EBERHARD-KALRS-UNIVERSITÄT TÜBINGEN
WILHELM-SCHICKARD-INSTITUT FÜR INFORMATIK
ARBEITSBEREICH FÜR THEORETISCHE INFORMATIK/FORMALE SPRACHEN

STUDIENARBEIT

EIN JAVA-APPLET ZUR VISUALISIERUNG DES ADVANCED ENCRYPTION STANDARD (AES)

VON ANDREAS LEHRMANN

SS2006

BETREUER: PRIV.-DOZ. DR. KLAUS REINHARDT

INHALTSVERZEICHNIS

1. DER AES ALGORITHMUS

1.1. EINLEITUNG

Im Januar 1997 beschloss das „National Institute of Standards and Technology“ (NIST) einen neuen Verschlüsselungsstandard mit dem Namen AES („Advanced Encryption Standard“) zu etablieren. Zu diesem Zweck wurde am 12. September 1997 ein öffentlicher Wettbewerb um den neuen Standard ins Leben gerufen. Ziel des Wettbewerbs war die Auswahl eines symmetrischen Blockchiffres mit einer Blocklänge von 128 Bit und einer variablen Schlüssellänge von wahlweise 128 Bit, 192 Bit oder 256 Bit. Der neue Standard sollte das zu diesem Zeitpunkt schon etwas betagte DES bzw. 3-DES ablösen und weltweit lizenzfrei verfügbar sein. Im August 1998 wurden auf der ersten „AES Candidate Conference“ (AES1) schließlich 15 Kandidaten der Fachwelt vorgestellt und in der Folgezeit von Kryptographieexperten aus aller Welt diskutiert und auf ihre Sicherheit überprüft. In der im August 1999 eingeleiteten Finalrunde befanden sich noch die folgenden fünf Algorithmen: MARS, RC6, Rijndael, Serpent und Twofish. Im folgenden Jahr fiel die Wahl nach weiteren eingehenden Analysen der Finalisten schließlich auf den von den beiden Belgiern John Daemon und Vincent Rijmen entwickelten Rijndael-Algorithmus (sprich: „Rheindahl“). Der gesamte Auswahlprozess ist unter [1] einzusehen. Im November 2001 wurde AES (Rijndael) schließlich in den USA zum „Federal Information Processing Standard“ (FIPS). Sämtliche den Standard betreffende Spezifikationen sind in der öffentlichen Ankündigung [2] nachzulesen.

Nach der Bekanntgabe von Rijndael als neuem Standard, wurde der Algorithmus vielerorts sofort in Hochschulvorlesungen über Kryptologie und Datensicherheit aufgenommen und hat sich dort seitdem als festes Element etabliert. Ziel der vorliegenden Studienarbeit war daher die Entwicklung eines Applets, welches den gesamten Ver- und Entschlüsselungsprozess von AES graphisch visualisiert, um zukünftigen Studenten ein Mittel an die Hand zu geben, dass ihnen den Ablauf und das Verständnis für den Algorithmus näher bringt. Auch ein didaktischer Einsatz durch die Dozenten der entsprechenden Vorlesung ist möglich. Im Gegensatz zu vielen verbreiteten Applets ist nicht nur das einfache Ver- und Entschlüsseln von beliebigen Textnachrichten möglich, sondern darüber hinaus kann der gesamte Prozess vom ursprünglichen Klartext bis zum Chiffretext bitgenau und in allen Einzelheiten verfolgt werden.

Das entwickelte Applet ist im interaktiven Kryptologie-Skript des Lehrstuhls für Theoretische Informatik/Formale Sprachen der Eberhard-Karls-Universität Tübingen unter

HTTP://WWW-FS.INFORMATIK.UNI-TUEBINGEN.DE/~REINHARD/KRYPTO/INDEX.HTML

einzusehen.

1.2. ENDLICHE KÖRPER

Eine Menge K, auf der zwei Verknüpfungen $\oplus$ und $\odot$ definiert sind heißt Körper, falls $\forall a,b,c \in K$ und $\forall x \in K \setminus \{0\}$ folgende Axiome erfüllt sind:

- $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ (Assoziativität der Addition) (1.1)
- $a \oplus b = b \oplus a$ (Kommutativität der Addition) (1.2)
- $\exists 0 \in K : a \oplus 0 = a$ (Neutralelement der Addition) (1.3)
- $\exists (-a) \in K : a \oplus (-a) = 0$ (Inverses Element der Addition) (1.4)

- $a \odot (b \odot c) = (a \odot b) \odot c$ (Assoziativität der Multiplikation) (1.5)
- $a \odot b = b \odot a$ (Kommutativität der Multiplikation) (1.6)
- $\exists 1 \in K : a \odot 1 = a$ (Neutralelement der Multiplikation) (1.7)
- $\exists (x^{-1}) \in K : x \odot (x^{-1}) = 1$ (Inverses Element der Multiplikation) (1.8)

- $a \odot (b \oplus c) = a \odot b \oplus a \odot c$ (Links-Distributivität) (1.9)

In alternativer Definition ist eine Menge K, auf der zwei Verknüpfungen $\oplus$ und $\odot$ definiert sind also ein Körper, falls $(K, \oplus, 0)$ und $(K, \odot, 1)$ abelsche Gruppen sind und die Links-Distributivität gilt. Typische Beispiele, die diese Axiome erfüllen sind der Körper der reellen Zahlen $\mathbb{R}$ und der Körper der rationalen Zahlen $\mathbb{Q}$. Im weiteren Verlauf beschränkt sich die Betrachtung auf solche Körper, die nur endlich viele Elemente besitzen. Diese besitzen die Eigenschaft, dass ihre Mächtigkeit immer einer Primzahlpotenz entspricht, d.h. für jeden endlichen Körper K ist $|K| = p^n$ für eine Primzahl p und eine natürliche Zahl n. Die Charakteristik

$$char(K) := \begin{cases} 0 & falls \sum_{k=1}^{n} 1 \neq 0 \quad \forall n \in \mathbb{N} \\ \min\left\{ n \middle| \sum_{k=1}^{n} 1 = 0, n \in \mathbb{N} \right\} & sonst \end{cases}$$

eines solchen Körpers ist stets p, die übliche Notation ist entweder $\mathbb{F}_q$ mit $q = p^n$ oder direkt $\mathbb{F}_{p^n}$, falls die Primzahlpotenz betont werden soll.

Zur Konstruktion eines Körper mit $q = p^n$ Elementen geht man folgendermaßen vor: Zunächst wählt man sich ein irreduzibles Polynom $f \in \mathbb{F}_p[X]$ mit $grad(f) = n$. f hat also die Eigenschaft, dass es keine nichtkonstanten Polynome $g, h \in \mathbb{F}_p[X]$ gibt mit $f = g \cdot h$. Mit anderen Worten: Ist $f = g \cdot h$, so gilt mindestens $g \in \mathbb{F}_p$ oder $h \in \mathbb{F}_p$.

Der gesuchte Körper ist dann

$$\mathbb{F}_{p^n} \cong \mathbb{F}_p[X]/(f), \tag{1.10}$$

wobei $\mathbb{F}_p[X]/(f)$ der Faktorring $\mathbb{F}_p[X]$ nach dem von f erzeugten Ideal ist.

Es ist also $(f) = f\mathbb{F}_p[X] = \{fg : g \in \mathbb{F}_p[X]\}$. Grundsätzlich gibt es verschiedene Repräsentantensysteme für die Restklassen modulo (f). Unter Verwendung des Standard-Repräsentantensystems (welches auch von AES verwendet wird) können die Elemente für einen solchermaßen definierten Körper betrachtet werden als

$$\mathbb{F}_{p^n} = \left\{ \sum_{k=0}^{n-1} a_k x^k \mid a_i \in \mathbb{F}_p, 0 \le i \le n-1 \right\}. \tag{1.11}$$

Addition und Multiplikation sind gegeben durch:

$$\oplus : \mathbb{F}_{p^n} \times \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}, (x_1, x_2) \mapsto (x_1 + x_2) \tag{1.12}$$

$$\odot : \mathbb{F}_{p^n} \times \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}, (x_1, x_2) \mapsto (x_1 \cdot x_2) \bmod f \tag{1.13}$$

Natürlich ist die Wahl des irreduziblen Polynoms damit entscheidend für die multiplikative Verknüpfungstafel, allerdings sind endliche Körper gleicher Mächtigkeit immer durch einen Isomorphismus ineinander überführbar und somit bis auf Isomorphie alle gleich.

Da $\mathbb{F}_p$ grundsätzlich Unterkörper jedes endlichen Körpers mit Charakteristik p ist, lässt sich ein endlicher Körper mit p^n Elementen zudem als $\mathbb{F}_p$-Vektorraum mit Basis $B = \{1, X, ..., X^{n-1}\}$ auffassen. In diesem Sinne ist $\mathbb{F}_{p^n}$ als n-dimensionaler Vektorraum isomorph zu

$$\mathbb{F}_p^n = \{(a_{n-1}, a_{n-2}, ..., a_0) \mid a_i \in \mathbb{F}_p, 0 \le i \le n-1\}. \tag{1.14}$$

Besonders wichtig für das weitere Vorgehen ist der Körper $\mathbb{F}_{2^8}$ aller Polynome p mit $grad(p) \le 7$ und Koeffizienten aus $\mathbb{F}_2 = \{0,1\}$. AES macht von diesem Körper ständigen Gebrauch. Als irreduzibles Polynom fungiert hierbei $m(X) = X^8 + X^4 + X^3 + X + 1$, d.h. der den AES-Verschlüsselungen zugrunde liegende Körper ist

$$\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1). \tag{1.15}$$

Addition und Multiplikation sind analog zu 1.12 und 1.13 definiert. Aufgrund der weiter oben erwähnten Isomorphie werden die im Zuge der AES-Verschlüsselung auftauchenden 8-Bit-Folgen (Bytes) je nach Bedarf als Element von $\mathbb{F}_2^8$ oder als Element von $\mathbb{F}_{2^8}$ aufgefasst. Auch die Darstellung als Hexadezimalzahl ist eindeutig und wird oft verwendet werden. Für ein beliebiges Byte $b = b_7 b_6 ... b_0$ ergibt sich die zweistellige Hexadezimaldarstellung $b_{Hex} = XY$ durch $X = \sum_{k=0}^{3} b_{k+4} 2^k$, $Y = \sum_{k=0}^{3} b_k 2^k$ und der üblichen Identifizierung $10 = A, 11 = B, ..., 15 = F$.

Für die Bytes $b = 11001101$ und c=00101110 ergeben sich beispielhaft also folgende Darstellungsmöglichkeiten:

Darstellung	b	c
Polynomdarstellung $\left(\mathbb{F}_{2^8}\right)$	$x^7 + x^6 + x^3 + x^2 + 1$	$x^5 + x^3 + x^2 + x$
Binärdarstellung $\left(\mathbb{F}_2^8\right)$	$(1,1,0,0,1,1,0,1)$	$(0,0,1,0,1,1,1,0)$
Hexadezimaldarstellung	CD	$2E$

Tabelle 1.1: Darstellungsformen von Bytes

Zum Teil ist für Elemente von $\mathbb{F}_2^8$ auch die Darstellung als Spaltenvektor praktischer. Zur besseren Übersicht sind diese als transponierte Zeilenvektoren gekennzeichnet, also zum Beispiel $(1,1,0,0,1,1,0,1)^T$ bzw. $(0,0,1,0,1,1,1,0)^T$. Einige beispielhafte Rechnungen in den verschiedenen Darstellungsformen sind in den Tabellen 1.2 (Addition) und 1.3 (Multiplikation) aufgeführt.

Darstellung	Addition
Polynomdarstellung $\left(\mathbb{F}_{2^8}\right)$	$\left(x^7 + x^6 + x^3 + x^2 + 1\right) \oplus \left(x^5 + x^3 + x^2 + x\right) = x^7 + x^6 + x^5 + x + 1$
Binärdarstellung $\left(\mathbb{F}_{2^8}\right)$	$(1,1,0,0,1,1,0,1) \oplus (0,0,1,0,1,1,1,0) = (1,1,1,0,0,0,1,1)$
Hexadezimaldarstellung	$CD \oplus 2E = E3$

Tabelle 1.2: Beispielhafte Additionsrechnung

Darstellung	Multiplikation
Polynomdarstellung $\left(\mathbb{F}_{2^8}\right)$	$\left(x^7 + x^6 + x^3 + x^2 + 1\right) \odot \left(x^5 + x^3 + x^2 + x\right) = x^6 + x^5 + x$
Binärdarstellung $\left(\mathbb{F}_{2^8}\right)$	$(1,1,0,0,1,1,0,1) \odot (0,0,1,0,1,1,1,0) = (0,1,1,0,0,0,1,0)$
Hexadezimaldarstellung	$CD \odot 2E = 62$

Tabelle 1.3: Beispielhafte Multiplikationsrechnung

1.3. ABLAUF

1.3.1. PREPROCESSING

Um einen Text mit Hilfe von AES verschlüsseln zu können, muss dieser zunächst in geeignete Form gebracht werden. Als iterierte Blockchiffre verwendet AES zur Verschlüsselung der einzelnen Blöcke so genannte Zustandsmatrizen. Eine Zustandsmatrix ist dabei eine $4 \times 4 - Matrix$, deren Einträge Bytes (8 Bit) sind. Da jede Zustandsmatrix genau einen Block repräsentieren soll, ist die Blocklänge dementsprechend auf 128 Bit festgelegt. Im Verlauf der Verschlüsselung werden diese Zustandsmatrizen rundenweise verschlüsselt; wir bezeichnen daher im Folgenden die Zustandsmatrix des i-ten Blocks nach der k-ten Verschlüsselungsrunde mit $Z_{i,k+2}$.

Während die Blocklänge konstant 128 Bit beträgt, sind in der AES-Spezifikation hinsichtlich Schlüssellänge und Anzahl der Verschlüsselungsrunden drei verschiedene Betriebsmodi aufgeführt (AES-128, AES-192, AES-256), wobei AES-128 allgemein als

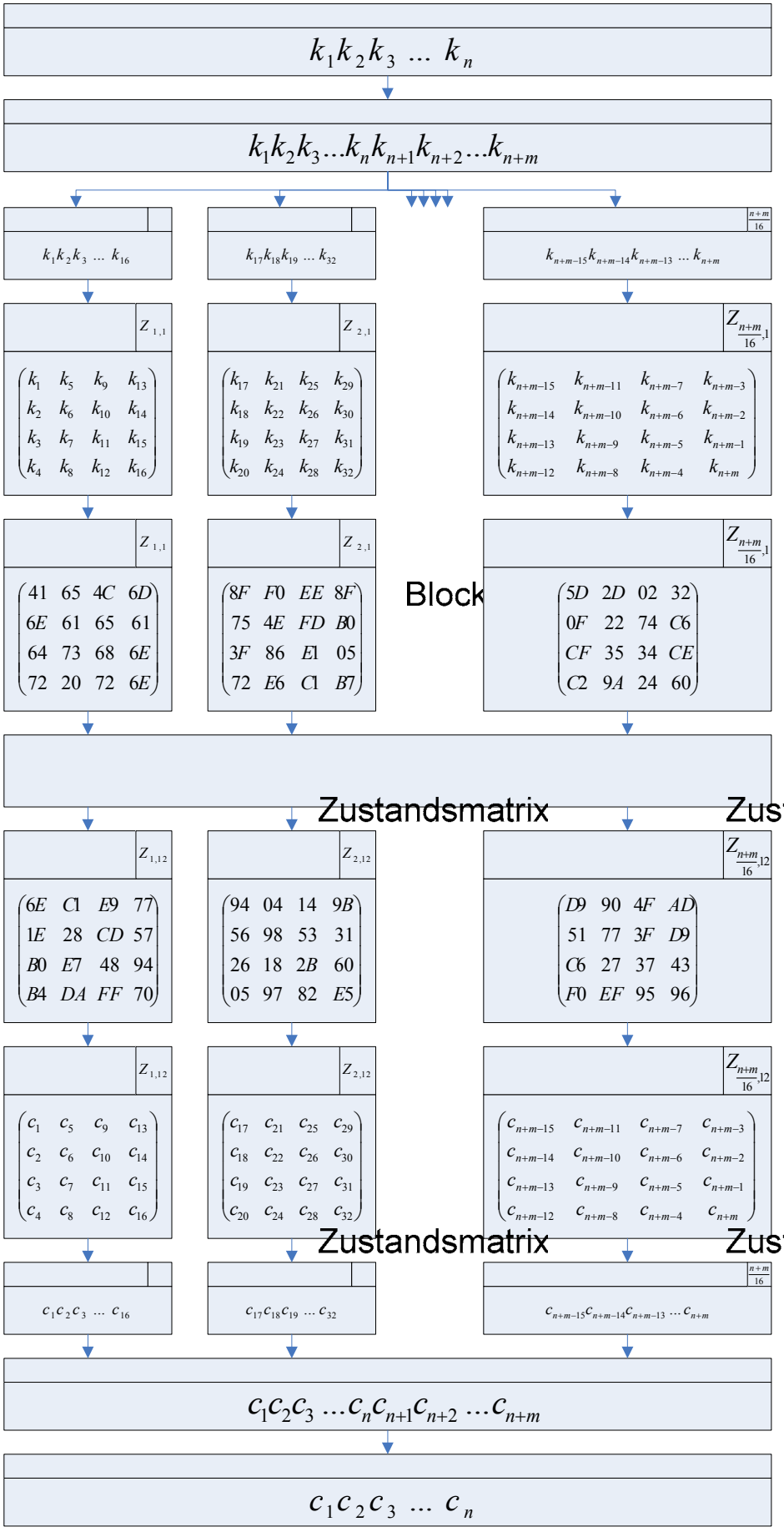
Norm angesehen wird. Alle zukünftigen Erläuterungen beziehen sich daher auf AES-128. Tabelle 1.4 zeigt die Verschlüsselungsparameter in Abhängigkeit des Betriebsmodus:

Betriebsmodus	Schlüssellänge	Blocklänge	Verschlüsselungsrunden
AES-128	128 Bit	128 Bit	10
AES-192	192 Bit	128 Bit	12
AES-256	256 Bit	128 Bit	14

Tabelle 1.4: AES-Betriebsmodi

Bevor die eigentliche Verschlüsselung beginnen kann, muss zunächst die benötigte Anzahl der Blöcke i bestimmt werden und für jeden Block die unverschlüsselte Zustandsmatrix $Z_{i,1}$ berechnet werden. Bei Verwendung von unkomprimierter Codierung entsprechend dem ASCII-Standard wird ein Zeichen durch 8 Bit codiert (7 Datenbit und 1 Paritätsbit). Bei einer Standard-Blocklänge von 128 Bit, ergibt sich demnach $\frac{128 \frac{Bit}{Block}}{8 \frac{Bit}{Zeichen}} = 16 \frac{Zeichen}{Block}$. Zur Erstellung der unverschlüsselten Zustandsmatrizen kann man daher folgendermaßen vorgehen:

Sei $Z = \{z \mid z\ ist\ Element\ des\ ASCII - Standards\}$ und $k = k_1 k_2 ... k_n, k_i \in Z$ mit $1 \le i \le n$ eine zu verschlüsselnde Nachricht mit Länge $l(k) = n$. Setze $k_p = k_1 k_2 ... k_n k_{n+1} ... k_{n+m}$ mit $m = \begin{cases} 16 - (n \bmod 16) & falls\ n \bmod 16 \neq 0 \\ 0 & sonst \end{cases}$, wobei alle k_i mit $n < i \le m$ beliebige Padding-Zeichen sein können. Auf diese Weise ist gewährleistet, dass $16 \mid l(k_p)$ gilt. k_p kann daher in $\frac{l(k_p)}{16}$ Blöcke $b_1, b_2, ..., b_{\frac{l(k_p)}{16}}$ der Länge 16 unterteilt werden mit $b_i = k_{16i-15} k_{16i-14} ... k_{16i-0}, 1 \le i \le \frac{l(k_p)}{16}$. Die so erstellten Blöcke können nun in die entsprechenden (unverschlüsselten) Zustandsmatrizen $Z_{i,1},\ 1 \le i \le \frac{l(k_p)}{16}$ überführt werden, indem sie spaltenweise eingelesen werden. Anschließend werden alle Einträge entsprechend dem ASCII-Standard in zweistellige Hexadezimalzahlen (entspricht genau den von Zustandsmatrizen geforderten 8 Bit/Eintrag) umgewandelt. Nun kann die eigentliche Verschlüsselung entsprechend dem nachfolgenden Abschnitt 1.3.2 vollzogen werden. Nach erfolgter Verschlüsselung, bei Normbedingungen also nach einer Vorrunde + 10 Verschlüsselungsrunden, haben wir aus jeder unverschlüsselten Zustandsmatrix $Z_{i,1}$ die verschlüsselte Zustandsmatrix $Z_{i,12}$ erhalten. Die verschlüsselte Nachricht wird erhalten, indem die bisherigen Schritte nun rückwärts durchlaufen werden und der Chiffretext zuletzt zusammengesetzt wird. Diagramm 1.1 fasst die bisherigen Ergebnisse zusammen. Die dortigen Hexadezimalzahlen sind rein symbolischer Natur und müssen in der Praxis ersetzt werden.



Klarte

Klartextnac

Block

Block 2

…

Verschlüsselung

1.3.2. VERSCHLÜSSELUNGSABLAUF

Basierend auf Abschnitt 1.3.1 erhält man nach erfolgtem Preprocessing die Zustandsmatrizen $Z_{1,1}, Z_{2,1}, ..., Z_{\frac{l(k_p)}{16},1}$. Diese werden nun unabhängig[1] voneinander verschlüsselt. Der folgende Abschnitt behandelt zunächst den allgemeinen Ablauf der Verschlüsselungsprozedur. Die auftauchenden Transformationen sind in Abschnitt 1.4, die Schlüsselexpansion in Abschnitt 1.5 detaillierter beschrieben.

AES nutzt zur Verschlüsselung standardmäßig einen 128-Bit-Schlüssel; die übliche Darstellung ist eine 32-stellige Hexadezimalzahl. Auch der Schlüssel (präziser: Initialschlüssel) wird durch eine $4 \times 4 - Matrix$ repräsentiert, indem die 32-stellige Hexadezimalzahl in Zweierblöcken spaltenweise eingelesen wird, so dass jeder Eintrag wieder 8 Bit umfasst. Wir bezeichnen diese Initialschlüsselmatrix im Folgenden mit $Z_{key,0}$. Mit Hilfe einer Schlüsselexpansion werden aus dem 128-Bit langen Initialschlüssel 10 weitere 128-Bit-Schlüssel generiert und in gleicher Weise als Schlüsselmatrizen $Z_{key,1} - Z_{key,10}$ repräsentiert. Die Verschlüsselung der im Preprocessing-Schritt erstellten unverschlüsselten Zustandsmatrizen $Z_{i,1}$, $1 \le i \le \frac{l(k_p)}{16}$ gliedert sich in eine Vorrunde, 9 Hauptrunden und eine Schlussrunde.

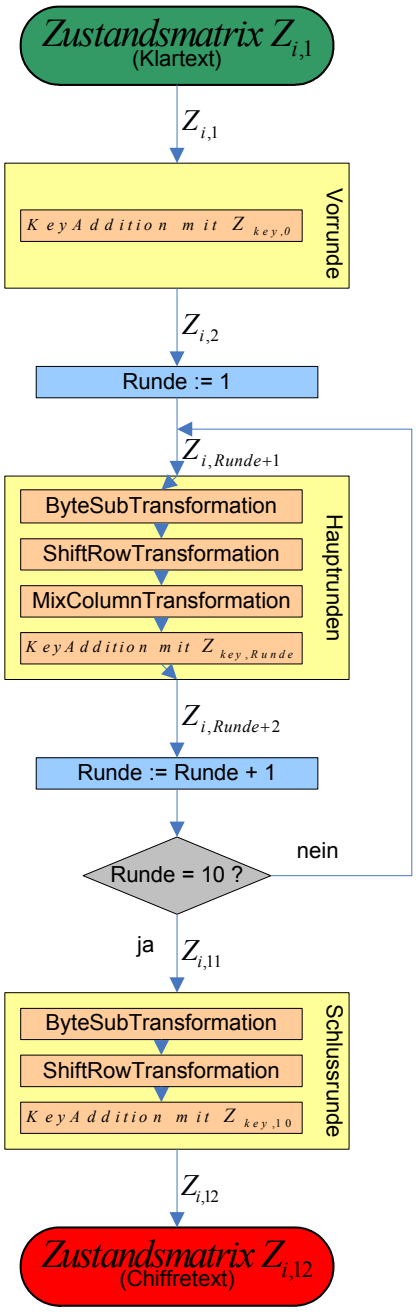


Diagramm 1.2: Verschlüsselungsablauf

- Vorrunde: Die Matrizen $Z_{i,1}$ und $Z_{key,0}$ werden als Matrizen über $\mathbb{F}_{2^8}$ aufgefasst und entsprechend der definierten Addition addiert. Die entstehende Matrix ist $Z_{i,1} \oplus Z_{key,0} = Z_{i,2}$ und dient als Eingabe der 1. Hauptrunde.

- Hauptrunden: Insgesamt sind die 9 Hauptrunden $HR_1 - HR_9$ zu durchlaufen, wobei in Hauptrunde HR_k, $1 \le k \le 9$ jeweils die Zustandsmatrix $Z_{i,k+1}$ als Eingabe dient und die Ausgabe $Z_{i,k+2}$ produziert wird. Die Überführung der Eingabe in die Ausgabe basiert auf 3 Transformationen (ByteSub-Transformation, ShiftRow-Transformation und MixColumn-Transformation), welche nacheinander auf die Eingabematrix angewendet werden sowie einer abschließenden Addition über $\mathbb{F}_{2^8}$ mit der Schlüsselmatrix $Z_{key,k}$.

[1] Es sind auch AES-Betriebsmodi bekannt, bei denen die Verschlüsselung des Blocks b_i vom Block b_{i-1} abhängt. Diese sollen hier außen vor gelassen werden.

- Schlussrunde: Die Schlussrunde, manchmal auch als 10. Hauptrunde bezeichnet um den im Standard definierten 10 Verschlüsselungsrunden gerecht zu werden, verläuft ähnlich wie die Hauptrunden $HR_1 - HR_9$, allerdings entfällt hier die MixColumn-Transformation. Auf die als Eingabe der Schlussrunde erhaltene Zustandsmatrix $Z_{i,11}$ wird also zunächst die ByteSubTransformation und anschließend die Shift-Row-Transformation angewendet. Das so erhaltene Ergebnis wird abschließend über $\mathbb{F}_{2^8}$ mit der letzten verbliebenen Schlüsselmatrix $Z_{key,10}$ addiert. Die entstehende Zustandsmatrix $Z_{i,12}$ ist die verschlüsselte Zustandsmatrix des ursprünglichen Blocks b_i bzw. der zugehörigen unverschlüsselten Zustandsmatrix $Z_{i,1}$. Diagramm 1.2 fasst den Verschlüsselungsvorgang noch einmal graphisch zusammen.

1.4. TRANSFORMATIONEN

Die drei im Folgenden beschriebenen Transformationen stellen zusammen mit der Schlüsseladdition die eigentliche Verschlüsselung des AES-Algorithmus dar. Jede Transformation erhält als Eingabe eine Zustandsmatrix und gibt auch wieder eine solche aus.

1.4.1. BYTESUB-TRANSFORMATION

Die ByteSub-Transformation ist abgesehen von der Vorrunde, die nur aus einer Schlüsseladdition besteht, in jeder Verschlüsselungsrunde die erste Transformation, der die Eingangsmatrix unterworfen wird. Sie ist nichtlinear und transformiert eine Zustandsmatrix $Z_{i,k} = (z_{mn})_{0 \le m,n \le 3}$ nicht als Ganzes, sondern komponentenweise, d.h. jeden Eintrag $z_{mn}, 0 \le m,n \le 3$ für sich. Die Transformation selbst gliedert sich in zwei Schritte: Zunächst wird unter Zuhilfenahme des Satzes von Bézout und dessen praktischer Anwendung durch den erweiterten euklidischen Algorithmus das Inverse des zu transformierenden Bytes gebildet. Im zweiten Schritt wird das gefundene Inverse einer affinen Transformation unterworfen.

1.4.1.1. Inversebestimmung

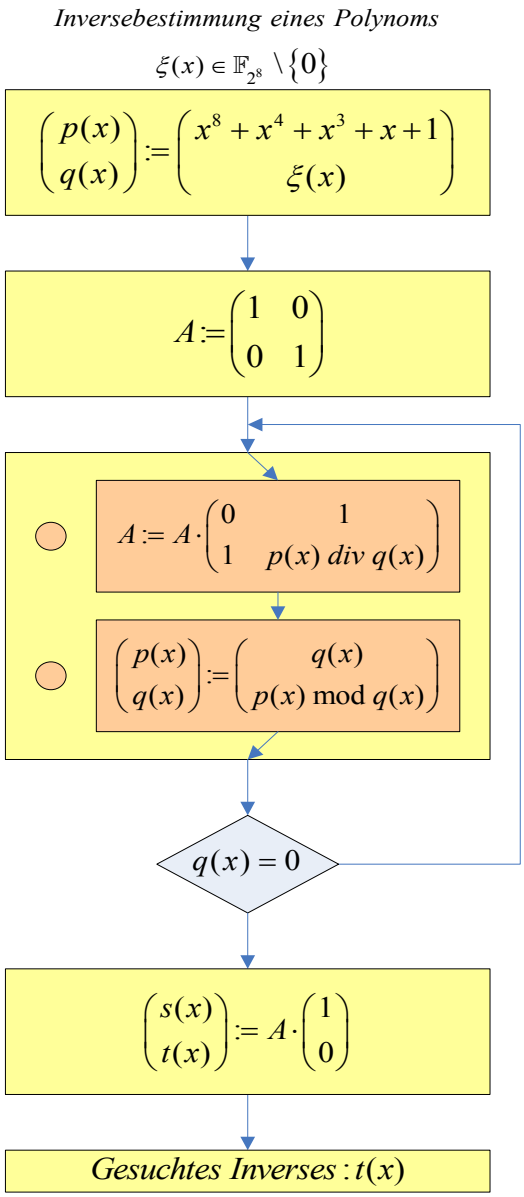
Zur Inversebestimmung wird ein zu transformierendes Byte $b = b_7 b_6 ... b_0$ zunächst als Element von $\mathbb{F}_{2^8}$, im verwendeten Standard-Repräsentantensystem also als Polynom $b(x) = \sum_{k=0}^{7} b_k x^k$, aufgefasst. Ziel dieses ersten Schritts ist die Berechnung eines Polynoms $b^{-1}(x) \in \mathbb{F}_{2^8}$, so dass gilt: $b(x) \odot b^{-1}(x) = 1$. Hierzu nutzen wir die Tatsache, dass das dem Körper $\mathbb{F}_{2^8}$ im AES-Algorithmus zugrunde liegende Polynom $m(x) = x^8 + x^4 + x^3 + x + 1$ irreduzibel ist. Hieraus folgt direkt, dass $GGT(m(x), b(x)) = 1 \quad \forall b(x) \in \mathbb{F}_{2^8}$. Nach dem Lemma von Bézout gilt nun:

$$\forall b(x) \in \mathbb{F}_{2^8} \setminus \{0\} : \exists s(x), t(x) \in \mathbb{F}_{2^8} : s(x) \odot b(x) \oplus t(x) \odot m(x) = 1 \qquad (1.16)$$

$s(x)$ und $t(x)$ können mit dem erweiterten euklidischen Algorithmus wie in Diagramm 1.3 veranschaulicht bestimmt werden. Da $t(x) \cdot m(x) \equiv 0 \pmod{m(x)} \quad \forall t(x) \in \mathbb{F}_{2^8}$, ist im Körper $\mathbb{F}_{2^8}$ aber auf Grund der Konstruktion ebenso $t(x) \odot m(x) = 0 \quad \forall t(x) \in \mathbb{F}_{2^8}$.
Damit ergibt sich:

$$s(x) \odot b(x) \oplus t(x) \odot m(x) = 1 \quad \Rightarrow \quad s(x) \odot b(x) = 1 \tag{1.17}$$

Wird der erweiterte euklidische Algorithmus also auf die Polynome $b(x)$ und $m(x)$ angewendet, ist das auf diese Weise gefundene Polynom $s(x)$ das gesuchte Inverse zum Polynom $b(x)$, d.h. $s(x) = b^{-1}(x)$. Falls $b(x) = 0$ existiert kein Inverses, denn in diesem Fall ist $b(x) \odot q(x) = 0 \neq 1 \quad \forall q \in \mathbb{F}_{2^8}$. Die Inversebestimmung entfällt dann und das Ergebnis der ByteSub-Transformation ergibt sich allein durch die im folgenden Abschnitt beschriebene affine Transformation.



1.4.1.2. Affine Transformation

Im zweiten Schritt wird das in Schritt 1 gefundene Inverse $b^{-1}(x)$ zunächst als Element von $\mathbb{F}_2^8$ aufgefasst. Sei $b^{-1} = (a_7, a_6, ..., a_0)^T$ dieses Element, so wird hierauf nun folgende affine Transformation angewandt:

$$BST_{AT} : \mathbb{F}_2^8 \to \mathbb{F}_2^8, \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \mapsto \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \tag{1.18}$$

Als Ergebnis dieser Abbildung erhält man einen Spaltenvektor $\left(b_0', b_1', ..., b_7'\right)^T$. Das Ergebnis der ByteSub-Transformation ist nun das Byte $b' = b_7' b_6' ... b_0'$. Die Struktur dieser Abbildung lässt sich dadurch erklären, dass eigentlich Bitweise substituiert wird. Gehen wir noch einmal von $b^{-1} = (a_7, a_6, ..., a_0)^T$ aus, dann ergibt sich das aus der ByteSub-Transformation resultierende Byte $b' = b_7' b_6' ... b_0'$ eigentlich durch

$$b_i' = a_i \oplus a_{(i+4) \bmod 8} \oplus a_{(i+5) \bmod 8} \oplus a_{(i+6) \bmod 8} \oplus a_{(i+7) \bmod 8} \oplus c_i \text{ mit } c = 01100011 \tag{1.19}$$

Die Abbildung BST_{AT} ist lediglich die handlichere Zusammenfassung von 1.19 auf ein ganzes Byte.

Bei der Entschlüsselung muss die ByteSubTransformation rückgängig gemacht werden. Da die affine Abbildung bijektiv ist und das Inverse des Inversen wegen $(p^{-1})^{-1} = p$ genau wie bei der Verschlüsselung berechnet werden kann, ist die Umkehrung ohne Probleme möglich. Ist nun $d' = (d_7', d_6', ..., d_0')^T$ das als Element von $\mathbb{F}_2^8$ aufgefasste Byte $d' = d_7' d_6' ... d_0'$, das der inversen ByteSub-Transformation unterworfen werden soll, muss nun zuerst die inverse affine Transformation

$$BST_{AT}^{-1} : \mathbb{F}_2^8 \to \mathbb{F}_2^8, \begin{pmatrix} d_7' \\ d_6' \\ d_5' \\ d_4' \\ d_3' \\ d_2' \\ d_1' \\ d_0' \end{pmatrix} \mapsto \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \cdot \left(\begin{pmatrix} d_0' \\ d_1' \\ d_2' \\ d_3' \\ d_4' \\ d_5' \\ d_6' \\ d_7' \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \right) \tag{1.20}$$

ausgeführt werden. Der auf diese Weise erhaltene Spaltenvektor $c = (c_0, c_1, ..., c_7)^T$ wird überführt in das Byte $c = c_7 c_6 ... c_0$. Dieses Byte wird nun aufgefasst als Polynom $c(x) \in \mathbb{F}_{2^8}$ und wie gehabt das Inverse mittels erweitertem euklidischem Algorithmus bestimmt. Das auf diese Weise gefundene Polynom $c(x)^{-1}$ wird wieder als Byte aufgefasst und stellt das Ergebnis der inversen ByteSub-Transformation dar.

1.4.1.3. S-Boxen

In der praktischen Anwendung des AES-Algorithmus wäre es viel zu aufwendig, bei jeder Anwendung der ByteSub-Transformation zuerst mittels erweitertem euklidischem Algorithmus das Inverse des zu transformierenden Bytes zu bestimmen und dieses anschließend der affinen Transformation zu unterwerfen. Aus diesem Grund werden die 256 möglichen Ergebnisse der ByteSub-Transformation vorberechnet und in einer $16 \times 16 - Matrix$

$$S = (a_{ij})_{0 \le i,j \le 15} \text{ mit } a_{ij} = BST(ij), \quad i, j \in \{0,1,...,F\} \tag{1.21}$$

gespeichert, d.h. für ein Byte $b = XY$ in Hexadezimaldarstellung ist $BST(b)$ in Reihe X/Spalte Y der Matrix S zu finden. BST steht hierbei für die komplette Anwendung der ByteSub-Transformation (also nicht nur der affinen Transformation). Nun kann das Ergebnis der Transformation bei Bedarf ganz einfach aus der Matrix S, oft auch als Substitutions-Box oder einfach S-Box bezeichnet, ausgelesen werden. Die Konstruktion von S^{-1}, die dieselbe Aufgabe für die inverse ByteSub-Transformation übernimmt, erfolgt völlig analog. Diagramm 1.4 zeigt die von AES verwendeten Substitutionsboxen S und S^{-1}.

Substitutionsbox der ByteSub-Transformation

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

Substitutionsbox der inversen ByteSub-Transformation

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

Diagramm 1.4: Substitutionsboxen

1.4.2. SHIFTROW-TRANSFORMATION

Die ShiftRow-Transformation beschreibt eine zyklische Rotation der Zeilen der Eingangsmatrix in der Weise, dass die k-te Zeile um $k-1$ Stellen zyklisch nach links verschoben wird (Zeilennummerierung beginnt bei $k=1$). Die zugehörige Abbildung lautet:

$$SRT : \mathbb{F}_{2^8}^{4\times 4} \to \mathbb{F}_{2^8}^{4\times 4}, \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \mapsto \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{pmatrix} \tag{1.22}$$

Die Umkehr der ShiftRow-Transformation bei der Entschlüsselung erfolgt völlig analog als zyklischer Rechtsshift der k-ten Zeile um $k-1$ Stellen, d.h.

$$SRT^{-1} : \mathbb{F}_{2^8}^{4\times 4} \to \mathbb{F}_{2^8}^{4\times 4}, \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \mapsto \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{13} & a_{10} & a_{11} & a_{12} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{31} & a_{32} & a_{33} & a_{30} \end{pmatrix} \tag{1.23}$$

1.4.3. MIXCOLUMN-TRANSFORMATION

Die MixColumn-Transformation arbeitet spaltenweise, indem die Eingangsmatrix $\left(a_{ij}\right)_{0\le i,j\le 3}$ in Spalten $s_j = (a_{0j}, a_{1j}, a_{2j}, a_{3j})^T, 0 \le j \le 3$ aufgeteilt wird und jede Spalte als Polynom $\vartheta_j(x) = a_{3j}x^3 + a_{2j}x^2 + a_{1j}x + a_{0j}$ mit $a_{ij} \in \mathbb{F}_{2^8}, 0 \le j \le 3$ vom Grad 3 aufgefasst wird. Für jede solchermaßen aufgefasste Spalte wird nun $\vartheta_j'(x), 0 \le j \le 3$ durch

$$\vartheta_j'(x) = \left(03x^3 + 01x^2 + 01x + 02\right) \otimes \vartheta_j(x) \tag{1.24}$$

berechnet. Wie bereits weiter oben erwähnt, sind die Koeffizienten der Polynome dabei als Elemente von $\mathbb{F}_{2^8}$ aufzufassen. Die Tatsache, dass diese Polynome nun Koeffizienten haben, die ihrerseits Elemente eines endlichen Körpers sind, macht es nötig eine gegenüber der Multiplikation $\odot$ in $\mathbb{F}_{2^8}$ etwas abgewandelte Multiplikation

$$\otimes : \mathbb{F}_{2^8}[x] \times \mathbb{F}_{2^8}[x] \to \mathbb{F}_{2^8}[x], (p,q) \mapsto p \odot q \bmod \left(x^4 + 1\right) \tag{1.25}$$

zu definieren. Der Grund hierfür ist der folgende: Würde man dieselbe Multiplikation wie in $\mathbb{F}_{2^8}$ verwenden, erhielte man als Ergebnis der Multiplikation von 1.23

$$\begin{aligned} \vartheta_j'(x) &= \left(03x^3 + 01x^2 + 01x + 02\right) \odot \vartheta(x)_j \\ &= c_{j,6}x^6 + c_{j,5}x^5 + c_{j,4}x^4 + c_{j,3}x^3 + c_{j,2}x^2 + c_{j,1}x + c_{j,0} \end{aligned} \tag{1.26}$$

mit

$$
\begin{aligned}
c_{j,0} &= \left(02 \odot a_{0j}\right) \\
c_{j,1} &= \left(01 \odot a_{0j}\right) \oplus \left(02 \odot a_{1j}\right) \\
c_{j,2} &= \left(01 \odot a_{0j}\right) \oplus \left(01 \odot a_{1j}\right) \oplus \left(02 \odot a_{2j}\right) \\
c_{j,3} &= \left(03 \odot a_{0j}\right) \oplus \left(01 \odot a_{1j}\right) \oplus \left(01 \odot a_{2j}\right) \oplus \left(02 \odot a_{3j}\right) \\
c_{j,4} &= \left(03 \odot a_{1j}\right) \oplus \left(01 \odot a_{2j}\right) \oplus \left(01 \odot a_{3j}\right) \\
c_{j,5} &= \left(03 \odot a_{2j}\right) \oplus \left(01 \odot a_{3j}\right) \\
c_{j,6} &= \left(03 \odot a_{3j}\right)
\end{aligned}
\tag{1.27}
$$

Da die Absicht jedoch darin besteht, wieder ein Polynom zu erhalten, dass eine Spalte repräsentieren kann (sprich: ein Polynom vom Grad ≤ 4), muss abschließend noch modulo $\left(x^4+1\right)$ gerechnet werden. Mittels Polynomdivision erhält man

$$
\begin{aligned}
& c_{j,6}x^6 + c_{j,5}x^5 + c_{j,4}x^4 + c_{j,3}x^3 + c_{j,2}x^2 + c_{j,1}x + c_{j,0} \bmod \left(x^4+1\right) \\
&= c_{j,3}x^3 + (c_{j,2} \oplus c_{j,6})x^2 + (c_{j,1} \oplus c_{j,5})x + (c_{j,0} \oplus c_{j,4})
\end{aligned}
\tag{1.28}
$$

Einsetzen von 1.27 liefert für die Multiplikation $\left(03x^3 + 01x^2 + 01x + 02\right) \otimes \vartheta_j(x)$ also das gesuchte Polynom $\vartheta_j'(x) = a_{3j}'x^3 + a_{2j}'x^2 + a_{1j}'x + a_{0j}'$

mit

$$
\begin{aligned}
a_{0j}' &= \left(02 \odot a_{0j}\right) \oplus \left(03 \odot a_{1j}\right) \oplus \left(01 \odot a_{2j}\right) \oplus \left(01 \odot a_{3j}\right) \\
a_{1j}' &= \left(01 \odot a_{0j}\right) \oplus \left(02 \odot a_{1j}\right) \oplus \left(03 \odot a_{2j}\right) \oplus \left(01 \odot a_{3j}\right) \\
a_{2j}' &= \left(01 \odot a_{0j}\right) \oplus \left(01 \odot a_{1j}\right) \oplus \left(02 \odot a_{2j}\right) \oplus \left(03 \odot a_{3j}\right) \\
a_{3j}' &= \left(03 \odot a_{0j}\right) \oplus \left(01 \odot a_{1j}\right) \oplus \left(01 \odot a_{2j}\right) \oplus \left(02 \odot a_{3j}\right)
\end{aligned}
\tag{1.29}
$$

Das Ergebnis der MixColumn-Transformation ist dann gegeben durch die substituierten Spalten $s_j' = (a_{0j}', a_{1j}', a_{2j}', a_{3j}')^T, 0 \leq j \leq 3$ bzw. insgesamt durch die Ausgabematrix $\left(s_j'\right)_{0 \leq j \leq 3}$.

Wie in 1.28 ersichtlich ist, lassen sich die substituierten Spalten $s_j' = (a_{0j}', a_{1j}', a_{2j}', a_{3j}')^T, 0 \leq j \leq 3$ alternativ auch durch eine Matrixmultiplikation über $\mathbb{F}_{2^8}$ berechnen. In diesem Fall ist

$$s'_j = \begin{pmatrix} a'_{0j} \\ a'_{1j} \\ a'_{2j} \\ a'_{3j} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \odot \begin{pmatrix} a_{0j} \\ a_{1j} \\ a_{2j} \\ a_{3j} \end{pmatrix}, 0 \le j \le 3 \tag{1.30}$$

Einer Ausdehnung dieser Rechnung von der einzelnen Spalte auf die gesamte Matrix steht nicht im Weg. Die MixColumn-Transformation ergibt sich daher insgesamt zu

$$MCT : \mathbb{F}_{2^8}^{4\times 4} \to \mathbb{F}_{2^8}^{4\times 4},\ A \mapsto \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \odot A \tag{1.31}$$

Die inverse MixColumn-Transformation kann nun leicht aus der normalen MixColumn-Transformation abgeleitet werden. Die Substitution der als Polynom über $\mathbb{F}_{2^8}$ aufgefassten Spalten $\varsigma(x)_j, 0 \le j \le 3$ ergibt sich nun allerdings durch

$$\varsigma'_j(x) = 0Bx^3 + 0Dx^2 + 09x + 0E \otimes \varsigma_j(x), \tag{1.32}$$

da $0Bx^3 + 0Dx^2 + 09x + 0E$ das Inverse zu $03x^3 + 01x^2 + 01x + 02$ über $\mathbb{F}_{2^8}$ ist. Analog zur obigen Rechnung ergibt sich unter diesen Vorraussetzungen nun

$$MCT^{-1} : \mathbb{F}_{2^8}^{4\times 4} \to \mathbb{F}_{2^8}^{4\times 4},\ A \mapsto \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \odot A \tag{1.33}$$

1.5. SCHLÜSSELEXPANSION

Wie in Abschnitt 1.3.2 bereits kurz angerissen, werden in AES aus dem 128 Bit langen Initialschlüssel (respektive der zugehörigen Initialschlüsselmatrix $Z_{key,0}$) 10 weitere Schlüssel (Schlüsselmatrizen) generiert. Die Schlüsseladditionen erfolgen immer am Ende jeder Runde, dabei wird der Initialschlüssel für die Vorrunde und die 10 erweiterten Schlüssel für die 9 Hauptrunden und die Schlussrunde benötigt. Die Generierung der 10 erweiterten Schlüsselmatrizen $Z_{key,1} - Z_{key,10}$ erfolgt dabei nicht als Ganzes sondern spaltenweise. Dieser Vorgang soll im Folgenden näher erläutert werden:

Zunächst wird die Initialschlüsselmatrix $Z_{key,0}$ aufgeteilt in Spalten s_0, s_1, s_2, s_3 mit $s_i = Z_{key,0} \cdot e_{i+1}, 0 \le i \le 3$, wobei e_i den i-ten kanonischen Basisvektor bezeichnet. Die restlichen benötigten Spalten $s_4, s_5, ..., s_{44}$ sind definiert durch

$$s_k = \begin{cases} s_{k-4} \oplus s_{k-1} & sonst \\ s_{k-4} \oplus \left(\left(BST\left(\left(e_4 \quad e_1 \quad e_2 \quad e_3 \right) \cdot s_{k-1} \right) \right) \oplus \begin{pmatrix} 2^{\left(\frac{k-4}{4}\right)} & 0 & 0 & 0 \end{pmatrix}^T \right) & falls\ k \equiv 0 \left(\mathrm{mod}\ 4 \right) \end{cases} \quad (1.34)$$

BST bezeichnet dabei die Anwendung der ByteSub-Transformation auf jeden Eintrag des übergebenen Spaltenvektors. Die erweiterten Schlüsselmatrizen sind dann

$$Z_{key,n} = \left(s_{4n} \quad s_{4n+1} \quad s_{4n+2} \quad s_{4n+3} \right), 1 \le n \le 10 \quad (1.35)$$

Im Einzelnen passiert also folgendes: Die Spalten 2,3 und 4 jeder erweiterten Schlüsselmatrix lassen sich leicht durch den oberen Fall von 1.34 berechnen. Jeweils bei der ersten Spalte jeder erweiterten Schlüsselmatrix gilt $k \equiv 0 \left(\mathrm{mod}\ 4 \right)$, womit der untere Fall eintritt. Ganz im Inneren sorgt hier die Multiplikation $\left(e_4 \quad e_1 \quad e_2 \quad e_3 \right) \cdot s_{k-1}$ für einen zyklischen Shift der Eingangsspalte s_{k-1} um eine Stelle (1 Byte). Auf das so erhaltene Ergebnis wird nun komponentenweise die ByteSub-Transformation angewendet. Vor der abschließenden Addition mit s_{k-4} findet noch eine Addition mit dem Spaltenvektor $\begin{pmatrix} 2^{\left(\frac{k-4}{4}\right)} & 0 & 0 & 0 \end{pmatrix}^T$ statt, welcher unabhängig von der Eingansspalte und damit bezüglich der einzelnen erweiterten Schlüsselmatrizen konstant ist. Üblicherweise werden seine Werte daher ähnlich den S-Boxen in einer Tabelle (Rcon-Tabelle) abgelegt, damit sie nicht bei jeder Schlüsselexpansion erneut bestimmt werden müssen.

2. Das Visualisierungsapplet

2.1. Oberfläche

Die Benutzeroberfläche des Visualisierungsapplets besteht aus den beiden Großteilen „Interaction“ und „View“. Im Interaction-Teil kann der Benutzer eine beliebige Textnachricht eingeben und verschlüsseln. Der View-Part dient der Auswertung des Verschlüsselungsvorgangs, indem er Hilfsmittel zur Verfügung stellt, anhand derer alle Einzelheiten des Verschlüsselungsprozesses nachvollzogen werden können. Die Benutzeroberfläche ist in Abbildung 2.1 zu sehen, das nachfolgende Listing liefert eine detaillierte Beschreibung der einzelnen Komponenten:

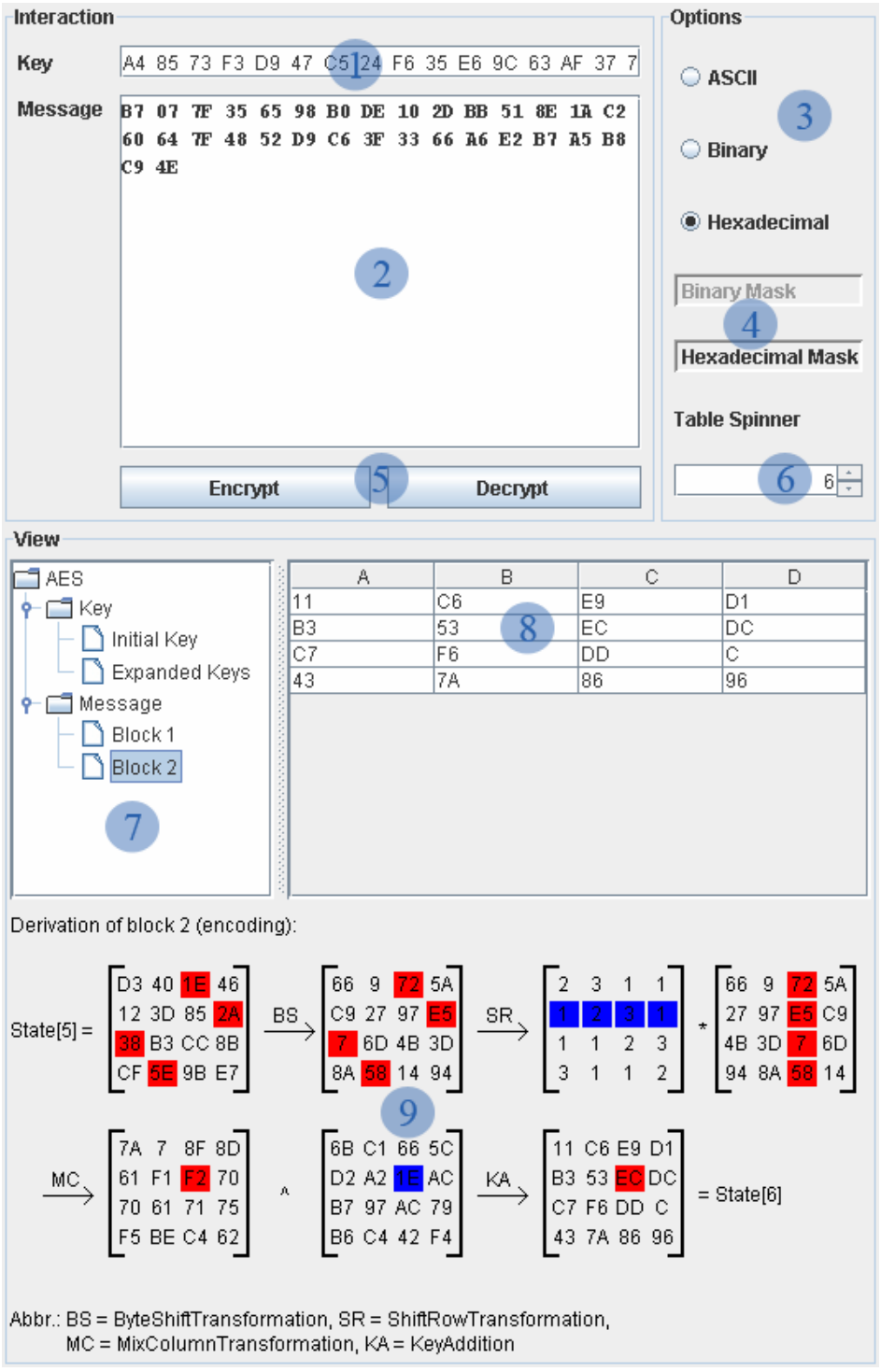


Abbildung 2.1: Benutzeroberfläche

- Schlüssel (1):

  Im Textfeld mit der Bezeichnung „Key“ kann der 128 Bit lange Initialschlüssel in Form einer 32-stelligen Hexadezimalzahl eingegeben werden. Der dort eingegebene Schlüssel wird allen weiteren Berechnungen zugrunde gelegt. Die Eingabemaske des Feldes trägt dabei Sorge dafür, dass ausschließlich gültige hexadezimale Zeichen (0-9, a-f, A-F) verwendet werden und die erforderliche Eingabelänge von 32 Stellen nicht über- oder unterschritten wird.

- Nachricht (2):

  Im „Message“-Feld kann die zu verschlüsselnde Textnachricht eingegeben werden. Die Art der Eingabe unterliegt abgesehen von den in Punkt (3) genannten optionsbedingten Restriktionen keinerlei Einschränkungen bezüglich Art oder Länge des Textes. Davon abgesehen ist bei längeren Texten natürlich auch eine entsprechend erhöhte Verschlüsselungszeit zu erwarten.

- Darstellungsform (3):

  Mit Hilfe der 3 Optionsbuttons „ASCII“, „Binary“ und „Hexadecimal“ ist es möglich, den im „Message“-Feld eingegeben Text in den entsprechenden Darstellungsformen zu betrachten. Eine Konvertierung ist grundsätzlich zu jedem beliebigen Zeitpunkt möglich, es ist allerdings darauf zu achten, dass die Textmanipulation in den Modi „Binary“ und „Hexadecimal“ einigen Einschränkungen unterliegt, weshalb diese in erster Linie zu Betrachtungs- und Anschauungszwecken im Sinne der Übersichtlichkeit dienen sollen. In den beiden genannten Modi ist es zwar möglich, die Binär- bzw. Hexadezimaldarstellung eines vorher im ASCII-Modus eingegeben Textes zu manipulieren, allerdings können bestehende Repräsentationen nicht durch zusätzliche Eingaben erweitert werden oder bereits bestehende Eingaben gelöscht werden.

- Aktivierte Masken (4):

  Die beiden Labels weisen im aktivierten Zustand darauf hin, dass die gerade getätigte Eingabe einer bestimmten Maske unterliegt. Ist als Darstellungsform „Binary“ gewählt, sind Manipulationen im Textfeld nur durch gültige Binärzeichen (0,1) möglich. Alle Versuche, andere Eingaben zu tätigen, werden ignoriert. Außerdem wird dafür gesorgt, dass die übersichtliche Darstellung in Blöcken von 8 Bit (1 Byte) gewahrt bleibt, indem selbständig zu gegebenem Anlass Leezeichen eingefügt bzw. gelöscht oder übersprungen werden. In gleichem Maße sorgt die Maske im „Hexadecimal“-Modus dafür, dass nur gültige Hexadezimalzeichen zur Eingabe benutzt werden, und die Unterteilung in Zweierblöcke aufrechterhalten bleibt. Im „ASCII“-Modus ist das „Message“-Feld mit keiner Maske belegt, so dass der Text hier nach eigenem Ermessen gestaltet werden kann.

- Verschlüsselung/Entschlüsselung (5):

  Die beiden Buttons „Encrypt“ bzw. „Decrypt“ sorgen für das Ver- bzw. Entschlüsseln der im „Message“-Feld eingegebenen Nachricht unter Benutzung des im „Key“-Feld eingetragenen Schlüssels. Die Darstellungsform wird dabei aufrechterhalten, d.h. wird auf den „Encrypt“-Button geklickt, während man sich im „Hexadecimal“-Modus befindet, wird sich nach erfolgter Verschlüsselung auch die verschlüsselte Nachricht in Hexadezimalform präsentieren. Natürlich kann auch im Nachhinein zu jeder Zeit die Darstellungsform gewechselt werden. Mehrmaliges klicken auf den „Encrypt“-Button hat zur Folge, dass die gewählte Nachricht mehrmals verschlüsselt wird bzw. dass die verschlüsselte Nachricht nochmals verschlüsselt wird. Um wieder die Originalnachricht zu erhalten, muss entsprechend mehrmals auf den „Decrypt“-Button geklickt werden.

- Table Spinner/Zustandswechsel (6):

  Das mit dem Namen „Table Spinner“ bezeichnete Auswahlfeld ist zentral zum Wechsel von Zustands- und Schlüsselmatrizen. Die Nummer gibt an, welche Matrix gerade betrachtet wird, d.h. eine „7“ gibt z.B. dass man gerade die 7. Zustandsmatrix $\left(Z_{i,7}\right)$ bzw. die 7. Schlüsselmatrix $\left(Z_{key,7}\right)$ betrachtet (je nachdem, welche Auswahl in Punkt (7) gerade aktiv ist). Über die kleinen Pfeile am rechten Rand des Feldes kann die Auswahl erhöht (Pfeil nach oben) oder erniedrigt (Pfeil nach unten) werden. Das Ergebnis, also die entsprechende Matrix, wird unter Punkt (8) angezeigt.

- Baumauswahl (7):

  Sobald eine Nachricht verschlüsselt wurde, wird die Baumstruktur mit allen relevanten Informationen gefüllt. Als Kinder des Pfades AES/Key/ finden sich die beiden Einträge „Initial Key“ und „Expanded Keys“. Ein Klick auf „Initial Key“ sorgt dafür, dass unter Punkt (8) die Initialschlüsselmatrix $\left(Z_{key,0}\right)$ angezeigt wird, d.h. der Benutzer kann verfolgen, wie sein unter Punkt (1) eingegebener Schlüssel zur entsprechenden Matrix verarbeitet wurde. Wählt man dagegen „Expanded Keys“ aus, werden die nach Abschnitt 1.5 generierten erweiterten Schlüsselmatrizen angezeigt. Durch die einzelnen erweiterten Schlüsselmatrizen kann wie in Punkt (6) beschrieben hindurch navigiert werden, indem der „Table Spinner“ entsprechend erhöht oder erniedrigt wird. Der Pfad AES/Message/ enthält als Kinder eine Auflistung aller Blöcke, in die der Text zerlegt wurde. Wird ein entsprechender Block ausgewählt, so können analog zu den erweiterten Schlüsselmatrizen die zu diesem speziellen Block gehörigen Zustandsmatrizen angezeigt werden, indem mit dem „Table Spinner“ eine entsprechende Auswahl getroffen wird. Gleichzeitig mit der Auswahl in der Baumstruktur wird sich auch das eigentliche Analysepanel (Punkt(9)) ändern. Auf die hier zur Verfügung stehenden Möglichkeiten soll unter Punkt (9) separat eingangen werden.

- Matrixrepräsentation (8):

  Wie schon öfter erwähnt, dienen die 4 Zeile und 4 Spalten dieser Tabelle der Repräsentation einer $4 \times 4 - Matrix$. Angezeigt wird jeweils die Matrix, die in Abhängigkeit von Punkt (6) (Welche Matrixnummer?) und Punkt (7) (Zustands- oder Schlüsselmatrix?) ausgewählt wurde. Außerdem wird über die ausgewählte Zelle die unter Punkt (9) näher beschriebene Herkunftsanzeige gesteuert.

- Analysepanel/Verschlüsselungsablauf (9):

  Mit Hilfe dieses Panels können alle während der Verschlüsselung auftretenden Daten eingesehen und analysiert werden.

  - Ist unter Punkt (8) z.B. „Block 1“ ausgewählt und der „Table Spinner“ auf „3“ gestellt, werden hier alle Transformationen und Schlüsseladditionen mitsamt Intermediärmatrizen angezeigt, die letztlich von der Zustandsmatrix $Z_{1,2}$ zur Zustandsmatrix $Z_{1,3}$ geführt haben. Wird nun unter Punkt (7) eine bestimmte Zelle ausgewählt, wird die Herkunftsanzeige aktiviert und es werden alle Einträge farblich hervorgehoben, von denen der selektierte Eintrag abhängig ist. Abhängigkeiten von der vorhergehenden Zustandsmatrix sind rot markiert, Abhängigkeiten von Schlüsselmatrizen oder Konstanten sind blau markiert. Während die Ergebnisse der MixColumn- und ShiftRow-Transformation bzw. der Schlüsseladdition bei Kenntnis der theoretischen Grundlagen direkt aus der Visualisierung ersichtlich sind, gestaltet sich dies bei der ByteSub-Transformation schwieriger. Hierzu gibt es die Möglichkeit, auf einen Eintrag der aus einer ByteSub-Transformation resultierenden Matrix zu klicken; es öffnet sich dann ein Fenster, in dem der Substitutionsweg genau nachverfolgt werden kann (siehe auch Abschnitt 2.1.1).

  - Grundsätzlich kann nicht nur der Weg von der unverschlüsselten Nachricht zur verschlüsselten Botschaft mitverfolgt werden, sondern auch der umgekehrte Weg zurück zur Originalnachricht. Der „Table Spinner“ benutzt hierzu eine Trenderkennung, nach der er sich richtet: Standardmäßig beginnt der „TableSpinner“ im „Verschlüsselungstrend“. Solange in Richtung „Verschlüsselung“ geblättert wird, d.h. auf den nach oben zeigenden Pfeil geklickt wird, wird bei Auswahl k im „Table Spinner“ immer der Weg von $Z_{i,k-1} \rightarrow Z_{i,k}$ angezeigt. Wird nun an einer bestimmten Stelle der Trend geändert und auf den Pfeil nach unten geklickt, verbleibt der „Table Spinner“ zunächst in der bisherigen Auswahl, wechselt jedoch in den „Entschlüsselungstrend“ und zeigt den Weg von $Z_{i,k} \rightarrow Z_{i,k-1}$ an. Erst beim zweiten Klick auf den Pfeil nach unten wechselt die Auswahl auf $k-1$ mit Anzeige von $Z_{i,k-1} \rightarrow Z_{i,k-2}$. Der „Table Spinner“ befindet sich nun solange im „Entschlüsselungstrend“ bis dieser durch einen Klick auf den Pfeil nach oben wieder in den „Verschlüsselungstrend“ geändert wird. Diese Technik hat zur Folge, dass abwechselndes Klicken auf den Pfeil nach oben und den Pfeil nach unten an der Auswahl nichts ändert, sondern auf diese Weise immer bequem zwischen den Trends gewechselt werden kann.

- Genau wie die Herkunft der Zustandsmatrizen kann auch die Entstehung der erweiterten Schlüsselmatrizen mitverfolgt werden. Hierzu wird unter Punkt (8) der Eintrag „Expanded Keys“ im Pfad AES/Key ausgewählt. Die Herkunftsanzeige funktioniert hier genau wie bei Zustandsmatrizen auch durch Auswahl einer bestimmten Zelle. Zur Verdeutlichung, dass die erweiterten Schlüsselmatrizen spaltenweise generiert werden und dabei immer von Spalten abhängen, die zu einer vorhergehenden Schlüsselmatrix gehören, wurden die Spalten der vorhergehenden und nachfolgenden Schlüsselmatrizen ebenfalls in die Visualisierung integriert. Um zu sehen, wie die Berechnung einer bestimmten Spalte von statten geht, kann diese einfach im Analysepanel angeklickt werden.

2.1.1. VISUALISIERUNG DER BYTESUB-TRANSFORMATION

Im Gegensatz zu allen weiteren Transformationen und der Schlüsseladdition war die ByteSub-Transformation nicht direkt in das Analysepanel zu integrieren, ohne Eingeständnisse an die Übersichtlichkeit zu machen. Um den Verschlüsselungsweg dennoch bestmöglich zu visualisieren, wurden die Vorgänge während der ByteSub-Transformation in ein externes Fenster ausgelagert, welches nach einem Klick auf einen Eintrag einer aus der ByteSub-Transformation resultierenden Matrix erscheint. Es werden dann alle relevanten Informationen angezeigt, die durch die ByteSub-Transformation zu dem angeklickten Wert geführt haben. Im Einzelnen wurden also die Inversebestimmung und die affine Transformation wie in Abbildung 2.2 ersichtlich visualisiert. Analog hierzu kann auch die inverse ByteSub-Transformation eingesehen werden, indem bei bestehender Visualisierung der Entschlüsselung auf die entsprechende Matrix geklickt wird.

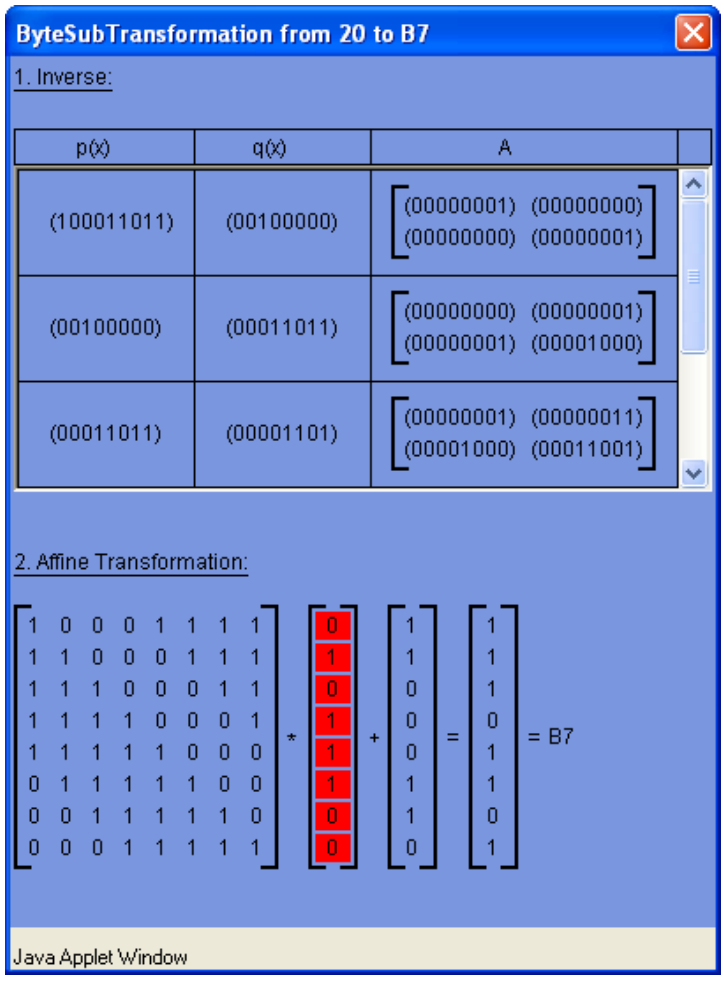


Abbildung 2.2: ByteSub-Transformation

Zur Inversebestimmung:

Die zu sehende Tabelle zeigt den Ablauf des erweiterten euklidischen Algorithmus, wie er in Diagramm 1.3 dargestellt ist. Sollte die Bestimmung mehr als 3 Durchläufe benötigt haben, können die weiteren Ergebnisse durch herunterscrollen der Bildlaufleiste am rechten Rand der Tabelle eingesehen werden. Um eine kompakte Darstellung zu erreichen, wurden die Polynome jeweils durch 8-tupel dargestellt. Dementsprechend ist also $\sum_{k=0}^{7} a_k x^k = (a_7 a_6 ... a_0)$.
Aufgeführt sind in der Tabelle alle Werte, die durch die Polynome $p(x)$ und $q(x)$ sowie die $2 \times 2 - Matrix \quad A$ im Verlauf des Algorithmus angenommen werden, bis dieser entsprechend Diagramm 1.3 bei $q(x) = 0$ abbricht. Auf die explizite Herleitung des Inversen aus dem letzten Stand der Matrix A wurde verzichtet, da dieses nach dem Abbruch direkt in Zeile 2/Spalte 1 der Matrix A abgelesen werden kann; die entsprechende Zelle wurde hierzu rot markiert.

Zur affinen Transformation:

Die gesamte affine Transformation ist nochmals abgebildet. Das im ersten Teil gefundene und nun in die affine Transformation eingesetzte Inverse ist wiederum rot markiert.

2.2 INTERNA

Das in Abschnitt 2.1 beschriebene Applet besteht aus insgesamt 23 Klassen und 3 Enumerationen. Letztere dienen in erster Line dazu, den Quellcode übersichtlicher zu gestalten und auf bloßen Zahlen basierte Optionen einer Methode zu vermeiden. Die Funktionalität der einzelnen Klassen soll im Folgenden kurz angerissen werden:

- **class** `AES`: Dies ist die zentrale Klasse für die Ver- und Entschlüsselung einer Nachricht mittels AES. Sie enthält alle benötigten Transformationen, Schlüsseladdition und –expansion, sowie eine Methode „Transform“, die den Ablauf der Ver- und Entschlüsselung regelt.

- **class** `Applet1`: Der Inhalt dieser Klasse umfasst die Initialisierungen und Konfigurationen sämtlicher Komponenten, die die Benutzeroberfläche betreffen. Auch alle für das Layout der Komponenten relevanten Daten sind hier implementiert. Einige Methoden des Nachrichtenverkehrs, die eng mit der Benutzeroberfläche verbunden sind, sind ebenfalls an dieser Stelle zu finden.

- **class** `BSTDialog`: Diese aus JDialog abgeleitete modale Klasse stellt die gesamte Benutzeroberfläche des zur Visualisierung der ByteSub-Transformation benötigten externen Fensters zur Verfügung.

- **class** `Constants`: In dieser Klasse sind alle im Verlauf der Verschlüsselung benötigten Konstanten abgelegt. Hierzu zählen unter anderem die S-Box, die inverse S-Box und die Matrix, mit der während der MixCoumn-Transformation multipliziert wird.

- **class** `Convert`: Alle zum Wechsel der Darstellungsform benötigten Methoden sind in der Klasse „Convert“ integriert. Im Einzelnen sind dies die Umwandlungen

  $ASCII \rightarrow HEX$
  $ASCII \rightarrow BIN$
  $HEX \rightarrow ASCII$
  $HEX \rightarrow BIN$
  $BIN \rightarrow ASCII$
  $BIN \rightarrow HEX$

- **class** `DerivationPanel`: Hier werden alle Nachrichten empfangen und verarbeitet, die die Anzeige des Analysepanels betreffen. Hierzu zählen insbesondere Mausereignisse und Änderungen am TableSpinner oder der Baumauswahl. Desweiteren wird die Koordination übernommen, welche Art von Inhalt (Verschlüsselung, Entschlüsselung, Schlüsselgenerierung) im Analysepanel angezeigt wird.

- **class** `DPDecodingHandler`: Wird von der Klasse DerivationPanel aufgerufen, falls eine Entschlüsselung im Analysepanel angezeigt werden soll. Die Klasse DPDecodingHandler besorgt sich dann vom AES-Objekt alle benötigten Daten und generiert/positioniert die relevanten Matrizen im Analysepanel. Auch sonstige Zeichnungen (Pfeile, etc) werden übernommen.

- **class** `DPEncodingHandler`: Wie `DPDecodingHandler`, nur für den Verschlüsselungsvorgang.

- **class** `DPKeyHandler`: Wie `DPDecodingHandler`, nur für die erweiterten Schlüsselmatrizen.

- **class** `EGGT`: Stellt für ein bestimmtes Polynom die von der Klasse `BSTDialog` genutzte Tabelle des erweiterten euklidischen Algorithmus zur Verfügung. Der Aufgabenbereich umfasst nur die graphische Repräsentation; der Datenteil wird von der Klasse `Polynomial` übernommen.

- **class** `Mask`: Durch die Klasse `Mask` werden alle im Nachrichtenfeld getätigten Tastaturanschläge erfasst. Befindet sich der Benutzer im Hexadezimalmodus oder im Binärmodus, wird hier dafür Sorge getragen, dass die Darstellung (Zweierblöcke im Hexadezimalmodus und Achterblöcke im Binärmodus) erhalten bleibt, indem die Tastatureingaben entsprechend weiterverarbeitet und angepasst werden. Hierzu gehören insbesondere ein modifiziertes Löchen und Überschreiben von bestehenden Einträgen.

- **class** `Matrix`: Stellt die grundlegende Repräsentation einer Matrix dar. Hierzu gehört sowohl der Datenteil, als auch die graphische Realisierung auf der Benutzeroberfläche.

- **class** `Misc`: Oft benötigte allgemeine Methoden, die sonst keiner anderen Klasse zuzuordnen waren, sind in dieser Klasse abgelegt.

- **class** `Polynomial`: Die zentrale Klasse zur Repräsentation eines Polynoms. Sie umfasst sowohl alle benötigten arithmetischen Operationen auf Polynomen (jeweils speziell an die Bedürfnisse des Körpers $\mathbb{F}_{2^8}$ angepasst) als auch die oft benötigte Typkonvertierung zwischen Bytes und Polynomen. Auch die Bestimmung des Inversen eines gegebenen Polynoms ist hier implementiert.

- **Class** `Spinner`: Erbt von JSpinner und erweitert diese um die weiter oben beschriebene Trenderkennung

- **Class** `SpinnerAction`: Implementiert das ChangeListener-Interface und ist zuständig für die Benachrichtigung aller Komponenten, die über Änderungen an er Auswahl des „TableSpinner's" informiert werden müssen.

- **Class** `Table`: Erweitert die Klasse JTable durch die Möglichkeit, Nachrichten über Änderungen an der Baumauswahl zu empfangen und entsprechend weiter zu verarbeiten. Insbesondere müssen der „Table Spinner" und das „Table Model" vom Table-Objekt über solcherlei Änderungen informiert werden.

- **Class** `TableAction`: Implementiert das ListSelectionListener-Interface und informiert das Analysepanel über Änderungen an der Zellauswahl, so dass die Herkunftsanzeige immer auf dem aktuellen Stand ist.

- **Class** `TableModel`: Verwaltet die vom AES-Objekt bereitgestellten Daten und übernimmt die Koordination, welche Informationen, d.h. welche Zustands- oder Schlüsselmatrizen, in der Tabelle dargestellt werden.

- **Class** `TextArea`: Modifizierte Variante der Klasse JTextArea, die die Möglichkeit bietet, verschiedene Mask-Objekte zu verwalten und zu gegebener Zeit zu aktivieren.

- **Class** `TextAreaAction`: Implementiert das KeyListener-Interface und ist dafür zuständig, alle Tastaturereignisse des Nachrichtenfeldes an die gerade aktivierte Maske (so denn eine aktiviert ist) weiter zu leiten, wo sie dann entsprechend verarbeitet werden.

- **Class** `TreeAction`: Implementiert das TreeSelectionListener-Interface und informiert alle relevanten Komponenten über den gerade selektierten Pfad, so dass diese ihre Visualisierung entsprechend anpassen können.

A. LITERATURVERZEICHNIS

[1] James Nechvatal, et al.,
Report on the Development of the Advanced Encryption Standard (AES),
2000

[2] Federal Information Processing Standards Publication 197,
Specification for the ADVANCED ENCRYPTION STANDARD (AES),
2001

[3] John Daemon, Vincent Rijmen,
The Design of Rijndael,
1. Auflage, 2001, Springer, Berlin

[4] Peter Hauck,
Skript Kryptologie und Datensicherheit,
2005, pp. 75-85

[5] Manfred Wolff, Peter Hauck, Wolfgang Küchlin,
Mathematik für Informatik und Bioinformatik,
1. Auflage, 2004, Springer, Berlin, pp. 87-132

[6] Christian Ullenboom,
Java ist auch eine Insel,
4. Auflage, 2005, Galileo Press GmbH, Bonn

[7] Guido Krüger,
Handbuch der Java-Programmierung,
3. Auflage, 2002, Addison-Wesley, München