Kapitel 4
Der Advanced Encryption Standard Rijndael

§1: Geschichte und Auswahlkriterien

DES wurde in Zusammenarbeit mit der National Security Agency der Vereinigten Staaten von IBM entwickelt und dann als amerikanischer Standard verkündet. Diese Vorgehensweise weckte von Anfang an den Verdacht, daß möglicherweise eine „Falltür“ eingebaut sei, insbesondere da zumindest ursprünglich nicht alle Design-Kriterien publiziert wurden.

Nachdem dreißig Jahre intensiver Kryptanalyse keine Falltür gefunden haben, müßte diese – so vorhanden – zumindest sehr gut versteckt sein; aber egal ob mit oder ohne Falltür: Wie wir im letzten Kapitel gesehen haben, erfüllt DES mit nur 56 Bit Schlüssellänge nicht mehr die heutigen Sicherheitsanforderungen. Ursprünglich war er ohnehin nur für eine Laufzeit von zehn Jahren vorgesehen und wurde nur immer wieder verlängert, weil die meisten Anwender von Kryptographie äußerst träge sind und sich nicht um Fortschritte der Kryptanalyse kümmern.

Obwohl es die Internationale Standardisierungsorganisation ISO ablehnt, ein Kryptoverfahren zu standardisieren (Ein Grund dafür ist die dann befürchtete Bündelung krimineller Energie auf dieses Verfahren), hat das das amerikanische Handelsministerium die Suche nach einem Nachfolgealgorithmus für DES am 2. Januar 1997 international ausgeschrieben; der vorläufige Name des Algorithmus war AES *(Advanced Encryption Standard).* Federführend für die Auswahl war das *National Institute of Standards and Technology* (NIST) in Gaithersburg, Maryland.

Dieses ging von Anfang an davon aus, daß der zu wählende Algorithmus *stärker* sein müsse als Triple DES; er sollte zwanzig bis dreißig Jahre lang anwendbar sein und dementsprechende Sicherheit bieten. Nach einer internationalen Konferenz über die Auswahlkriterien am 15. April 1997 veröffentlichte es am 12. September 1997 die endgültige Ausschreibung.

Minimalanforderung an die einzureichenden Algorithmen waren danach, daß es sich um symmetrische Blockchiffren handeln muß, die mindestens eine Blocklänge von 128 Bit bei Schlüssellängen von 128 Bit, 192 Bit und 256 Bit vorsieht.

Als Kriterien für die Wahl zwischen den einzelnen Algorithmen wurden die folgenden Aspekte genannt:

1. **Sicherheit:** Wie sicher ist der Algorithmus im Vergleich zu den anderen Kandidaten? Inwieweit ist seine Ausgabe ununterscheidbar von der einer Zufallspermutation? Wie gut ist die mathematische Basis für die Sicherheit des Algorithmus begründet? (Im Gegensatz zu DES sollten dieses Mal alle Kriterien publiziert werden.)
2. **Kosten:** Welche Lizensgebühren werden fällig? Wie effizient geht der Algorithmus mit Rechenzeit und Speicherplatz um?
3. **Flexibilität:** Ein Algorithmus, der auf einer Vielzahl von Plattformen implementierbar ist (PCs, 8-Bit-Prozessoren, ATM-Netze, HDTV, ...) ist vorzuziehen, genauso einer, der auch als Stromchiffre, kryptographisch sichere Hash-Funktion oder ähnliches verwendet werden kann.
4. **Software:** Der Algorithmus *muß* auch softwaremäßig effizient implementierbar sein.
5. **Einfachheit:** Der Aufbau des Algorithmus soll möglichst einfach sein.

Nicht nur die Art der Suche unterschied sich also beträchtlich von der DES-Entwicklung hinter verschlossenen Türen, auch die Auswahlkriterien hatten sich stark geändert: DES war noch in erster Linie für Hardware entworfen worden; manche Aspekte von DES wie etwa die für die kryptographische Sicherheit völlig irrelevante Anfangspermutation hatten wohl keinen anderen Sinn als die Verlangsamung von Software-

Implementierungen.

Inzwischen hat freilich die Erfahrung mit der Kryptographie in der Geschäftswelt gezeigt, daß man dort den Kostenaufwand für Spezialhardware scheut und sich lieber mit den dubiosen Verfahren begnügt, die in der ohnehin vorhandenen Office-Software eingebaut ist (Preis für das Knacken durch spezialisierte Unternehmen je nach Programm zwischen 40 $ und 250 $), wohingegen sich die Hacker innerhalb und außerhalb der Geheimdienste durch einen hohen Aufwand nur wenig beeindrucken lassen.

Die Ausschreibung führte bis zum Abgabeschluß am 15. Juni 1998 zu fünfzehn Vorschlägen aus aller Welt, darunter auch einem der Deutschen Telekom. Diese wurden in der ersten AES-Konferenz vom 20.–22. August 1998 in Ventura, Kalifornien vorgestellt und der Fachwelt zur Analyse und Kommentierung empfohlen.

Die zweite AES-Konferenz am 22. und 23. April 1999 in Rom führte zu einer ersten Diskussion der Ergebnisse und Empfehlungen, welche der fünfzehn Algorithmen weiter betrachtet werden sollten. Die endgültige Entscheidung des NIST wurde am 9. August 1999 bekanntgegeben:

Bei fünf der eingereichten Algorithmen hatte sich herausgestellt, daß sie entweder mit einem Aufwand zu knacken sind, der deutlich unter der Durchsuchung des gesamten Schlüsselraums liegt (so z.B. der Algorithmus der Deutschen Telekom) oder aber, daß es zu viele „schwache“ Schlüssel gibt. Da diese fünf Algorithmen auch mit die langsamsten Kandidaten waren, sprach nichts dafür, sie noch weiter zu betrachten.

Fünf weitere Kandidaten zeigten ebenfalls Schwächen, die zwar für sich allein betrachtet nicht so schwerwiegend waren, daß man die Verfahren eliminieren müßte; da diese fünf Kandidaten andererseits keine entscheidenden Vorteile an anderer Stelle boten, wurden auch sie eliminiert, so daß noch fünf Finalisten übrigblieben: Drei aus USA (MARS von IBM, RC6 von RSA und Twofish von Counterpane), Serpent von drei Kryptologen aus England, Israel und Norwegen, und Rijndael von zwei Kryptologen, die von der Katholischen Universität Leuven in Belgien kommen.

Bei der dritten AES-Konferenz am 13. und 14. April 2000 in New York bekam Rijndael 86 Stimmen, Serpen 59, Twofish 31, RC6 23 und MARS 13, so daß es nicht weiter verwunderte, daß das NIST am 2. Oktober 2000 Rijndael für den vorgeschlagenen Standard nominierte. Der offizielle Standard wird voraussichtlich im Sommer 2001 vorliegen.

Rijndael hat seinen Namen von seinen beiden Autoren JOAN DAEMEN und VINCENT RIJMEN, die 1995 beziehungsweise 1997 an der Elektrotechnischen Fakultät der Katholischen Universität Leuven über Themen aus der Kryptographie promoviert hatten; Rijmen arbeitet immer noch dort als Postdoc. Aufgrund der Wahl von Rijndael zum AES wurden sie als flämische Persönlichkeiten des Jahres 2000 gewählt.

Als Aussprachehilfe für Personen, die kein Niederländisch, Flämisch, Surinamer oder Afrikaans sprechen, geben die Autoren folgende englischen Approximationen des Wortes „Rijndael“: „Reign Dahl“, „Rain Doll“ und „Rhine Dahl“.

§2: Algebraische Vorbereitungen

Alle Operationen von Rijndael sind algebraisch definiert. Es gibt einmal Operationen auf Byte-Ebene, die durch die Grundrechenarten im Körper mit 256 Elementen realisiert sind; dazu kommen Operationen auf 4-Byte-Wörtern, die im Polynomring über diesem Körper definiert sind. Wir wollen uns langsam an die Definition dieses Körpers und dieses Ringes herantasten; im Hinblick auf Algorithmen der *public key*-Kryptographie, die uns später in der Vorlesung interessieren werden, ist die Präsentation geringfügig allgemeiner, als es für Rijndael notwendig wäre.

a) Der Euklidische Algorithmus

Wir werden den EUKLIDischen Algorithmus im folgenden hauptsächlich für Polynome brauchen; zum besseren Verständnis der Idee (und für spätere Anwendungen in der *public key*-Kryptographie) betrachten wir ihn aber zunächst im wohlbekannten Fall der natürlichen Zahlen.

Ziel ist die Berechnung des größten gemeinsamen Teilers (ggT) zweier natürlicher Zahlen a und b, also der größten natürlichen Zahl d, die sowohl a als auch b teilt; wir schreiben kurz

$$d = \operatorname{ggT}(a, b) .$$

Grundidee des EUKLIDischen Algorithmus ist die Anwendung der Division mit Rest: Für je zwei natürliche Zahlen x und y gibt es nichtnegative ganze Zahlen q und r, so daß

$$x = qy + r \qquad \text{und} \qquad 0 \le r < y$$

ist. Alsdann ist

$$\operatorname{ggT}(x, y) = \operatorname{ggT}(y, r) ,$$

denn wegen der beiden Gleichungen

$$x = qy + r \qquad \text{und} \qquad r = x - qy$$

teilt jeder gemeinsame Teiler von x und y auch r, und jeder gemeinsame Teiler von y und q teilt x.

Der EUKLIDische Algorithmus nutzt dies aus, um die Zahlen, deren ggT bestimmt werden muß, sukzessive zu verkleinern, bis der ggT zweier Zahlen berechnet werden muß, von denen die eine Teiler der anderen ist; in diesem Fall ist natürlich die kleinere der beiden Zahlen gleich dem ggT.

Formal sieht der EUKLIDische Algorithmus zur Berechnung des ggT zweier natürlicher Zahlen a und b folgendermaßen aus:

Schritt 0: Setze $r_0 = a$ und $r_1 = b$

Schritt i, $i \ge 1$**:** Falls $r_i = 0$ ist, endet der Algorithmus mit $\operatorname{ggT}(a, b) = r_{i-1}$; andernfalls dividiere man r_{i-1} mit Rest durch r_i mit dem Ergebnis $r_{i-1} = q_i r_i + r_{i+1}$.

(Bei einer tatsächlichen Implementierung bieten sich hier natürlich einige offensichtliche Optimierungen an.)

Der Algorithmus muß nach endlich vielen Schritten enden, denn bei der Division mit Rest ist stets $0 \le r_{i+1} < r_i$, so daß r_i mit jedem Schritt

kleiner wird, was bei natürlichen Zahlen nicht unbegrenzt möglich ist. Da außerdem in jedem Schritt

$$\operatorname{ggT}(r_{i-1}, r_i) = \operatorname{ggT}(r_i, r_{i+1})$$

ist und im letzten Schritt, wenn r_{i-1} den vorigen Wert r_{i-2} teilt,

$$\operatorname{ggT}(r_{i-1}, r_{i-2}) = r_{i-1}$$

ist, folgt induktiv

$$\operatorname{ggT}(a, b) = r_{i-1} ,$$

so daß der Algorithmus das richtige Ergebnis liefert.

Wir brauchen für das folgende vor allemm eine Erweiterung des Grundalgorithmus, den sogenannten erweiterten EUKLIDischen Algorithmus: Die Gleichung

$$r_{i-1} = q_i r_i + r_{i+1}$$

läßt sich umschreiben als

$$r_{i+1} = q_i r_i - r_{i+1} ,$$

so daß r_{i+1} eine ganzzahlige Linearkombination von r_i und r_{i-1} ist. Da entsprechend auch r_i Linearkombination von r_{i-1} und r_{i-2} ist, folgt induktiv, daß der ggT von a und b als ganzzahlige Linearkombination von a und b dargestellt werden kann.

Algorithmisch sieht diese Erweiterung folgendermaßen aus:

Schritt 0: Setze $r_0 = a$, $r_1 = b$, $\alpha_0 = \beta_1 = 1$ und $\alpha_1 = \beta_0 = 0$. Mit $i = 1$ ist dann

$$r_{i-1} = \alpha_{i-1} a + \beta_{i-1} b \qquad \text{und} \qquad r_i = \alpha_i a + \beta_i b .$$

Diese Relationen werden in jedem der folgenden Schritte erhalten:

Schritt i, $i \ge 1$**:** Falls $r_i = 0$ ist, endet der Algorithmus mit

$$\operatorname{ggT}(a, b) = r_{i-1} = \alpha_{i-1} a + \beta_{i-1} b .$$

Andernfalls dividiere man r_{i-1} mit Rest durch r_i mit dem Ergebnis

$$r_{i-1} = q_i r_i + r_{i+1} .$$

Dann ist

$$r_{i+1} = q_i r_i - r_{i-1} = q_i(\alpha_i a + \beta_i b) - (\alpha_{i-1}a + \beta_{i-1}b)$$
$$= (q_i\alpha_i - \alpha_{i-1})a + (q_i\beta_i - \beta_{i-1})b\,;$$

man setze also

$$\alpha_{i+1} = q_i\alpha_i - \alpha_{i-1} \qquad \text{und} \qquad \beta_{i+1} = q_i\beta_i - \beta_{i-1}\,.$$

Genau wie oben folgt, daß der Algorithmus für alle natürlichen Zahlen a und b endet und daß am Ende der richtige ggT berechnet wird; außerdem sind die α_i und β_i so definiert, daß in jedem Schritt $r_i = \alpha_i a + \beta_i b$ ist, insbesondere ist also im letzten Schritt der ggT als Linearkombination der Ausgangszahlen dargestellt.

Als Beispiel wollen wir den ggT von 200 und 148 als Linearkombination darstellen. Im nullten Schritt haben wir 200 und 148 als die trivialen Linearkombinationen.

Im ersten Schritt dividieren wir, da 148 nicht verschwindet, 200 mit Rest durch 148:

$$200 = 1 \cdot 148 + 52 \qquad \text{und} \qquad 52 = 1 \cdot 200 - 1 \cdot 148\,.$$

Da auch $52 \neq 0$, dividieren wir im zweiten Schritt 148 durch 52:

$148 = 2\cdot 52 + 44$ und $44 = 148 - 2\cdot(1\cdot 200 - 1\cdot 148) = 3\cdot 148 - 2\cdot 200\,.$

Auch $44 \neq 0$, wir dividieren also weiter: $52 = 1 \cdot 44 + 8$ und

$8 = 52 - 44 = (1\cdot 200 - 1\cdot 148) - (3\cdot 148 - 2\cdot 200) = 3\cdot 200 - 4\cdot 148\,.$

Im nächsten Schritt erhalten wir $44 = 5 \cdot 8 + 4$ und

$4 = 44 - 5\cdot 8 = (3\cdot 148 - 2\cdot 200) - 5\cdot(3\cdot 200 - 4\cdot 148) = 23\cdot 148 - 17\cdot 200\,.$

Bei der Division von acht durch vier schließlich erhalten wir Divisionsrest null; damit ist vier der ggT von 148 und 200 und kann in der angegebenen Weise linear kombiniert werden.

b) Körper von Primzahlordnung

Ist p eine Primzahl und $0 < a < p$ eine natürliche Zahl, so ist der ggT von a und p natürlich gleich eins. Also gibt es ganze Zahlen α, β, so daß

$$\alpha a + \beta p = 1 \qquad \text{oder} \qquad \alpha a - 1 = \beta p$$

ist. Damit ist $\alpha a - 1$ durch p teilbar oder, anders ausgedrückt

$$\alpha a \equiv 1 \bmod p\,.$$

Somit ist α also ein multiplikatives Inverse zu a.

Wenn wir auf der Menge $\mathbb{F}_p \underset{\text{def}}{=} \{0, 1, \ldots, p-1\}$ eine Addition und Multiplikation einführen durch die Vorschriften

$$a \oplus b \underset{\text{def}}{=} (a+b) \bmod p \qquad \text{und} \qquad a \odot b \underset{\text{def}}{=} ab \bmod p$$

definieren, ist klar, daß – genau wie in den ganzen Zahlen – das Kommutativ- und das Assoziativgesetz sowohl für die $\oplus$ als auch für $\odot$ gilt, und auch das Distributivgesetz folgt sofort aus dem für $\mathbb{Z}$. Bezüglich $\oplus$ ist die Null neutrales Element und $p - a$ invers zu a; bezüglich $\odot$ ist die Eins neutrales Element, und wie wir gerade gesehen haben, gibt es zu jedem $a \neq 0$ ein Inverses. Somit ist $\mathbb{F}_p$ ein Körper.

Das Rechnen in diesem Körper ist größtenteils einfach: Die Addition kann auf die gewöhnliche Addition in $\mathbb{Z}$ zurückgeführt werden; da $a + b$ für $a, b \in \mathbb{F}_p$ zwischen Null und $2p - 2$ liegt, ist

$$a \oplus b = \begin{cases} a + b & \text{falls } a + b < p \\ a + b - p & \text{sonst} \end{cases}\,.$$

Bei der Multiplikation ist die Situation nicht ganz so einfach; hier braucht man eine Division mit Rest, um $a \oplus b$ zu berechnen.

Das additive Inverse ist, wie bereits erwähnt, einfach $p - a$; die Berechnung des multiplikativen Inversen erfordert einen erweiterten EUKLIDischen Algorithmus und ist damit die rechenaufwendigste Operation in $\mathbb{F}_p$.

Bei der Kryptographie mit öffentlichen Schlüssel werden uns Körper $\mathbb{F}_p$ zu großen Primzahlen p interessieren; hier, für AES, brauchen wir nur den Fall $p = 2$, in dem natürlich alles trivial ist:

$\oplus$	0	1
0	0	1
1	1	0

$\odot$	0	1
0	0	0
1	0	1

c) Der ggT zweier Polynome

Nun sei k ein Körper; in den folgenden Anwendungen wird k oft der Körper mit zwei Elementen sein, gelegentlich aber der noch zu definierende Körper mit 256 Elementen. Weiter seien

$$A = x^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$$

und

$$B = x^m + b_{m-1}x^{m-1} + \cdots + b_1x + b_0$$

Polynome mit Koeffizienten a_i, b_i aus k; wir bezeichnen

$$n = \deg A \qquad \text{und} \qquad m = \deg B$$

als die Grade von A und B.

Dann läßt sich bekanntlich das Polynom A mit Rest durch B dividieren, d.h. man kann Polyonome Q, R bestimmen, für die

$$A = QB + R \quad \text{ist mit} \quad \deg R < \deg B\,.$$

Mit dieser Division lassen sich sowohl der gewöhnliche als auch der erweiterte EUKLIDische Algorithmus sofort verallgemeinern auf Polynome; da der Grad von R kleiner ist als der von B und Grade als nichtnegative ganze Zahlen nicht unbegrenzt kleiner werden können, folgt daß der Algorithmus auch für Polynome stets nach endlich vielen Schritten endet.

Das Ergebnis kann allerdings in manchen Fällen unerwartet ausfallen: Betrachten wir etwa über dem Körper $\mathbb{Q}$ der rationalen Zahlen die beiden Polynome

$$P = X^8 + X^6 - 3X^4 - 3X^3 + 8X^2 + 2X - 5$$

und

$$Q = 3X^6 + 5X^4 - 4X^2 - 9X + 21\,.$$

Division von P durch Q führt auf den Quotienten $X^2/3 - 2/9$ und Divisionsrest

$$R_2 = -\frac{5}{9}X^4 + \frac{1}{9}X^2 - \frac{1}{3}\,.$$

Division von Q durch R_2 ergibt

$$R_3 = -\frac{117}{25}X^2 - 9X + \frac{441}{25}\,,$$

bei der Division von R_2 durch R_3 bleibt Rest

$$R_4 = \frac{233150}{6591}X - \frac{102500}{2197}\,,$$

und bei der letzten Divison verbleibt als Rest der ggT

$$R_5 = \frac{1288744821}{543589225}\,.$$

Da beide Ausgangspolynome ganzzahlige Koeffizienten haben, erscheint ein ggT mit einem so großen Nenner seltsam. In der Tat ist jedes Polynom durch jede von Null verschiedene Konstante teilbar; ist also ein Polynom P Teiler eines Polynoms Q, so ist auch jedes von Null verschiedene skalare Vielfache von P Teiler von Q. Somit können wir nicht sinnvoll von *dem* größten gemeinsamen Teiler zweier Polynome reden:

Als erstes haben wir noch nicht definiert, wann ein Polynom größer sein soll als ein anderes. Das ist aber kein Problem, denn wir können einfach sagen, P sei größter gemeinsamer Teiler von A und B, wenn P ein gemeinsamer Teiler ist und jeder andere gemeinsamen Teiler ein Teiler von P ist.

Der größte gemeinsame Teiler, den uns der EUKLIDISCHE Algorithmus für Polynome liefert, hat diese Eigenschaft, denn da dieser ggT als Linearkombination von A und B geschrieben werden kann, muß jedes Polynom, das sowohl A als auch B teilt, auch den ggT teilen.

Problematischer ist, daß es viele solche größten gemeinsamen Teiler gibt: Zumindest jedes von Null verschiedene skalare Vielfache eines ggT ist selbst einer. Zum Glück ist das aber auch schon alles, was passieren kann: Sind nämlich P und Q zwei größte gemeinsame Teiler von A und B, so muß nach Definition P ein Teiler von Q sein und umgekehrt. Da der Grad eines Teilers stets kleiner oder gleich dem des Polynoms ist, haben die beiden also insbesondere denselben Grad, und ihr Quotient, egal in welcher Reihenfolge, hat Grad null und ist somit eine Konstante.

Der größte gemeinsame Teiler zweier Polynome über einem Körper ist also eindeutig bis auf Multiplikation mit einer nichtverschwindenen Konstanten; diese Konstante kann nach Belieben gewählt werden und wird meist so gewählt, daß das Ergebnis in irgendeinem Sinne einfach wird.

Auf das obige Beispiel angewendet heißt das, daß mit

$$R_5 = \frac{1288744821}{543589225}$$

auch eins ein ggT von A und B ist und man daher im allgemeinen sagen würde, „der" ggT von A und B sei eins. Es ist ein wohlbekanntes (und umgehbares) Problem der Computeralgebra, daß der EUKLIDische Algorithmus diese einfache Lösung in einer so komplizierten Form liefert; da wir aber vor allem Polynome über endlichen Körpern benötigen, braucht uns das nicht weiter zu kümmern.

d) Endliche Körper von Primzahlpotenzordnung

In Abschnitt *b)* hatten wir die ganzen Zahlen modulo p zu einem Körper gemacht; der einzige nichttriviale Schritt dabei war die Existenz des multiplikativen Inversen, die wir aus der linearen Kombinierbarkeit des ggT folgerten und daraus, daß der ggT einer Zahl mit einer Primzahl gleich eins ist, falls die Zahl kein Vielfaches der Primzahl ist.

Genauso wollen wir jetzt Körper definieren, indem wir Polynome über einem festen Körper k modulo einem vorgegebenen Polynom P betrachten: Für ein beliebiges Polynom A über k ist $A \bmod P$ gleich dem Rest bei der Division von A durch P.

Falls A kleineren Grad als P hat, ist natürlich einfach $A \bmod P = A$; zum konkreten Rechnen können wir daher ausgehen vom Vektorraum V aller Polynome vom Grad höchstens d, wobei $d + 1$ der Grad von P ist. Die Addition ist die gewöhnliche Addition von Polynomen, das Nullpolynom ist Neutralelement, und $-A$ ist invers zu A.

Das Produkt AB zweier Polynome $A, B \in V$ kann größeren Grad als d haben; wir setzen daher

$$A \odot B = AB \bmod P\,;$$

dies ist ein Polynom vom Grad höchstens d, und es ist klar, daß die so definierte Multiplikation kommutativ und assoziativ ist und das Distributivgesetz erfüllt. Das konstante Polynom 1 ist Neutralelement auch bezüglich dieser Multiplikation.

Ein inverses Polynom zu A ist ein Polynom B, für das $A \odot B = 1$ ist, d.h.

$$AB = 1 + CP \qquad \text{oder} \qquad AB + CP = 1$$

für ein geeignetes Polynom C. Zu vorgegebenen Polynomen A und P gibt es solche Polynome B und C genau dann, wenn der ggT von A und P gleich eins ist; alsdann lassen sich B und C nach dem EUKLIDischen Algorithmus berechnen.

Wenn wir möchten, daß jedes Polynom A, dessen Grad kleiner als $\deg P$ ist, ein Inverses hat, müssen wir sicherstellen, daß A und P immer teilerfremd sind; dies ist offensichtlich genau dann der Fall, wenn P keinen nichttrivialen Teiler hat, keinen Teiler also, dessen Grad größer als eins und kleiner als $\deg P$ ist. Ein solches Polynom heißt *irreduzibel.*

Falls es ein irreduzibles Polynom P vom Grad n mit Koeffizienten aus k gibt, läßt sich der Vektorraum k^n also zu einem Körper machen, indem wir ein n-tupel $(a_0, \ldots, a_{n-1})$ mit dem Polynom

$$a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \cdots + a_1X + a_0$$

identifizieren und die Multiplikation als Multiplikation von Polynomen modulo P erklären.

Bekanntestes Beispiel ist $k = \mathbf{R}$: Für $n = 2$ gibt es irreduzible Polynome vom Grad n, beispielsweise das Polynom $P = X^2 + 1$. Da

$$\begin{aligned}(a_1X + a_0)(b_1X + b_0) &= a_1b_1X^2 + (a_0b_1 + a_1b_0)X + a_0b_0 \\ &\equiv (a_0b_1 + a_1b_0)X + (a_0b_0 - a_1b_1) \bmod X^2 + 1\end{aligned}$$

ist, folgt

$$(a_0, a_1) \odot (b_0, b_1) = (a_0b_0 - a_1b_1, a_0b_1 + a_1b_0)\,,$$

wir erhalten also den Körper der komplexen Zahlen. Weitere Beispiele über $\mathbf{R}$ gibt es nicht, denn ein irreduzibles reelles Polynom muß entweder Grad eins oder Grad zwei haben, und da jedes irreduzible quadratische

Polynom zwei konjugiert komplexe Nullstellen hat, entstehen dabei immer die komplexen Zahlen – lediglich die Basis über **R** ändert sich.

Über endlichen Körpern ist die Situation etwas komplizierter: Hier gibt es für *jedes* n mindestens ein irreduzibles Polynom vom Grad n, allerdings gilt auch hier, daß zwei irreduzible Polynome desselben Grads auf den gleichen Körper führen. Da wir diese Ergebnis im folgenden nicht brauchen, sei auf einen Beweis verzichtet; Interessenten finden ihn in den üblichen Lehrbüchern der Algebra. Es gibt keinen einfachen Ausdruck für ein irredduzibles Polynom vom Grad n über einem vorgegebenen endlichen Körper k; in der Computeralgebra behilft man sich meist damit, daß man so lange zufällige Polynome vom Grad n erzeugt, bis man ein irreduzibles gefunden hat.

e) Der Körper mit 256 Elementen

Für AES ist der Körper $\mathbb{F}_{256}$ von zentraler Bedeutung; da $256 = 2^8$ ist, handelt es sich hier um den Vektorraum $\mathbb{F}_2^8$. Die Addition ist sehr einfach: Da die Addition in $\mathbb{F}_2$ mit dem exklusiven Oder übereinstimmt, ist die Addition in $\mathbb{F}_{256}$ das bitweise exklusive Oder, eine Operation, die nicht nur in den gängingen CPUs, sondern auch in vielen Prozessoren für spezielle Anwendungen in der Signalverarbeitung als Grundoperation implementiert und somit sehr schnell ist.

Um eine Multiplikation zu definieren, brauchen wir ein irreduzibles Polynom vom Grad acht über $\mathbb{F}_2$. Da $\mathbb{F}_2$ ein sehr kleiner Körper und acht eine ziemlich kleine Zahl ist, hat zumindest ein Computer keinerlei Schwierigkeiten, alle diese Polynome zu bestimmen: Nach der allgemeinen Theorie endlicher Körper, für die wieder auf Algebralehrbücher verwiesen werden muß, sind das genau die irreduziblen Faktoren vom Grad acht in der Faktorisierung des Polynoms $X^{255} - 1$ über $\mathbb{F}_2$. Faktorisierung von Polynomen über Körpern von Primzahlordung ist einer der Grundalgorithmen der Computeralgebra und geht auch noch bei sehr viel komplizierteren Polynomen sehr schnell; hier zeigt das Ergebnis, daß es dreißig Faktoren vom Grad acht gibt. Diese führen zwar alle auf denselben Körper, aber das praktische Rechnen in diesem Körper hängt natürlich stark von der Wahl des Polynoms ab. Insbesondere wird die Geschwindigkeit umso höher, je weniger Terme das Polynom hat.

Dreizehn der dreißig Polynome bestehen aus sieben nichtverschwindenden Termen, die restlichen siebzehn aus fünf; Wir wählen natürlich eines der letzteren. Alle diese Polynome haben, wie jedes Polynom vom Grad acht über $\mathbb{F}_2$, den führenden Term X^8; danach folgen vier weitere Terme. Bei der Reduktion modulo einem solchen Polynom $P = X^8 + \mathit{Rest}$ benutzt man, daß dann

$$X^8 \equiv \mathit{Rest}, \qquad X^9 \equiv X \cdot \mathit{Rest}, \quad \ldots$$

ist; dies wird umso häufiger mehrfach angewandt werden müssen, je höheren Grad die Terme in *Rest* haben. Am effizientesten kann man also rechnen, wenn das Polynom *Rest* den kleinstmöglichen Grad hat, und wenn zudem auch noch die hinteren Terme von *Rest* möglichst kleinen Grad haben. Inspektion der siebzehn Polynome mit fünf Termen zeigt, daß das Polynom

$$m(X) = X^8 + X^4 + X^3 + X + 1$$

in dieser Hinsicht optimal ist.

Die genaue Festlegung für das Rechnen in $\mathbb{F}_{256} = \mathbb{F}_2^8$ für die Zwecke von AES ist folgende: Wir schreiben ein Byte als $(a_7, a_6, \ldots, a_0)$ und identifizieren es mit dem Polynom

$$a_7X^7 + a_6X^6 + \cdots + a_1X + a_0\,;$$

das Byte 0000 0010 entspricht also X.

Der Einfachheit halber schreiben wir Bytes meist als zweiziffrige Hexadezimalzahlen: Im betrachteten Beispiel wäre das 01_{hex}, und das Byte $\text{A5}_{\text{hex}} = 1010\,0101$ entspricht dem Polynom $X^7 + X^5 + X^2 + 1$.

Man beachte, daß trotz dieser Schreibweise die Addition und Multiplikation in $\mathbb{F}_{256}$ natürlich nichts mit der Addition und Multiplikation von Hexadezimalzahlen zu tun haben. Zwar ist $01_{\text{hex}} + 02_{\text{hex}} = 03_{\text{hex}}$, aber $01_{\text{hex}} + 01_{\text{hex}} = 00_{\text{hex}}$ und $05_{\text{hex}} + 04_{\text{hex}} = 01_{\text{hex}}$.

Zur Berechnung von $\text{A5}_{\text{hex}} \odot 01_{\text{hex}}$ müssen wir das Polynom

$$(X^7 + X^5 + X^2 + 1) \cdot X = X^8 + X^6 + X^3 + X$$

berechnen und modulo $m(X)$ reduzieren. Da

$$X^8 \bmod m(X) = X^4 + X^3 + X^2 + 1$$

ist (in $\mathbb{F}_2$ ist $-1 = 1$), ist dies

$$(X^4 + X^3 + X^2 + 1) + (X^6 + X^3 + X) = X^6 + X^4 + X^2 + X + 1 .$$

Somit ist $\mathrm{A5}_{\mathrm{hex}} \odot 01_{\mathrm{hex}} = 56_{\mathrm{hex}}$.

Trotz der Wahl des optimalen Polynoms ist die Multiplikation als immer noch erheblich aufwendiger als die Addition.

§3: Spezifikation von Rijndael

a) Terminologie und Bezeichnungen

Rijndael arbeitet mit verschiedenen Blocklängen sowie auch verschiedenen Schlüssellängen: Beide können (unabhänging voneinander) die Werte 128, 192 und 256 annehmen; als „Erweiterung“ sind auch Längen von 160 und 224 Bits spezifiziert. In jedem Fall sind also Blöcklänge und Schlüssellänge Vielfache von 32, die zwischen 128 und 256 liegen. Wir schreiben die Blocklänge als $32N_b$ und die Schlüssellänge als $32N_k$; die Zahlen N_b und N_k liegen also jeweils zwischen vier und acht, wobei als „offizielle“ Werte nur 4, 6 und 8 spezifiziert sind.

Rijndael verschlüsselt somit einen Block von $32N_b$ Bit oder $4N_b$ Bytes, und genau wie DES geschieht dies sukzessive in mehreren Runden. In der Terminologie von Rijndael wird der Block in seinem jeweiligen Bearbeitungsstand als „Zustand“ bezeichnet; die einzelnen Runden finden also jeweils einen bestimmten Zustand vor und verändern diesen.

Die Anzahl der Runden wird als N_r bezeichnet; sie hängt von N_b und N_k ab gemäs der Gleichung

$$N_r = \max(N_b, N_k) + 6 .$$

b) Die Grundoperationen

Auch wenn Rijndael nicht mehr nach dem Prinzip eines FEISTEL-Netzwerks arbeitet, sind in den einzelnen Runden immer noch die klassischen SHANNONschen Prinzipien von Konfusion und Diffusion realisiert.

Für die Konfusion sind bei DES die verschiedenen S-Boxen zuständig; bei Rijndael gibt es nur eine einzige Funktion zu diesem Zweck; diese ist definiert als Hintereinanderausführung der Bildung des (multiplikativen) Inversen in $\mathbb{F}_{256}$ mit einer über $\mathbb{F}_2$ affinen Abbildung von $\mathbb{F}_{256}$ nach $\mathbb{F}_{256}$.

Die Inversenbildung ist natürlich nur für $\mathbb{F}_{256} \setminus \{0\}$ erklärt; um trotzdem eine bijektive Abbildung von $\mathbb{F}_{256}$ nach $\mathbb{F}_{256}$ zu bekommen, nehmen wir die einzig mögliche Fortsetzung, bilden die Null also auf sich selbst ab. Auch wenn es algebraisch kompletter Unsinn ist, wollen wir zur Vermeidung von Fallunterscheidungen für die Definition von Rijndael die Kurzschreibweise $0^{-1} = 0$ verwenden mit der Interpretation, daß $x \mapsto x^{-1}$ die Fortsetzung der Inversenbildung zu einer bijektiven Abbildung von $\mathbb{F}_{256}$ nach $\mathbb{F}_{256}$ sein soll.

Diese Abbildung ist weder über $\mathbb{F}_{256}$ noch über $\mathbb{F}_2$ linear, und sie ist das einzige nichtlineare Element von Rijndael. Rechnerisch ist die Bestimmung von x^{-1} aufwendig, allerdings gibt es nur 256 mögliche Werte für x, die man vorausberechnen und in 256 Byte abspeichern kann; dieser Speicherbedarf dürfte selbst für Smartcards problemlos sein.

Diffusion wird durch eine Reihe von $\mathbb{F}_{256}$-linearen Abbildungen erzielt, die Operationen sind also im Gegensatz zu den Bit-Permutationen von DES allesamt auf Byte-Niveau definiert; die Multiplikation von $\mathbb{F}_{256}$ sorgt dafür, daß trotzdem auch eine beträchtliche Diffusion innerhalb der Bytes stattfindet. Durch Tabellen gegebene Permutationen gibt es in AES überhaupt nicht; alle Diffusion wird durch Shift-Operationen sowie durch eine algebraische Operation realisiert.

Letztere arbeitet mit Wörtern von vier Byte, die wiederum mit Polynomen identifiziert werden, jetzt aber nicht, wie bei der Definition der Multiplikation in $\mathbb{F}_{256}$, mit Polynomen über $\mathbb{F}_2$, sondern solchen über $\mathbb{F}_{256}$. Wir schreiben die neuen Polynome daher zur besseren Unterscheidung in einer neuen Variablen Y und identifizieren F_{256}^4 somit mit dem Vektorraum aller Polynome vom Grad höchstens drei in Y mit Koeffizienten aus $\mathbb{F}_{256}$, indem wir das Quadrupel (b_0, b_1, b_2, b_3) auffassen als Polynom

$$b_3Y^3 + b_2Y^2 + b_1Y + b_0 \qquad \text{mit} \qquad b_i \in \mathbb{F}_{256} .$$

Diese Polynome multiplizieren wir modulo dem Polynom $M = Y^4 + 1$.

Man beachte, daß dieses Polynom nicht irreduzibel über $\mathbb{F}_2$ ist: Da alle Binomialkoeffizienten außer dem ersten und dem letzten gerade sind, ist

$$(Y + 1)^4 = Y^4 + 1 \bmod 2 .$$

Mit der hier definierten Multiplikation wird F_{256}^4 also nicht zu einem Körper. Rijndael verwendendet diese Multiplikation allerdings nur für eine einzige lineare Abbildung von $\mathbb{F}_{256}^4$ nach $\mathbb{F}_{256}^4$, und diese ist gegeben durch Multiplikation mit dem Polynom

$$C = 03_{\text{hex}} Y^3 + Y^2 + Y + 02_{\text{hex}} .$$

An der Stelle $Y = 1$ ist

$$C = 03_{\text{hex}} + 1 + 1 + 02_{\text{hex}} = 03_{\text{hex}} + 02_{\text{hex}} = 1 ,$$

das Polynom ist also nicht durch $Y + 1$ teilbar (zur Erinnerung: über $\mathbb{F}_2$ wie auch $\mathbb{F}_{256}$ ist $Y + 1 = Y - 1$), und damit auch teilerfremd zu $Y^4 + 1 = (Y + 1)^4$. Damit hat zumindest dieses Polynom ein multiplikatives Inverses, das man mit dem EUKLIDischen Algorithmus über $\mathbb{F}_{256}$ auch berechnen kann – wenn es von Hand auch ziemlich aufwendig ist. Im Maple-*worksheet* zu diesem Paragraphen findet man die detailierte Rechnung mit Unterprogrammen für die Rechenoperationen von $\mathbb{F}_{256}$; als Ergebnis findet man das inverse Polynom

$$C' = 0\text{B}_{\text{hex}} X^3 + 0\text{D}_{\text{hex}} X^2 + 09_{\text{hex}} X + 0\text{E}_{\text{hex}} .$$

Wer will, kann nachrechnen, daß $C \cdot C'$ modulo M gleich eins ist, allerdings erfordert bereits das recht viele Multiplikationen in $\mathbb{F}_{256}$.

c) Der Aufbau der Runden

Nach diesen Vorbereitungen können wir uns mit dem Aufbau der einzelnen Runden beschäftigen. Abgesehen von einer leichten Modifikation bei der letzten Runde besteht jede Runde aus denselben vier Schritten

1. Bytesubstitution
2. Zeilenshift
3. Spaltenmix
4. Addition des Rundenschlüssels

1.) Die Bytesubstitution: Dies ist der Konfusionsschritt; er operiert auf Bytes und wird auf jedes einzelne Byte des Zustands in derselben Weise angewendet; mit geeigneter Hardware kann dies also auch parallel erfolgen. Da es nur 256 mögliche Byte gibt, wird man diese Operation im allgemeinen über eine Tabelle implementieren; der Speicherbedarf von 256 Byte sollte auch auf einer Smartcard im allgemeinen problemlos sein,

Der erste Schritt ist, wie bereits erwähnt, die Inversenbildung im Körper $\mathbb{F}_{256}$; danach folgt eine Diffusion auf Bitniveau, indem $\mathbb{F}_{256}$ als affiner Raum $\mathbb{F}_2^8$ interpretiert wird und die affine Abbildung

$$\vec{x} \mapsto M\vec{x} + \vec{b}$$

folgt mit

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \text{und} \quad \vec{b} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} .$$

Betrachtet man $\mathbb{F}_{256}$ nicht als affinen Raum, sondern als Raum der Polynome vom Grad höchstens sieben, kann man die affine Abbildung auf interpretieren als

$$P \mapsto (X^7 + X^6 + X^5 + X^4 + 1)P + (X^7 + X^6 + X^2 + X) \bmod X^8 + 1 ;$$

da $X^7 + X^6 + X^5 + X^4 + 1$ an der Stelle eins den Wert eins annimmt, ist dieses Polynom teilerfremd zu $X^8 + 1 = (X + 1)^8$, die Abbildung ist also invertierbar – was sie natürlich auch sein muß, damit die Chiffre entschlüsselbar ist.

Die Matrix M ist somit invertierbar, und die Umkehrabbildung hat als affine Abbildung die Form

$$\vec{y} \mapsto M^{-1}\vec{y} + M^{-1}\vec{b} ;$$

mit den üblichen Algorithmen der linearen Algebra rechnet man leicht nach, daß

$$M^{-1} = \begin{pmatrix} 0&0&1&0&0&1&0&1\\ 1&0&0&1&0&0&1&0\\ 0&1&0&0&1&0&0&1\\ 1&0&1&0&0&1&0&0\\ 0&1&0&1&0&0&1&0\\ 0&0&1&0&1&0&0&1\\ 1&0&0&1&0&1&0&0\\ 0&1&0&0&1&0&1&0 \end{pmatrix} \quad \text{und} \quad M^{-1}\vec{b} = \begin{pmatrix} 1\\0\\1\\0\\0\\0\\0\\0 \end{pmatrix}$$

ist.

Die Inversenabbildung ist natürlich (auch mit der Konvention, daß die Null auf sich selbst abgebildet wird) zu sich selbst invers; die Bytesubstitution wird also rückgängig gemacht, indem man ein Byte $\vec{y}$ zunächst auf $A^{-1}\vec{y} + A^{-1}\vec{b}$ abbildet und dann die erweiterte Inversenabbildung von $\mathbb{F}_{256}$ anwendet.

Betrachtet man $\mathbb{F}_{256}$ nicht als affinen Raum, sondern als Raum der Polynome vom Grad höchstens sieben, kann man die affine Abbildung auch interpretieren als

$$P \mapsto (X^7 + X^6 + X^5 + X^4 + 1)P + (X^7 + X^6 + X^2 + X) \bmod X^8 + 1\ ;$$

2.) Die Zeilenshifts: Der Diffusionsschritt ist zweigeteilt: Der zu verschlüsselnde Block ist ein Vektor aus $4N_b$ Bytes; er wird umgeschrieben in eine Byte-Matrix mit vier Zeilen und N_b-Spalten, die spaltenweise aufgefüllt wird. Wird die Matrix als $A = (a_{ij})$ bezeichnet, wobei der Zeilenindex i von 0 bis 3 und der Spaltenindex j von 0 bis $N_b - 1$ geht, ist der ursprüngliche Vektor also (in Zeilenschreibweise)

$$(a_{00}, a_{10}, a_{20}, a_{30}, a_{01}, a_{11}, \ldots, a_{N_b-2,3}, a_{N_b-1,0}, \ldots, a_{N_b-1,3})\ .$$

Indiziert man die Komponenten des Vektors durch einen Index n linear von 0 bis $4N_b - 1$, ist also

$$i = n \bmod 4, \quad j = \left[\frac{n}{4}\right] \qquad \text{und} \qquad n = i + 4j\ .$$

Der erste Diffusionsschritt ist eine zyklische Verschiebung der Zeilen von A: Die 0-te Zeile wird überhaupt nicht verschoben und die erste

um eine Stelle; die Richtung der Verschiebung ist, wie auch stets im folgenden, nach links. Bei den beiden unteren Reihen hängt die Weite der Verschiebung von N_b ab: Für $N_b \neq 8$ wird die zweite Zeile um zwei Stellen verschoben, für $N_b = 8$ und drei. Die dritte Zeile wird für $N_b \leq 6$ um drei Stellen, für $N_b \leq 7$ um drei Stellen verschoben. Mit entsprechender Harware können die Verschiebungen der einzelnen Zeilen natürlich parallel durchgeführt werden.

3.) Der Spaltenmix: Auf Spaltenebene ist die Diffusion aufwendiger: Ein Spaltenvektor ist ein Element von $\mathbb{F}_{256}^4$. Diesen Vektorraum hatten wir oben mit den kubischen Polynomen über $\mathbb{F}_{256}$ identifiziert und dort eine Multiplikation eingeführt über die Polynommultiplikation modulo $Y^4 + 1$. Genau dies verwendet der Spaltenmix, indem jeder Spaltenvektor mit dem Polynom

$$C = 03_{\text{hex}}Y^3 + Y^2 + Y + 02_{\text{hex}}$$

multipliziert wird. Wegen der einfachen Struktur des Polynoms $Y^4 + 1$ ist dies eine sehr einfache lineare Abbildung mit Matrix

$$\begin{pmatrix} 02_{\text{hex}} & 03_{\text{hex}} & 01_{\text{hex}} & 01_{\text{hex}}\\ 01_{\text{hex}} & 02_{\text{hex}} & 03_{\text{hex}} & 01_{\text{hex}}\\ 01_{\text{hex}} & 01_{\text{hex}} & 02_{\text{hex}} & 03_{\text{hex}}\\ 03_{\text{hex}} & 01_{\text{hex}} & 01_{\text{hex}} & 02_{\text{hex}} \end{pmatrix}$$

bezüglich der Basis $\{1, X, X^2, X^3\}$. Diese kann für die N_b Spalten parallel durchgeführt werden.

Im Vergleich zum Zeilenshift ist diese Operation relativ aufwendig, da mehrere Multiplikationen in $\mathbb{F}_{256}$ durchgeführt werden müssen. In der letzten Runde, in der dieser Diffusionschritt keiner anschließenden Konfusion mehr unterworfen wird und somit keinen großen Sicherheitsgewinn mehr bietet, wird daher auf diesen Schritt verzichtet.

Wie wir bereits gesehen haben, ist C modulo $Y^4 + 1$ invertierbar mit inversem Polynom

$$C' = 0\text{B}_{\text{hex}}X^3 + 0\text{D}_{\text{hex}}X^2 + 09_{\text{hex}}X + 0\text{E}_{\text{hex}}\ ;$$

Multiplikation mit C' ist als lineare Abbildung gegeben durch die Matrix

$$\begin{pmatrix} 0\mathrm{E}_{\mathrm{hex}} & 0\mathrm{B}_{\mathrm{hex}} & 0\mathrm{D}_{\mathrm{hex}} & 09_{\mathrm{hex}} \\ 09_{\mathrm{hex}} & 0\mathrm{E}_{\mathrm{hex}} & 0\mathrm{B}_{\mathrm{hex}} & 0\mathrm{D}_{\mathrm{hex}} \\ 0\mathrm{D}_{\mathrm{hex}} & 09_{\mathrm{hex}} & 0\mathrm{E}_{\mathrm{hex}} & 0\mathrm{B}_{\mathrm{hex}} \\ 0\mathrm{B}_{\mathrm{hex}} & 0\mathrm{D}_{\mathrm{hex}} & 09_{\mathrm{hex}} & 0\mathrm{E}_{\mathrm{hex}} \end{pmatrix};$$

als Übung für das Rechnen in $\mathbb{F}_{256}$ sollte man zumindest für einige Einträge nachrechnen, daß das Produkt dieser beiden Matrizen über $\mathbb{F}_{256}$ gleich der Einheitsmatrix ist.

4.) Schlüsselexpansion und Rundenschlüssel: Die bisherigen drei Schritte sorgten für Konfusion und Diffusion; sie sind aber für jeden, der das grundsätzliche Verfahren kennt, leicht rückgängig zu machen.

Dieser Schritt ist genau wie bei DES eine einfache Addition des Rundenschlüssels; die Addition ist dabei die gewöhnliche Vektoraddition in $F_2^{32N_n}$, also das logische XOR. Sicherheitsrelevant ist, genau wie bei DES, die Berechnung der Rundenschlüssel aus dem vorgegebenen Schlüssel; der Schlüssel mit seinen $4N_k$ Bytes muß so aufgebläht werden auf $N_r \times 4N_b$ Schlüsselbytes, daß ein Kryptanalytiker aus einem oder auch einigen irgendwie ermittelten Rundenschlüsseln nicht auf den Gesamtschlüssel schließen kann.

Rijndael verwendet dazu im wesentlichen dieselben Techniken wie bei den Verschlüsselungsschritten: Das erweiterte Schlüsselfeld wird aufgefaßt als eine Folge von Worten W_i mit einer Länge von vier Byte oder 32 Bit. Die ersten Worte W_0 bis W_{N_k-1} sind der eigentliche Schlüssel des Verfahrens, der Rest wird rekursiv daraus berechnet. Da die ersten N_b Worte nicht als Rundenschlüssel verwendet werden, sondern vor der ersten Runde zum Klartext addiert werden, hat das gesamte Schlüsselfeld eine Länge von $N_b(N_r + 1)$ Worten.

Für $i \geq N_k$ wird W_i sukkzessive aus dem unmittelbaren Vorgänger W_{i-1} und dem N_k Wörter zurückliegenden Vorgänger W_{i-N_k} berechnet:

- Falls i nicht durch N_k teilbar ist und im Falle $N_k > 6$ auch nicht durch vier, ist
$$W_i = W_{i-N_k} \oplus W_{i-1} ,$$
wobei $\oplus$ die Vektorraumaddition in $\mathbb{F}_2^{32} = \mathbb{F}_{256}^8$ bezeichnet, also das logische XOR für 32-Bit-Wörter.
- Falls i durch N_k teilbar ist, wird W_{i-1} zunächst zyklisch um eins nach links verschoben. Dann wird auf jedes Byte des so entstandenen Worts die Bytesubstitution aus Absatz *1)* angewendet, und das Ergebnis wird mit $\oplus$ zu W_{i-N_k} addiert. Dazu wird noch eine Rundenkonstante addiert, deren erstes Byte dasjenige Element von $\mathbb{F}_{256}$ ist, das der Potenz X^{i/N_k} modulo dem die Multiplikation definierenden Polynom entspricht und dessen weitere drei Bytes alle Null sind.
- Falls $N_k > 6$ und $i \equiv 4 \bmod N_k$, wird die Bytesubstitution auf W_{i-1} angewendet und das Ergebnis zu W_{i-N_k} addiert.

d) Gesamtablauf von Rijndael

Nachdem nun alle Einzelheiten spezifiziert sind, kann der Gesamtablauf von Rijndael leicht angegeben werden:

1. Die ersten N_b Worte des Schlüsselfelds werden zum Klartext addiert.
2. $N_r - 1$ Runden werden gemäß obiger Beschreibung durchgeführt.
3. Die N_r-te Runde wird entsprechend ausgeführt, aber ohne den Spaltenmix.

e) Geschwindigkeitsoptimierung

So wie Rijndael spezifiziert ist, ist vor allem die Bytesubstitution sehr langsam; aber wie bereits erwähnt, läßt sie sich leicht über eine Tabelle implementieren.

Ein weiterer rechnerisch aufwendiger Bestandteil von Rijndael sind die Multiplikationen im Körper $\mathbb{F}_{256}$. Eine vorausberechnete Multiplikationstabelle würde $256 \times 256 = 64 \times 1024$ Byte oder 64 Kilobyte in Anspruch nehmen, was erstens etwas viel und zweitens auch noch optimal ist: Die Multiplikation von $\mathbb{F}_{256}$ wird nur beim Spaltenmix benötigt, und da das hierzu verwendete Polynom C nur $01_{\mathrm{hex}}, 02_{\mathrm{hex}}$ und 03_{hex} als Koeffizienten hat, reicht es für die Verschlüsselung, wenn man die Produkte mit 02_{hex} und 03_{hex} bilden kann. Der jeweils andere Faktor des Produkts ist stets das Ergebnis einer Bytesubstitution, man spart also

Rechenzeit, wenn man für jedes Byte das Ergebnis der Bytesubstitution sowie dessen Produkt mit 02_{hex} und mit 03_{hex} abspeichert; mit 768 Byte für Tabellen kann man also zur Laufzeit auf alle Multiplikationen und Divisionen in $\mathbb{F}_{256}$ verzichten.

Mit vier Kilobyte Tabellenplatz läßt sich sogar die gesamte Rundentransformation auf $4N_b$ XOR-Operationen auf 32-Bit-Wörtern zurückführen: Man speichere für jedes Byte $a \in \mathbb{F}_{256}$ die vier 32-Bit-Wörter

$$T_0(a) = \begin{pmatrix} 02_{\text{hex}} \odot S(a) \\ S(a) \\ S(a) \\ 03_{\text{hex}} \odot S(a) \end{pmatrix}, \quad T_1(a) = \begin{pmatrix} 02_{\text{hex}} \odot S(a) \\ 03_{\text{hex}} \odot S(a) \\ S(a) \\ S(a) \end{pmatrix},$$

$$T_2(a) = \begin{pmatrix} S(a) \\ 03_{\text{hex}} \odot S(a) \\ 02_{\text{hex}} \odot S(a) \\ S(a) \end{pmatrix} \quad \text{und} \quad T_3(a) = \begin{pmatrix} S(a) \\ S(a) \\ 03_{\text{hex}} \odot S(a) \\ 02_{\text{hex}} \odot S(a) \end{pmatrix},$$

wobei $S: \mathbb{F}_{256} \to F_{256}$ die Bytesubstitution ist.

Ist dann (bei Matrizenschreibweise) $\vec{b}_j$ der j-te Spaltenvektor des Zustands zu Beginn einer Runde, $\vec{e}_j$ der entsprechende Vektor am Ende der Runde und $\vec{k}_j$ der j-te Spaltenvektor des Rundenschlüssels, so ist

$$\vec{e}_j = T_0(b_{0j}) \oplus T_1(b_{1,j\ominus 1}) \oplus T_2(b_{2,j\ominus c_2}) \oplus T_3(b_{3,j\ominus c_3}) \oplus \vec{k}_j\,;$$

dabei stehen c_2 und c_3 die Beträge, um die die zweite und dritte Zeile verschoben werden, d.h.

$$c_2 = \begin{cases} 2 & \text{für } N_b < 8 \\ 3 & \text{für } N_b = 8 \end{cases} \quad \text{und} \quad c_3 = \begin{cases} 2 & \text{für } N_b < 7 \\ 3 & \text{für } N_b \geq 7 \end{cases},$$

und $\ominus$ steht für die Subtraktion modulo N_b, wie man sie für die Zeilenshifts braucht. Man beachte, daß auch diese Rechnung bei entsprechender Hardware für alle N_b Spalten parallel ausgeführt werden kann.

Da sich auch die sämtlichen Rundenschlüssel mit einem Speicheraufwand von weniger als einem halben Kilobyte vorausberechnen lassen, läßt sich die Verschlüsselung mittels Rijndael auf einem Standard-PC also sehr schnell durchführen.

Bei anderen Implementierungen, bei denen es Probleme mit dem Speicherplatz gibt, kann man auf Kosten einer höheren Rechenzeit mit sehr viel weniger Speicher auskommen. Eine relativ billige Art und Weise, wie man bei 32-Bit-Prozessoren drei Kilobyte einsparen kann, folgt beispielsweise aus der Beobachtung, daß die verschiedenen $T_i(a)$ durch zyklische Verschiebung auseinander entstehen. Somit reicht es, die $T_0(a)$ abzuspeichern, und indem man analog zum HORNER-Schema rechnet, liegt der zusätzliche Rechenaufwand nur bei drei zyklischen Vertauschungen pro Spalte und pro Runde: Ist $\mathcal{Z}$ die zyklische Linksverschiebung um ein Byte, ist

$$\vec{e}_j = \vec{k}_j \oplus T_0(b_{0j}) \oplus \mathcal{Z}\Big(T_0(b_{1,j-1}) \oplus \mathcal{Z}\big(T_0(b_{2,j-c_2}) \oplus \mathcal{Z}\left(T_0(b_{3,j-c_3})\right)\big)\Big).$$

Für die Entschlüsselung braucht man natürlich entsprechende Tabellen; hier werden die Produkte mit $0B_{\text{hex}}$, $0D_{\text{hex}}$, 09_{hex} und $0E_{\text{hex}}$ benötigt.