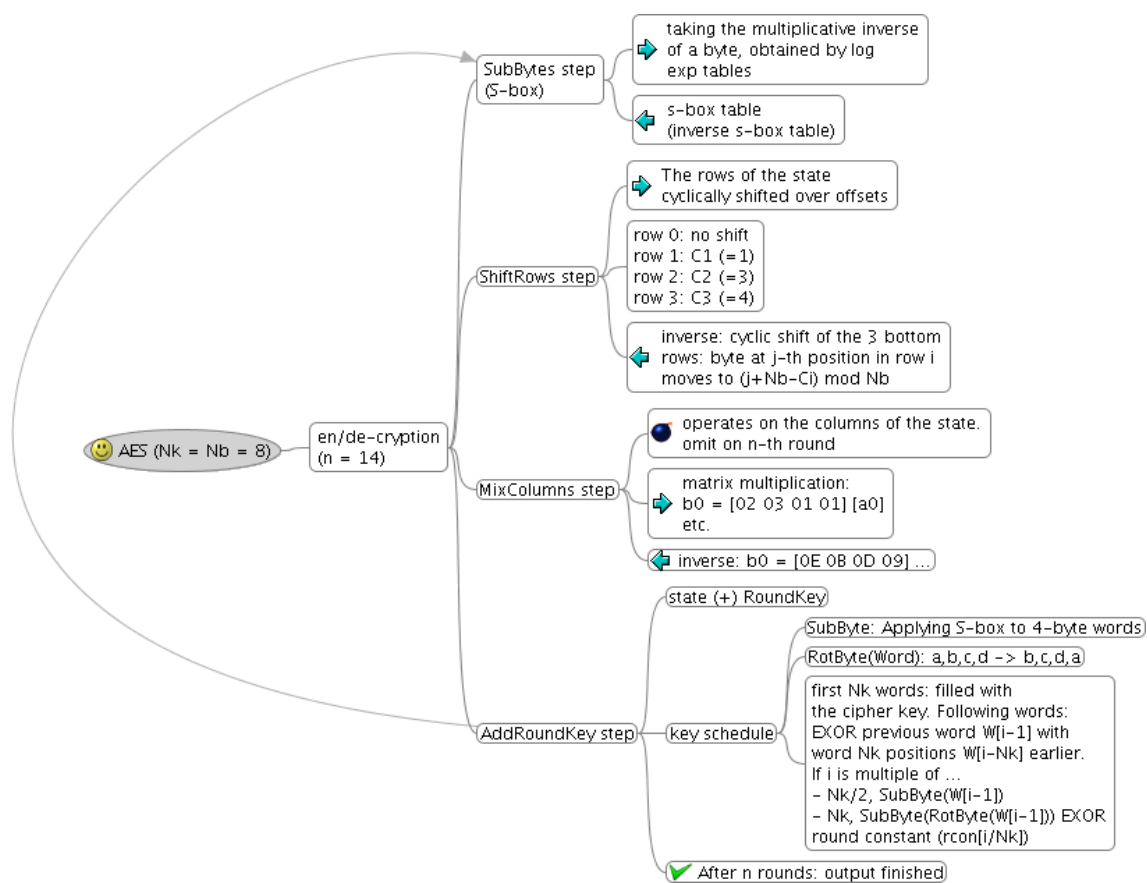


Abbildung 5: Übersichtliche Darstellung von AES aus [1]

### B Anhang: AES-Tabellen

#### S-Box

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2

80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Auf die inverse S-Box sei an dieser Stelle verzichtet; sie ist die genau umgekehrte Zuordnung der Werte der S-Box, dh.  $51 \Rightarrow d1$  wird zu  $d1 \Rightarrow 51$ .

### Logarithmstabelle

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

### Exponentierungstabelle

		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f		
---		--		--		--		--		--		--		--		--		--	
00		01	05	11	55	1a	72	a1	13	5f	38	d8	95	f7	06	1e	66		
10		e5	5c	37	eb	6a	d9	90	e6	53	04	14	44	4f	68	d3	b2		
20		4c	67	e0	4d	62	f1	18	78	83	b9	6b	dc	81	b3	49	76		
30		b5	57	10	50	0b	27	bb	61	fe	2b	87	ad	2f	93	e9	60		
40		fb	3a	d2	b7	5d	32	fa	3f	c3	e2	47	40	5b	2c	9c	da		
50		9f	d5	ac	2a	82	bc	7a	89	9b	c1	e8	65	ea	6f	c8	c5		
60		fc	21	a5	07	1b	77	b0	46	45	4a	79	86	a8	3e	c6	f3		

70 |12 5a 29 8d 8f 85 a7 0d 39 dd 84 a2 1c 6c c7 f6  
80 |03 0f 33 ff 2e 96 f8 35 e1 48 73 a4 02 0a 22 aa  
90 |34 e4 59 26 be 70 ab 31 f5 0c 3c cc d1 b8 6e cd  
a0 |d4 a9 3b d7 a6 08 28 88 9e d0 bd 7f 98 ce db 9a  
b0 |c4 f9 30 f0 1d 69 d6 a3 19 7d 92 ec 71 ae 20 a0  
c0 |16 4e 6d c2 e7 56 15 41 5e 3d c9 c0 ed 74 bf 75  
d0 |ba 64 ef 7e 9d df 8e 80 b6 58 23 af 25 b1 43 54  
e0 |1f 63 f4 09 2d 99 cb ca cf de 8b 91 e3 42 51 0e  
f0 |36 ee 7b 8c 8a 94 f2 17 4b 7c 97 fd 24 b4 52 01

## C Anhang: Danksagung

Dem VDE, der für einen Großteil des finanziellen Aufwandes des Projektes aufkam, einschließlich der enorm kostspieligen Fertigung von 20 Prototypen, bin ich zu größtem Dank verpflichtet. Bei Herrn Rieger von der Kreissparkasse Schweinfurt möchte ich mich ebenso herzlich für die Förderung meines Projektes bedanken. Ohne den VDE und ihn wäre diese Arbeit undenkbar gewesen.

Großer Dank gebührt aber auch vor allem Dipl.-Ing. Markus Holz und Dipl.-Ing. Thomas Jambor von der Universität Hannover, die mich nicht nur in einem Kurzlehrgang in Hannover in das Mikrochipdesign einführten, sondern mir auch im Verlauf der weiteren Arbeit mit wertvollen Ratschlägen betreuend zur Seite standen. Sie waren es, die mich das nötige Wissen lehrten, um überhaupt mit der Chipentwicklung beginnen zu können.

## Literatur

- [1] BELLER, MORITZ: *Spezifikation zum Chipentwurf Verschlüsselungsadapter*.  
<http://www.4momo.de/forschung/spez-chip-mbeller.pdf>, 2005.
- [2] COURTOIS, NICOLAS: *New Attacks on AES/Rijndael*.  
<http://www.cryptosystem.net/aes/>.
- [3] J. DAEMEN, V. RIJMEN: *Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES); kurz FIPS-197*. National Institute of Standards and Technology (NIST), 2001.
- [4] KURT MEYBERG, PETER VACHENAUER: *Höhere Mathematik 1*. Springer, 2001.
- [5] NIELS FERGUSON, RICHARD SCHROEPEL, DOUG WHITING: *A simple algebraic representation of Rijndael*. Selected Areas in Cryptography, Proc. SAC 2001, Lecture Notes in Computer Science #2259, pp. 103-111, Springer Verlag;  
auch <http://www.macfergus.com/pub/rdalgeq.html>, 2001.
- [6] WIKIPEDIA: *Informationsgesellschaft*.  
<http://de.wikipedia.org/wiki/Informationsgesellschaft>, 14. Juni 2005.
- [7] WIKIPEDIA: *Massenspeicher*.  
<http://de.wikipedia.org/wiki/Massenspeicher>, 15. September 2005.