Der Schlüssel zu Matlab

Jörg J. Buchholz

10. April 2002

Seit es Begehrlichkeiten zwischen Lebewesen gibt, verschließen oder verstecken wir unsere Besitztümer, um sie vor dem Zugriff Anderer zu schützen; angefangen vom Hund, der seinen Knochen im Garten vergräbt oder dem Nashornvogel, der seine komplette Familie in einem Baum einmauert, bis hin zu atombombengeschützten Hochsicherheitstresoren, in denen unser Gold lagert oder Terroristenchefs, die in weltweit ausgestrahlten Videos verschlüsselte Botschaften verstecken.

1 „Ich bin ein verschlüsselter Prinz . . .“

Gerade das letzte Beispiel zeigt, dass es in unserer Zeit immer wichtiger zu werden scheint, auch immaterielle Güter (Daten, Informationen, Nachrichten, . . .) zu schützen, beziehungsweise auf der anderen Seite zu versuchen, diesen Schutz zu durchbrechen. Die Wissenschaft, die sich mit dem Verschlüsseln einer Nachricht beschäftigt, heißt Kryptographie; die Gegenseite (und dies können durchaus „die Guten“ sein) benutzt Methoden der Kryptanalyse, um die Verschlüsselung zu brechen und so an die begehrte Nachricht zu gelangen.

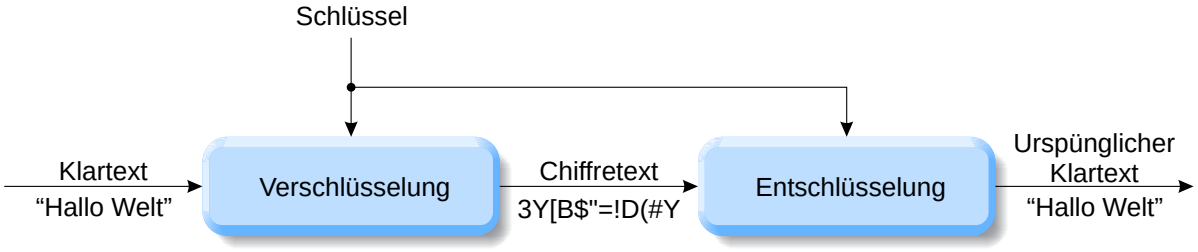

Abbildung 1: Symmetrische Ver- und Entschlüsselung mit dem gleichen Schlüssel

In Abbildung 1 ist dargestellt, wie mittels eines geheimen Schlüssels eine lesbare Nach-

richt (Klartext) zu unlesbarem Chiffretext verschlüsselt wird und wie dieser verschlüsselte Text später mit dem gleichen Schlüssel wieder in den lesbaren Originaltext entschlüsselt wird. Die Sicherheit solch eines symmetrischen Algorithmus liegt dabei verständlicherweise in der Geheimhaltung des Schlüssels. Jeder, der den Schlüssel besitzt, kann die verschlüsselte Nachricht entschlüsseln. Sollen Ver- und Entschlüsselung beispielsweise von unterschiedlichen Personen an unterschiedlichen Orten durchgeführt werden, so muss ein sicherer Kanal (Bote, persönliches Gespräch, ...) gefunden werden, über den der geheime Schlüssel ausgetauscht werden kann.

Da aber gerade der Transport eines Schlüssels ein nicht zu unterschätzendes Sicherheitsrisiko darstellt, sind in den letzten zehn Jahren asymmetrische Algorithmen mit öffentlichem Schlüssel (public key) zur Blüte gelangt, bei denen die Verschlüsselung mit einem öffentlichen, für jedermann zugänglichen Schlüssel erfolgt, während zur Entschlüsselung ein privater geheimer Schlüssel verwendet wird, der sich natürlich nicht, oder nur mit unvertretbar hohem Aufwand, aus dem öffentlichen Schlüssel berechnen lässt. Solche Public-Key-Algorithmen klinken sich mittlerweile recht nahtlos und transparent beispielsweise in gängige E-Mail-Programme ein; das Ver- bzw. Entschlüsseln einer E-Mail geschieht mit Plug-ins wie PGP (Pretty Good Privacy) kraft eines einzigen Mausklicks [1].

Wenn nun also die Sicherheit einer Chiffrierung ausschließlich von der Geheimhaltung des Schlüssels abhängt, macht es auf der anderen Seite natürlich Sinn, den Algorithmus selbst möglichst breit zu veröffentlichen, um möglichst vielen Experten die Möglichkeit zu geben, möglichst viele Fehler und Schwachstellen möglichst schnell zu finden und auszumerzen. Aus diesem Grund veröffentlichte das amerikanische NBS (National Bureau of Standards) schon im Jahre 1975 die Einzelheiten des DES (Data Encryption Standard), der, nach ein paar turbulenten Workshops, zwei Jahre später als Bundesstandard anerkannt wurde und seitdem einen unglaublichen weltweiten Siegeszug in unzähligen kryptographischen Anwendungen gefeiert hat.

Erst mit den in der letzten Dekade entwickelten Verfahren der differenziellen und linearen Kryptanalyse, die sich außerdem noch der sich nach dem Moore'schen Gesetz alle 18 Monate verdoppelnden Rechenleistung bedienen durften, konnte der DES endgültig von seinem Thron gestürzt werden. Einem Algorithmus, der heute mit handelsüblicher Hardware in ein paar Stunden geknackt werden kann, vertraut man eben vielleicht doch nicht unbedingt gerne sein Bankguthaben oder das Leben seiner Soldaten an.

2 Analytisch Effektiv Sicher

Einen weiteren unangenehmen Beigeschmack erzeugte von Anbeginn an die Tatsache, dass im DES Transformationstabellen (die so genannten S-Boxes) verwendet werden, die in einer geheimen Zusammenarbeit von IBM und der NSA (National Security Agency)

entwickelt wurden und die zwar willkürlich und zufällig aussehen, die aber nach Aussage vieler Kryptanalytiker sehr sorgfältig konstruiert wurden; möglicherweise mit dem nicht ganz uneigennützigen Ziel, den DES durch eine „Hintertür“ leichter knacken zu können.

Dies war die Geburtsstunde des designierten DES-Nachfolgers: Am 26. November 2001 veröffentlichte das NIST (National Institute of Standards and Technology) die Spezifikation des AES (Advanced Encryption Standard) [2]; verbunden mit der berechtigten Hoffnung, dass der AES am großen Erfolg seines Vorgängers anknüpfen und in den kommenden zwanzig Jahren seinen Weg in die meisten Telefone, Chipkarten, Festplatten, Neokorteximplantate, ... finden wird.

Der AES ist, genau wie sein Wegbereiter, ein Blockalgorithmus; der zu verschlüsselnde Klartext wird also in Blöcke von jeweils 128 Bits unterteilt, was bei acht Bits pro Byte bedeutet, dass immer 16 Zeichen (Bytes) gleichzeitig verarbeitet werden. Dazu werden die Zeichen, wie in Abbildung 2 dargestellt, in einem ersten Schritt spaltenweise in die so genannte Zustandsmatrix einsortiert.

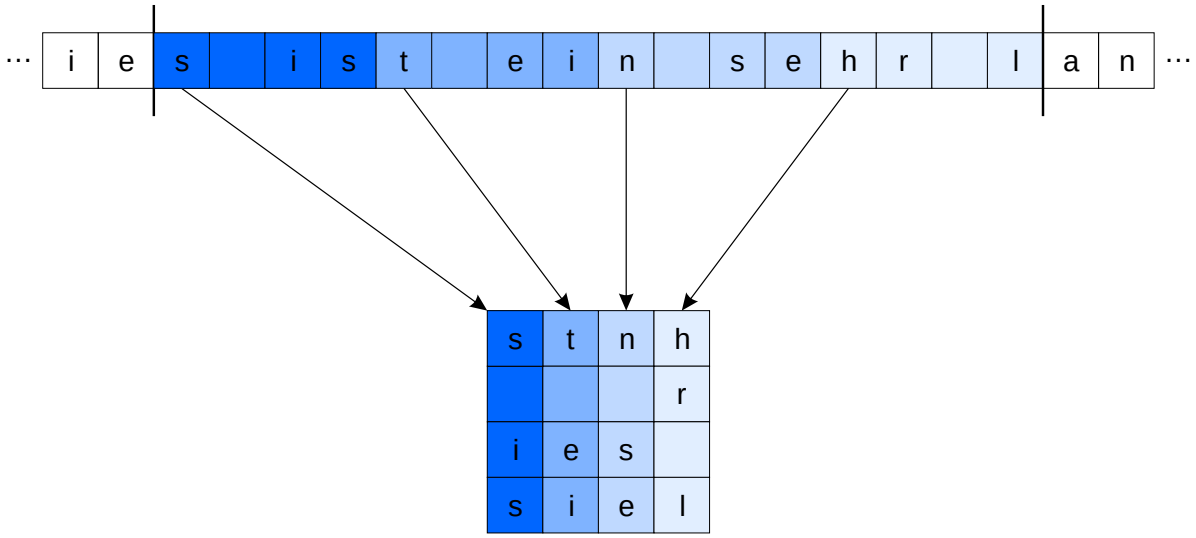

Abbildung 2: 16 Bytes bilden eine Zustandsmatrix

3 Jetzt geht's rund

Die Zustandsmatrix wird dann bei der Verschlüsselung in insgesamt elf Runden unter Benutzung des Schlüssels und der wiederholten Anwendung verschiedener Transformationen so gründlich verunstaltet, dass das analytische Rückrechnen zur Originalzustandsmatrix, ohne Kenntnis des Schlüssels, von allen Kryptanalytikern (momentan) als unmöglich bezeichnet wird:

```
% Erste Runde
state = add_round_key (state, round_key)

% Runde 2 bis Runde 10
for i_round = 2 : 10
   state = sub_bytes (state, s_box)
   state = shift_rows (state)
   state = mix_columns (state, poly_mat)
   state = add_round_key (state, round_key)
end

% Letzte Runde
state = sub_bytes (state, s_box)
state = shift_rows (state)
state = add_round_key (state, round_key)
```

Tabelle 1: Die Verschlüsselung geschieht in elf Runden

Auch was einen Brute-Force-Angriff anbelangt, also mal schnell alle möglichen Schlüssel auszuprobieren, haben Joan Daemen und Vincent Rijmen [3] aus dem Ärger mit der kurzen Schlüssellänge (56 Bit) des DES gelernt. Das NIST meint zu diesem Thema:

> „Assuming that one could build a machine that could recover a DES key in a second (i.e., try 2^{55} keys per second), then it would take that machine approximately 149 thousand-billion (149 trillion) years to crack a 128-bit AES key. To put that into perspective, the universe is believed to be less than 20 billion years old.“

Um auch für die nächsten Technologieschübe gewappnet zu sein, wurden sogar noch Varianten des AES mit 192 und 256 Bits definiert. Ob diese dann allerdings auch Quantenrechnern widerstehen, die alle Schlüssel gleichzeitig testen oder riesigen „Think-tanks“, in denen Myriaden von genmanipulierten Cracker-Bakterien ihren Lebenssinn darin sehen, die in ihren Erbanlagen individuell abgelegten Schlüssel an Kryptofutter zu versuchen, mag die Zukunft zeigen.

4 1 + 1 = 0

Die erste Verschlüsselungsrunde besteht, wie in Tabelle 1 dargestellt, nur aus einer „Addition“ (`add_round_key`) der aktuellen Zustandsmatrix (`state`) und des aus dem Schlüssel abgeleiteten ersten Rundenschlüssel (`round_key`), bei dem es sich ebenfalls um eine 4×4-Matrix handelt. Da die im AES verarbeiteten Bytes aber, wie in Kapitel 11 erläutert, als Elemente eines Galois-Feldes aufgefasst werden, muss anstelle einer „normalen“

Addition eine bit-weise Exklusiv-Oder-Verknüpfung der Elemente von Zustandsmatrix und Rundenschlüssel verwendet werden.

Und während nun herkömmliche Programmiersprachen für das elementweise xor-Verknüpfen der Matrizen natürlich gleich zwei in einander geschachtelte Schleifenkonstrukte erfordern, schlägt an dieser Stelle Matlab's Marvelous Matrix Manipulation Mastery mal wieder gnadenlos zu: der Matlab-Befehl `bitxor` lässt sich freundlicherweise nicht nur auf Skalare sondern auch direkt auf Vektoren und Matrizen anwenden, so dass sich die erste Verschlüsselungsrunde unter Matlab in einer einzigen Zeile erledigen lässt, indem statt des Befehls `add_round_key` direkt `bitxor` eingesetzt wird.[1]

5 Alte Schachteln

Die Runden zwei bis elf verwenden als erste Verschleierungstaktik das Ersetzen (`sub_bytes`) eines jeden Bytes der Zustandsmatrix mit Hilfe einer Ersetzungstabelle (`s_box`), die a priori definiert ist und die nicht wie im Falle des DES in dubioser Weise willkürlich zurechtgebastelt wurde, sondern deterministisch reproduziert werden kann. Die dazu notwendigen Algorithmen würden allerdings ein wenig den Rahmen dieses Artikels sprengen, sind aber in [2] definiert und in [4] ausführlich erläutert.

Die S-Box besteht aus den algorithmisch angeordneten 256 Elementen des Galois-Feldes $\mathrm{GF}(2^8)$, also den Zahlen von 0 bis 255, die üblicherweise in Form einer 16×16-Matrix hexadezimal dargestellt werden:

[1] Die vollständige Implementation des AES-Algorithmus in Matlab (m-Dateien und ausführliche Dokumentation) finden Sie auf der AES-Seite des Autors [4] oder auf Matlab Central.

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

Tabelle 2: Ersetzungstabelle (S-Box)

Wenn nun beispielsweise ein Element der Zustandsmatrix den Wert 7 besitzt, so wird diese 7 als Index in die S-Box aufgefasst, so dass dort der Wert $\texttt{c5}_{hex} = 197_{dez}$ gefunden (ja - das erste Element hat den Index 0) und somit die 7 in der Zustandsmatrix durch den Wert 197 ersetzt wird.

Auch hier erlaubt es Matlab eleganterweise, das gleichzeitige Ersetzen aller Elemente der Zustandsmatrix in einer einzigen Quelltextzeile auszudrücken, indem die komplette Zustandsmatrix als Indexmatrix der S-Box verwendet wird.

6 Links, zwo, drei, vier

Die nächste in jeder Runde durchzuführende Transformation (`shift_rows`) schiebt, wie Abbildung 3 verdeutlicht, die zweite Zeile der Zustandsmatrix um einen Platz nach links. Das dabei links herausfallende Element (s_{21}) wird auf den frei werdenden Platz ganz rechts in der zweiten Zeile wieder hinein geschoben.

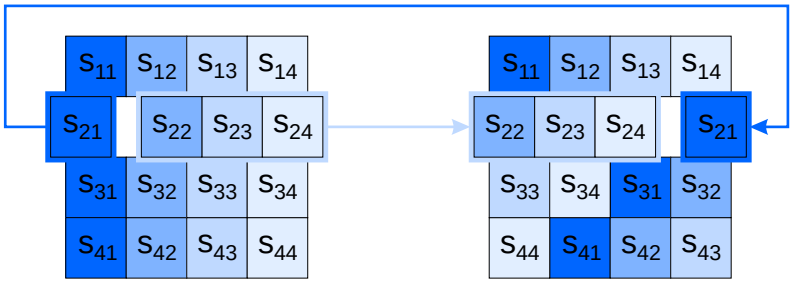

Abbildung 3: Zyklisches Verschieben der Zeilen der Zustandsmatrix

Auf die gleiche Weise werden die dritte und die vierte Zeile um zwei beziehungsweise drei Plätze nach links geschoben.

7 Alles dreht sich, alles bewegt sich

Die letzte in jeder Runde durchzuführende Transformation der Zustandsmatrix hat den aussagekräftigen Namen `mix_columns` bekommen und macht entsprechend auch genau dies. Aus der Multiplikation der Zustandsmatrix $\mathbf{S}$ mit einer vollbesetzten „Misch-Matrix“ $\mathbf{P}$ von links resultiert eine neue Zustandsmatrix $\mathbf{S}'$

$$\mathbf{S}' = \mathbf{P} \bullet \mathbf{S}$$

In der elementweisen Darstellung der Transformation

$$\begin{bmatrix} s'_{11} & s'_{12} & s'_{13} & s'_{14} \\ s'_{21} & s'_{22} & s'_{23} & s'_{24} \\ s'_{31} & s'_{32} & s'_{33} & s'_{34} \\ s'_{41} & s'_{42} & s'_{43} & s'_{44} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \bullet \begin{bmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \\ s_{41} & s_{42} & s_{43} & s_{44} \end{bmatrix}$$

wird deutlich, dass sich die einzelnen Elementen von $\mathbf{S}'$ dann tatsächlich jeweils aus einer Linearkombination der Elemente der entsprechenden Spalte von $\mathbf{S}$ berechnen:

$$s'_{23} = \begin{bmatrix} 1 & 2 & 3 & 1 \end{bmatrix} \bullet \begin{bmatrix} s_{13} \\ s_{23} \\ s_{33} \\ s_{43} \end{bmatrix} = 1 \bullet s_{13} \oplus 2 \bullet s_{23} \oplus 3 \bullet s_{33} \oplus 1 \bullet s_{43}$$

Dabei symbolisiert $\bullet$ die in Kapitel 11 erläuterte binäre Polynommultiplikation und $\oplus$ repräsentiert die bit-weise xor-Operation.

8 Last Orders Please

Zum Abschluss einer jeden regulären Runde wird nach Tabelle 1, wie schon in der ersten Runde, wieder der aktuelle Rundenschlüssel zur Zustandsmatrix addiert.

Die letzte Runde sieht fast wie eine reguläre Runde aus; lediglich der Aufruf von `mix_columns` fehlt. Auch diese Modifikation der letzten Runde geschieht zielgerichtet in der Absicht, die Kryptanalyse des Algorithmus zu erschweren.

9 Schlüsselbund

In jeder der elf Runden wird ein aktueller Rundenschlüssel verwendet. Daher müssen, üblicherweise in einer Initialisierungsphase, aus dem vom Nutzer vorgegebenen Schlüssel insgesamt elf Rundenschlüssel erzeugt werden. Unter der Annahme, dass der Schlüssel selbst aus 128 Bits besteht, lassen sich diese 16 Bytes wieder als eine 4×4-Matrix anordnen. In der Rundenschlüsselerzeugungsroutine werden dann die bekannten Transformationen (Ersetzen mittels einer S-Box, Schieben einzelner Zeilen und Addition von Konstanten) angewandt, um insgesamt elf 4×4-Matrizen zu erzeugen, die später in den Runden als Rundenschlüssel zur jeweiligen Zustandmatrix addiert werden können.

10 Alles retour

Was wäre eine Verschlüsselung ohne die zugehörige Entschlüsselung, um wieder zum ursprünglichen Klartext zurück zu gelangen? Zur Entschlüsselung des Chiffretexts muss „eigentlich nur“ jede einzelne Transformation, die während der Verschlüsselung durchgeführt wurde, rückgängig gemacht werden. Der letzte Schritt der Verschlüsselung war nach Abbildung 1 das Addieren des letzten Rundenschlüssels. Als erster Schritt der Entschlüsselung muss also der letzte Rundenschlüssel wieder subtrahiert werden. Im Rahmen der Galois-Felder wird aber sowohl die Addition als auch die Subtraktion zweier Elemente durch eine xor-Verküpfung realisiert, so dass die Entschlüsselung wiederum mit einem Aufruf der Form `state = add_round_key (state, round_key)` beginnt.

Der vorletzte Schritt der Verschlüsselung war ein Schieben der Zustandsmatrixzeilen nach links. Bei der Enschlüsselung muss daher entsprechend nach rechts geschoben werden.

Die Umkehrung der S-Box-Ersetzung geschieht am einfachsten durch die Definition und Anwendung einer inversen S-Box, in der die Werte der S-Box-Elemente als Indizes und vice versa aufgefasst werden.

Auch die in `mix_columns` durchgeführte Matrizenmultiplikation läßt sich durch die Mul-

tiplikation mit der inversen Matrix rückgängig machen. Dabei muss die Inverse natürlich in GF(2^8) gebildet werden:

$$\mathbf{P} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \quad \Rightarrow \quad \mathbf{P}^{-1} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix}$$

Wenn dann, nachdem alle elf Runden in umgekehrter Reihenfolge durchlaufen wurden, die letzte xor-Verknüpfung mit dem ersten Rundenschlüssel stattgefunden hat, sollte sich der Kreis geschlossen haben und in der Zustandsmatrix wieder der ursprüngliche Klartext zu finden sein. Wenn nicht ...

11 Galois-Felder

Alle Bytes, mit denen im AES gerechnet wird, sind Elemente eines endlichen Körpers, genauer eines Galois-Feldes und noch genauer des GF(2^8). Das GF(2^8) wird also aufgebaut von den Zahlen 0 bis 255. Dass der Körper endlich ist, bedeutet, dass auch bei einer Addition oder Multiplikation zweier Elemente von GF(2^8) wieder nur ein Element von GF(2^8) entstehen darf. Um dies zu gewährleisten, wird jedes Byte als ein Polynom ausgedrückt, dessen Koeffizienten durch die einzelnen Bits definiert sind.

Die Zahl 163 beispielsweise kann daher in dezimaler, hexadezimaler, binärer und „polynomialer“ Schreibweise beschrieben werden:

$$\begin{aligned} 163_d &= \mathrm{A3}_h \\ &= 10100011_b \\ &= 1 \cdot x^7 + 0 \cdot x^6 + 1 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0 \\ &= x^7 + x^5 + x + 1 \end{aligned}$$

Werden nun zwei Polynome addiert, so addieren sich üblicherweise die Koeffizienten gleicher Potenzen. Die Summe von 87 und 163 würde daher in Polynomschreibweise

$$\overbrace{x^6 + x^4 + x^2 + x + 1}^{87} \quad + \quad \overbrace{x^7 + x^5 + x + 1}^{163} = x^7 + x^6 + x^5 + x^4 + x^2 + 2 \cdot x + 2$$

zwei Koeffizienten mit dem Wert 2 enthalten, die sich natürlich nicht als ein Bit auffassen lassen. Diese (geraden) Koeffizienten werden daher bei der Arithmetik in GF(2^8) schlicht weggelassen, so dass sich als Ergebnis

$$87 \oplus 163 = x^7 + x^6 + x^5 + x^4 + x^2 = 11110100_b = 244_d$$

ergibt.

Auf Bitebene entspricht diese Art der Addition, wie in Abbildung 4 dargestellt, einer xor-Verknüpfung der beiden Summanden.

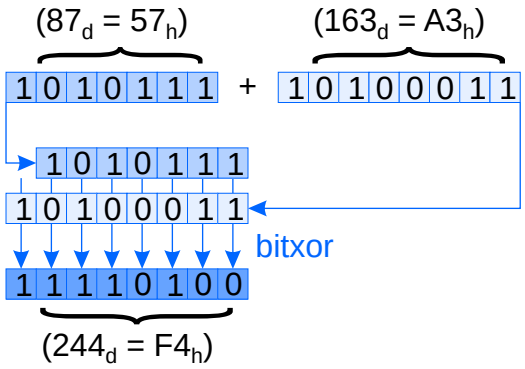

Abbildung 4: Addition durch xor-Verknüpfung

Auch bei der Multiplikation zweier Galois-Feld-Elemente werden die Faktoren als Polynome aufgefasst und im Ergebnis nur Koeffizienten von 0 bzw. 1 zugelassen. Die praktische Realisierung (Abbildung 5) benötigt daher nur Schiebe- und xor-Operationen.

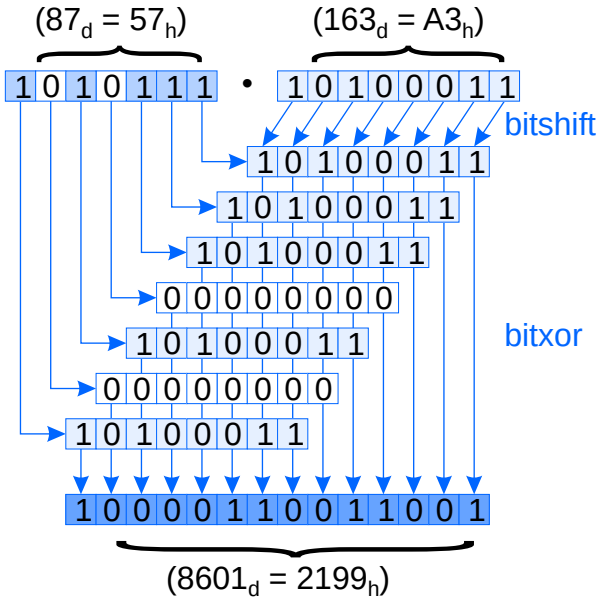

Abbildung 5: Multiplikation durch Schiebe- und xor-Operationen

Leider ergibt sich durch die Multiplikation eine Zahl ($10000110011001_b = 2199_h = 8601_d$), die mehr als 8 Bits beinhaltet (also größer als 255 ist) und daher wieder kein

Element von GF(2^8) darstellt. Um nun zu erreichen, dass sich das Ergebnis einer jeden Multiplikation zweier Bytes wieder in einem Byte unterbringen läßt, wird das Ergebnis(polynom) durch ein so genanntes irreduzibles Polynom ($283_d = 11B_h = 100011011_b = x^8 + x^4 + x^3 + x + 1$) geteilt und der dabei entstehende Divisionsrest als Endergebnis aufgefasst.

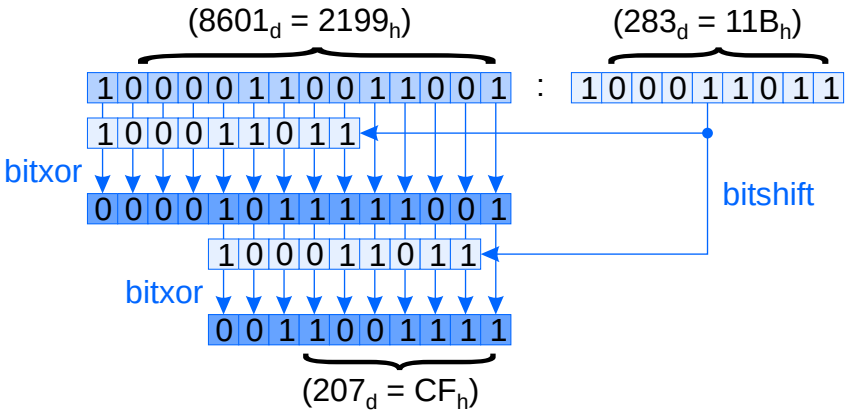

Abbildung 6: Modulo-Division

Diese Modulo-Division ist in Abbildung 6 dargestellt und kann auf Bitebene durchgeführt werden, indem der Nenner (100011011_b) jeweils mit seiner führenden 1 unter die führende 1 des aktuellen Divisionsrests geschoben wird und eine xor-Verknüpfung durchgeführt wird; solange, bis der Divisionsrest in ein Byte passt und das Ergebnis daher wieder ein Element von GF(2^8) darstellt.

Literatur

[1] Zimmermann, P.: *PGP Freeware.* `http://www.pgp.com/products/freeware`, (2002).

[2] National Institute of Standards and Technology: *Specification for the Advanced Encryption Standard (AES).* Federal Information Processing Standards Publication 197, `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`, (2002).

[3] Rijmen, V.: *The block cipher Rijndael.* `http://www.esat.kuleuven.ac.be/~rijmen/rijndael`, (2002).

[4] Buchholz, J. J.: *Matlab Implementation of the Advanced Encryption Standard.* `http://buchholz.hs-bremen.de/aes/aes.htm`, (2002).