

# Rijndael – Nachfolger des DES

Der zukünftige Advanced Encryption Standard

Michael Welschenbach

*Der symmetrische Blockverschlüsselungs-Algorithmus Rijndael von Joan Daemen und Vincent Rijmen wurde im vergangenen Jahr vom US-amerikanischen National Institute of Standards and Technology (NIST) als Advanced Encryption Standard (AES) nominiert. AES steht damit sozusagen vor der Tür, er soll bis zum Herbst 2001 verabschiedet und als Standard in Kraft sein. Über die Eigenschaften und die Funktionsweise des neuen Algorithmus, der die Nachfolge des Data Encryption Standard (DES) antreten wird, berichtet dieser Artikel.*



Michael Welschenbach

ist Mitglied der Geschäftsleitung von SRC Security Research & Consulting GmbH, ein von der deutschen Kreditwirtschaft getragenes Unternehmen. SRC berät als unabhängiger Dienstleister zu aktuellen Themen der IT-Sicherheit.

E-Mail: [michael.welschenbach@src-gmbh.de](mailto:michael.welschenbach@src-gmbh.de)

## 1 Einleitung

Das amerikanische *National Institute of Standards and Technology* (NIST) hat 1997 einen Wettbewerb ausgeschrieben, mit dem Ziel, unter der Bezeichnung *Advanced Encryption Standard* (AES) einen neuen nationalen Standard (*Federal Information Processing Standard*, FIPS) für die Verschlüsselung mit einem symmetrischem Blockalgorithmus zu schaffen. Durch den neuen Standard soll ein Verschlüsselungsalgorithmus festgelegt werden, der allen heutigen Sicherheitsanforderungen genügt und der in allen Design- und Implementierungsaspekten öffentlich bekannt und weltweit kostenfrei nutzbar sein wird. Er soll den betagten *Data Encryption Standard* (DES) ablösen, der allerdings als *Triple DES* zunächst noch für den Einsatz im U.S.-Behördenbereich zugelassen bleiben soll. Zukünftig jedoch soll der AES die kryptographische Basis der amerikanischen Administration für den Schutz sensibler Daten sein.

Der AES-Wettbewerb hat auch außerhalb der USA große Aufmerksamkeit hervorgerufen, nicht nur, weil alles, was sich in den USA in Sachen Verschlüsselung tut, weltweit eine starke Signalwirkung entfaltet, sondern weil ausdrücklich zu internationaler Beteiligung an der Entwicklung des neuen Blockverschlüsselungsverfahrens aufgerufen wurde.

## 2 Die Kandidaten

Von ursprünglich fünfzehn Kandidaten, die 1998 ins Rennen gingen, wurden unter internationaler fachlicher Beteiligung bis 1999 zehn aussortiert. Als Kandidaten waren danach noch die Algorithmen *MARS* von IBM, *RC6* von RSA Laboratories, *Rijndael* von Joan Daemen und Vincent

Rijmen, *Serpent* von Ross Anderson, Eli Biham und Lars Knudson sowie *Twofish* von Bruce Schneier et al. im Rennen. Schließlich stand im Oktober 2000 der Sieger des Auswahlprozesses fest: Der Algorithmus *Rijndael*<sup>1</sup> von Joan Daemen und Vincent Rijmen aus Belgien wurde als zukünftiger Advanced Encryption Standard nominiert (vgl. [NIST]). Rijndael ist ein Nachfolger der bereits früher veröffentlichten Blockchiffre *Square* derselben Autoren (vgl. [SQUA]), die sich jedoch als weniger stark herausgestellt hatte. Rijndael wurde speziell gegen die für *Square* entdeckten Angriffsmöglichkeiten gestärkt. Der AES-Report von NIST gibt folgende Gründe für die Entscheidung an:

**Sicherheit:** Alle Kandidaten erfüllen die Anforderungen des AES hinsichtlich der Sicherheit gegenüber allen bekannten Angriffen. Implementierungen von *Serpent* und *Rijndael* können im Vergleich zu den übrigen Kandidaten mit dem geringsten Aufwand gegen Angriffe geschützt werden, die auf Messungen des zeitlichen Verhaltens der Hardware (sogenannte Timing-Attacken) oder von Änderungen der Stromaufnahme (sogenannte Power bzw. Differential Power Analysis-Attacken<sup>2</sup>) beruhen. Die notwendigerweise mit derartigen Schutzmaßnahmen verbundenen Einbußen

<sup>1</sup> Der Name Rijndael ist aus einer Zusammenziehung der Namen der Autoren entstanden. Je nach Quelle wird die Aussprache von Rijndael in die Nähe von „*Rain-Doll*“ oder „*Rhine-Dahl*“ gestellt.

<sup>2</sup> Power Analysis-Attacken (Simple PA/Differential PA) basieren auf der Feststellung von Korrelationen zwischen einzelnen oder Gruppen von geheimen Schlüssel-Bits und der mittleren Stromaufnahme für die Ausführung von einzelnen Instruktionen oder Code-Sequenzen (vgl. u. a. [KOJJ], [CHAR], [GOBA] und [MESS]).

an Performanz wirken sich bei Rijndael, Serpent und Twofish nur gering aus, mit größeren Vorteilen für Rijndael.

**Geschwindigkeit:** Rijndael gehört zu den Kandidaten, die die schnellsten Implementierungen erlauben. Der Algorithmus zeichnet sich insbesondere durch eine gleichmäßig gute Performanz über alle betrachteten Plattformen aus, wie z. B. 32-Bit-Prozessoren, 8-Bit Mikrocontroller, die derzeit weitverbreitet in Chipkarten eingesetzt werden, und Implementierungen in Hardware. Rijndael erlaubt unter allen Kandidaten die schnellste Erzeugung von Rundenschlüsseln.

**Speicherbedarf:** Rijndael benötigt sehr geringe Ressourcen an RAM- und ROM-Speicher und ist damit hervorragend für den Einsatz in Umgebungen mit beschränkten Ressourcen geeignet. Der Algorithmus bietet insbesondere die Möglichkeit, die Rundenschlüssel für jede Runde separat zu berechnen (Schlüsselerzeugung „on-the-fly“). Alle diese Eigenschaften haben eine große Bedeutung für Anwendungen in Chipkarten. Insgesamt muss Rijndael sich hinsichtlich des Ressourcenbedarfs keinem der vier anderen Konkurrenten geschlagen geben.

**Implementierbarkeit in Hardware:** Rijndael und Serpent sind diejenigen Kandidaten mit der besten Performanz in Hardware-Ausführungen, Rijndael mit einem leichten Vorteil aufgrund besserer Ergebnisse in Output- und Cipher-Feedback-Modi.

Der Report führt noch weitere Kriterien an, die zum Gesamtergebnis beitragen, das in einer abschließenden Bewertung zusammengefasst wird ([NIST], Abschn. 7): „There are many unknowns regarding future computing platforms and the wide range of environments in which the AES will be implemented. However, when considered together, Rijndael’s combination of security, performance, efficiency, implementability, and flexibility make it an appropriate selection for the AES for use in the technology of today and in the future.“

Angesichts der Transparenz des Auswahlprozesses und der politisch interessanten Tatsache, dass mit Rijndael ein Algorithmus europäischer Provenienz ausgewählt wurde, dürfte sich künftig jede Spekulation über geheimgehaltene Eigenschaften, verborgene Falltüren und absichtlich eingebaute Schwachstellen verbieten, wie

sie im Zusammenhang mit dem DES nie gänzlich verstummt sind.

### 3 Rijndael

Rijndael ist ein symmetrischer Blockverschlüsselungsalgorithmus mit variabler Block- und Schlüssellänge: Er kann Blöcke zu 128, 192 und 256 Bit und ebensolche Schlüssellängen verarbeiten, wobei alle Kombinationen von Block- und Schlüssellängen möglich sind. Die akzeptierten Schlüssellängen entsprechen den Vorgaben für AES, die „offizielle“ Blocklänge wird voraussichtlich jedoch nur 128 Bit betragen. Jeder Klartextblock wird zur Verschlüsselung mehrere Male mit einer sich wiederholenden Abfolge verschiedener Funktionen behandelt, in sogenannten *Runden*. Die Anzahl der Runden ist von der Block- und der Schlüssellänge abhängig.

Anzahl Runden Nr	Blocklänge in Bit		
	128	192	256
Schlüssellängen in Bit	10	12	14
128	12	12	14
192	14	14	14

Tab. 1: Anzahl der Rijndael-Runden in Abhängigkeit von Block- und Schlüssellängen

Bei Rijndael handelt es sich nicht um einen sogenannten Feistel-Algorithmus, dessen wesentliches Charakteristikum darin besteht, dass Blöcke jeweils in linke und rechte Hälften geteilt werden, die Rundenfunktion auf die eine Hälfte angewendet wird und das Ergebnis zu der anderen Hälfte XORiert wird. Danach werden die beiden Hälften vertauscht. Der DES ist der bekannteste Blockalgorithmus nach diesem Prinzip. Rijndael hingegen ist aus einzelnen Schichten aufgebaut, die nacheinander unterschiedliche Wirkungen auf jeweils einen gesamten Block ausüben. Für die Verschlüsselung eines Blocks wird dieser nacheinander den folgenden Transformationen unterworfen:

1. Der erste Rundenschlüssel wird mittels XOR mit dem Block verknüpft.
2. Es werden  $Nr - 1$  reguläre Runden durchlaufen.
3. Eine abschließende Runde wird ausgeführt, in der die MixColumns-Transformation der regulären Runden ausgelassen wird.

Jede reguläre Runde des Schrittes 2 besteht aus 4 Einzelschritten:

1. **Substitution**  
Jedes Byte eines Blocks wird durch eine S-Box-Anwendung ersetzt.
2. **Permutation**  
Die Bytes des Blocks werden in der sogenannten *ShiftRows*-Transformation permutiert.
3. **Diffusion**  
Die sogenannte *MixColumns*-Transformation wird ausgeführt.
4. **Verknüpfung mit dem Schlüssel**  
Der jeweilige Rundenschlüssel mit dem Block durch XOR verknüpft.

Die Schichtung von Transformationen innerhalb einer Runde wird schematisch durch die folgende Abbildung veranschaulicht:

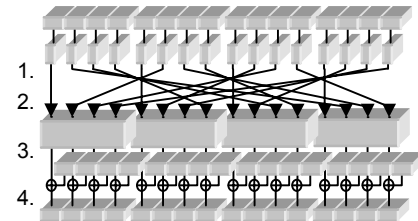


Abb. 1: Schichtung von Transformationen der Rijndael-Runden

Jede dieser Schichten übt eine bestimmte Wirkung innerhalb einer Runde und damit auf jeden Klartextblock aus:

**Einfluss des Schlüssels:** Durch die Verknüpfung mit dem Rundenschlüssel vor der ersten Runde und als letzter Schritt innerhalb jeder Runde wirkt sich dieser auf jedes Bit der Rundenergebnisse aus. Es gibt im Verlauf der Verschlüsselung eines Blocks keinen Schritt, dessen Ergebnis nicht in jedem Bit abhängig vom Schlüssel wäre.

**Nichtlineare Schicht:** Die S-Box-Substitution ist eine nichtlineare Operation. Die Konstruktion der S-Box sorgt für nahezu idealen Schutz vor differentieller und linearer Kryptoanalyse (vgl. [NIST]).

**Lineare Schicht:** Die ShiftRows- und MixColumns-Transformationen sorgen für eine optimale Durchmischung der Bits eines Blocks.

Im Folgenden bezeichne  $Nb$  bzw.  $Nk = 4, 6, \text{ oder } 8$  die Block- bzw. Schlüssellängen in 4-Byte-Wörtern.

Klartext und Chiffretext werden jeweils als Felder von Bytes ein- bzw. ausgegeben. Ein Klartextblock, übergeben als Feld  $m_0, \dots, m_{4 \cdot Nb-1}$  von Bytes, wird intern als zweidimensionales Feld *State* der Gestalt

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$	$b_{0,4}$	...	$b_{0,Nb-1}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$	$b_{1,4}$	...	$b_{1,Nb-1}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$	$b_{2,4}$	...	$b_{2,Nb-1}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	...	$b_{3,Nb-1}$

Tab. 2: Repräsentierung von Nachrichtenblöcken in dem Feld *State*

aufgefasst, wohin die Klartextbytes spaltenweise einsortiert werden ( $m_n \rightarrow b_{i,j}$ , mit  $i = n \bmod 4$  und  $j = \lfloor n/4 \rfloor$ ). Je nach Länge der Nachricht muss diese in einzelne Textblöcke unterteilt werden. Der Zugriff auf *State* innerhalb der Rijndael-Funktionen gestaltet sich je nach Operation unterschiedlich. Die S-Box-Substitution operiert byteweise, ShiftRows operiert auf den Zeilen von *State* und die Funktionen AddRoundKey und MixColumns schließlich operieren auf 4-Byte-Wörtern und greifen dazu spaltenweise auf die Werte von *State* zu. Alle Transformationen sind aus Byte-Operationen aufgebaut, was der Implementierbarkeit sehr entgegen kommt. Die einzelnen Funktionen und ihre Wirkungen werden in den folgenden Abschnitten detaillierter erläutert.

### 3.1 Berechnung der Rundenschlüssel

Ver- und Entschlüsselung erfordern die Erzeugung von jeweils  $Nr$ -vielen Rundenschlüsseln, dem sogenannten *Schlüsselplan*. Dies erfolgt durch eine Expandierung des geheimen Anwenderschlüssels, indem rekursiv abgeleitete 4-Byte-Wörter an den Anwenderschlüssel angehängt werden.

Der Schlüsselerzeugung liegen die folgenden Designziele zugrunde:

- Der Erzeugungsprozess ist nichtlinear, er ist nicht invertierbar und ohne Kenntnis des geheimen Benutzerschlüssels können eventuell bekannte Teile eines Schlüsselplans in keiner Richtung vollständig rekonstruiert werden.
- Es wird eine starke Diffusionswirkung auf Benutzerschlüssel ausgeübt.
- Die Schlüsselerzeugung ist schnell und flexibel implementierbar.

Für die Schlüssellänge von 128 Bit wird die Schlüsselerzeugung durch das folgende Schema veranschaulicht:

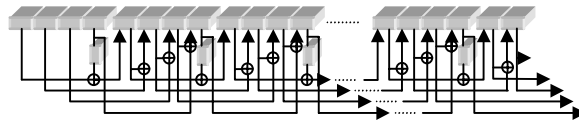


Abb. 2: Ableitungsschema für die Rundenschlüssel

Die ersten  $Nk$  Wörter  $k_0, \dots, k_{Nk-1}$  des Schlüsselplans werden durch den geheimen Anwenderschlüssel selbst gebildet. Für  $Nk = 4$  oder  $6$  wird das jeweils nächste 4-Byte-Wort  $k_i$  durch eine XOR-Verknüpfung des vorhergehenden Wortes  $k_{i-1}$  mit  $k_{i-Nk}$  bestimmt. Falls  $i \equiv 0 \pmod{Nk}$  ist, wird auf das Wort  $k_{i-1}$  vor der XOR-Verknüpfung eine Funktion  $F_{Nk}(k, i)$  angewendet, die aus einer zyklischen Links-Verschiebung (Links-Rotation)  $r(k)$  von  $k$  um ein Byte, einer Ersetzung  $S(r(k))$  aus der Rijndael S-Box und einer XOR-Verknüpfung mit einer Konstanten  $c(\lfloor i/Nk \rfloor)$  zusammengesetzt ist, so dass sich die Funktion  $F$  insgesamt zu  $F_{Nk}(k, i) := S(r(k)) \oplus c(\lfloor i/Nk \rfloor)$  ergibt.

Die Konstanten  $c(j)$  sind definiert durch  $c(j) := (rc(j), 0, 0, 0)$ , wobei die  $rc(j)$  vorberechnet und in einer Tabelle gespeichert werden können.

Für Schlüssel der Länge von 256 Bit (d. h.  $Nk = 8$ ) wird eine zusätzliche S-Box-Operation zwischengeschaltet: Falls  $i \equiv 4 \pmod{Nk}$  wird vor der XOR-Operation  $k_{i-1}$  durch  $S(k_{i-1})$  ersetzt.

Auf diese Weise werden als Schlüsselplan insgesamt  $Nb \cdot (Nr + 1)$  4-Byte-Wörter gebildet, einschließlich des geheimen Benutzerschlüssels. Für jede Runde  $i = 0, \dots, Nr-1$  werden dem Schlüsselplan die jeweils nächsten  $Nb$  4-Byte-Wörter  $k_{Nb \cdot i}$  bis  $k_{Nb \cdot (i+1)}$  als Rundenschlüssel entnommen. Die Rundenschlüssel werden in Analogie zur Strukturierung der Nachrichtenblöcke als zweidimensionales Feld aufgefasst:

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	...	$k_{0,Nb-1}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	...	$k_{1,Nb-1}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	...	$k_{2,Nb-1}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	...	$k_{3,Nb-1}$

Tab. 3: Repräsentierung von Rundenschlüsseln

Die Schlüsselerzeugung wird durch den im folgenden angegebenen Pseudocode präzisiert:

```

KeyExpansion (byte CipherKey,
word ExpandedKey)
{
  for (i = 0; i < Nk; i++)
    ExpandedKey[i] =
      (CipherKey[4*i],
       CipherKey[4*i + 1],
       CipherKey[4*i + 2],
       CipherKey[4*i + 3]);
  for (i = Nk; i < Nb*(Nr+1); i++)
    {
      temp = ExpandedKey[i - 1];
      if (i % Nk == 0)
        temp = SubBytes (RotBytes
(temp)) ^ Rcon[i/Nk];
      else if ((Nk == 8) && (i % Nk
== 4))
        temp = SubBytes (temp);
      ExpandedKey[i] =
ExpandedKey[i - Nk] ^ temp;
    }
}
    
```

Es sind keine sogenannten schwachen Schlüssel bekannt, deren Verwendung das Verfahren schwächen würde.

### 3.2 Die S-Box

Die Substitutionsbox oder S-Box des Rijndael-Algorithmus gibt an, wie in jeder Runde jedes Byte eines Blocks durch einen anderen Wert zu ersetzen ist. Die S-Box besteht aus einer Liste von 256 Bytes, die konstruiert werden, indem zunächst jedes Byte außer der Null, aufgefasst als Vertreter von  $\mathbb{F}_{2^8}$ , durch sein multiplikatives Inverses ersetzt wird (die Null bleibt am Platz). Danach wird eine affine Abbildung über  $\mathbb{F}_2$  als Matrixmultiplikation und Addition von  $(11000110)$  berechnet:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

In dieser Darstellung bezeichnet  $x_0$  bzw.  $y_0$  das jeweils niedrigstwertige und  $x_7$  bzw.  $y_7$  das jeweils höchstwertige Bit eines Byte, dem 8-Tupel (11 000110) entspricht somit der Hexadezimalwert '63'.

Der Konstruktion der S-Box liegen Designkriterien zugrunde, wonach die Anfälligkeit für die Methoden der linearen und der differentiellen Kryptoanalyse sowie für algebraische Attacken minimiert werden.

### 3.3 Die ShiftRows-Transformation

Der nächste Schritt in der Abfolge einer Runde besteht in der Permutation eines Blocks auf Byteebene. Dazu werden die Bytes innerhalb der einzelnen Zeilen (ausgenommen die erste) eines Blocks zyklisch verschoben:

vor ShiftRows	nach ShiftRows
0 4 8 12	0 4 8 12
1 5 9 13	5 9 13 1
2 6 10 14	10 14 2 6
3 7 11 15	15 3 7 11

Tab. 4: ShiftRows für  $Nb = 4$

### 3.4 Die MixColumns-Transformation

Nach der zeilenweisen Permutation im vorhergehenden Schritt wird in diesem Schritt jede Spalte eines Blocks als Polynom über  $\mathbb{F}_{2^8}$  aufgefasst und mit dem konstanten Polynom  $a(x) := a_3x^3 + a_2x^2 + a_1x + a_0$  mit Koeffizienten  $a_0(x) = x$ ,  $a_1(x) = 1$ ,  $a_2(x) = 1$  und  $a_3(x) = x + 1$  multipliziert und modulo  $M(x) := x^4 + 1$  reduziert. Dieser Schritt kann sehr effizient implementiert werden, indem jede Spalte  $(b_{i,j})_{i=0,\dots,3}$  der Zustandsvariablen  $State$  über  $\mathbb{F}_{2^8}$  mit einer Matrix multipliziert wird:

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} \leftarrow \begin{bmatrix} '02' & '03' & '01' & '01' \\ '01' & '02' & '03' & '01' \\ '01' & '01' & '02' & '03' \\ '03' & '01' & '01' & '02' \end{bmatrix} \begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix}$$

Die Multiplikation von Polynomen mit  $x$  und  $x + 1$  (bzw. '02' und '03') sind besonders schnell als Shift- und XOR-Operationen zu realisieren. Beispielsweise kann die Multiplikation eines Polynoms  $f$  aus  $\mathbb{F}_{2^8}[X]$  mit  $x + 1$  (bzw. '03') und die anschließende

Reduzierung modulo  $m(x) := x^8 + x^4 + x^3 + x + 1$  erfolgen, indem die Binärzahl  $a$  aus den Koeffizienten von  $f$  um eine Stelle nach links geschoben mit sich selbst und danach bei Auftreten eines Übertrags mit '011B' XORiert wird. Zwei Zeilen in C veranschaulichen die Vorgehensweise:

```
a ^= a << 1;
if (a & 0x100) a ^= 0x11B;
```

Durch die MixColumns-Transformation tritt jedes Byte einer Spalte in Wechselwirkung mit jedem anderen Byte der Spalte. Die vorangegangene zeilenweise operierende ShiftRows-Transformation bewirkt, dass in jeder Runde andere Bytes miteinander vermischt werden, wodurch insgesamt eine starke Diffusionswirkung erreicht wird. Weitere Designkriterien sind die Invertierbarkeit und die Performanz der Transformation.

Zur Invertierung der MixColumns-Transformation wird jede Spalte  $(b_{i,j})$  eines Blocks mit dem Polynom  $r(x) := r_3x^3 + r_2x^2 + r_1x + r_0$  mit Koeffizienten  $r_0(x) = x^3 + x^2 + x$ ,  $r_1(x) = x^3 + 1$ ,  $r_2(x) = x^3 + x^2 + 1$  und  $r_3(x) = x^3 + x + 1$  multipliziert und modulo  $M(x) := x^4 + 1$  reduziert. Die korrespondierende Matrix lautet

$$\begin{bmatrix} '0E' & '0B' & '0D' & '09' \\ '09' & '0E' & '0B' & '0D' \\ '0D' & '09' & '0E' & '0B' \\ '0B' & '0D' & '09' & '0E' \end{bmatrix}$$

Die bei der Invertierung auftretenden Multiplikationen werden am schnellsten über zwei Tabellen ausgeführt, die in die zum einen die Potenzen eines Generatorpolynoms  $g(x)$  von  $\mathbb{F}_{2^8}$  und zum anderen die Logarithmen der Werte von  $\mathbb{F}_{2^8}$  zur Basis  $g(x)$  gespeichert sind.

### 3.5 AddRoundKey

Als letzter Schritt einer Runde wird der jeweilige Rundenschlüssel  $(k_{0,j}, k_{1,j}, k_{2,j}, k_{3,j})$  mit dem in  $State$  repräsentierten Block durch XOR verknüpft: Für  $j = 0, \dots, Nb-1$  wird

$$(b_{0,j}, b_{1,j}, b_{2,j}, b_{3,j}) \leftarrow (b_{0,j}, b_{1,j}, b_{2,j}, b_{3,j}) \oplus (k_{0,j}, k_{1,j}, k_{2,j}, k_{3,j})$$

gesetzt.

## 3.6 Die Verschlüsselung eines Blocks als Gesamtablauf

Die einzelnen Operationen der vorangegangenen Abschnitte werden in der Rundenfunktion `Round` zusammengefasst:

```
Round (word State, word RoundKey)
{
    SubBytes (State);
    ShiftRows (State);
    MixColumns (State);
    AddRoundKey (State, RoundKey)
}
```

Die letzte Runde ohne `MixColumns` wird in `FinalRound` ausgeführt:

```
FinalRound (word State, word RoundKey)
{
    SubBytes (State);
    ShiftRows (State);
    AddRoundKey (State, RoundKey)
}
```

Die Verschlüsselung mit Rijndael im Gesamtablauf wird durch den folgenden Pseudocode nach [DAR1] veranschaulicht:

```
Rijndael (byte State, byte CipherKey)
{
    KeyExpansion (CipherKey, ExpandedKey);
    AddRoundKey (State, ExpandedKey);
    for (i = 1; i < Nr; i++)
        Round (State, ExpandedKey + Nb*i);
    FinalRound (State, ExpandedKey + Nb*Nr);
}
```

Es besteht die Möglichkeit, die Erzeugung der Rundenschlüssel außerhalb der Funktion `Rijndael` vorzubereiten und den Schlüsselplan `ExpandedKey` anstelle des Anwenderschlüssels `CipherKey` zu übergeben. Dies ist vorteilhaft, wenn für die Verschlüsselung von Nachrichten, die länger als ein Block sind, mehrere Aufrufe von `Rijndael` mit dem gleichen Anwenderschlüssel erforderlich werden.

Für 32-Bit-Prozessoren ist es günstig, die Rudentransformationen sowohl für die Ver- als auch für die Entschlüsselung vorzuberechnen und die Ergebnisse als Tabellen zu speichern. Das Ersetzen der Permutations- und Matrixoperationen durch Zugriffe auf Tabellen erspart einen beträchtlichen Rechenaufwand und wirkt sich dementspre-

chend günstig auf den Durchsatz der Ver- bzw. Entschlüsselung aus.

### 3.7 Entschlüsselung

Die Entschlüsselung erfordert es, die Operationen zur Verschlüsselung in umgekehrter Reihenfolge mit jeweils inversen Transformationen `InvSubBytes`, `InvShiftRow`, und `InvMixColumns` zu durchlaufen. Die algebraische Struktur von Rijndael gestattet es, die Transformationen für die Entschlüsselung so zu arrangieren, dass auch hier mit Tabellen gearbeitet werden kann. Hierzu ist zu beobachten, dass die Substitution  $S$  und die `InvShiftRows`-Transformation kommutativ sind, dass also innerhalb einer Runde deren Reihenfolge vertauschbar ist. Aufgrund der Homomorphieeigenschaft  $f(x + y) = f(x) + f(y)$  linearer Abbildungen können auch die `InvMixColumns`-Transformation und die Addition des Rundenschlüssels vertauscht werden, wenn `InvMixColumns` vorher auf den Rundenschlüssel angewendet wurde. Innerhalb einer Runde ergibt sich der folgende Ablauf:

```
InvRound (word State, word
RoundKey)
{
    InvSubBytes (State);
    InvShiftRows (State);
    InvMixColumns (State);
    AddRoundKey (State,
InvMixColumns (RoundKey));
}

InvFinalRound (word State, word
RoundKey)
{
    InvSubBytes (State);
    InvShiftRows (State);
    AddRoundKeys (State, RoundKey);
}
```

Die Gesamtoperation zur Entschlüsselung eines Blocks lautet hiermit:

```
InvRijndael (byte State, byte
CipherKey)
{
    KeyExpansion (CipherKey,
ExpandedKey);
    AddRoundKey (State, ExpandedKey
+ Nb*Nr);
    for (i = Nr - 1; i > 0; i--)
        Round (State, ExpandedKey +
Nb*i);
    InvFinalRound (State,
ExpandedKey);
}
```

Für diese Form der Entschlüsselung können analog zur Verschlüsselung Tabellen vorberechnet werden.

### 3.8 Performance

Implementierungen für verschiedene Plattformen haben die überragende Performanz von Rijndael bestätigt. Die Bandbreite reicht dabei von kleinen 8-Bit-Controllern mit wenig Speicher und Schlüsselerzeugung on-the-fly bis zu sehr leistungsfähigen 32-Bit-Prozessoren. Bei ausreichend verfügbarem Speicherplatz kann zudem eine weitgehende Beschleunigung durch Tabellen erreicht werden, die den Zugriff auf die vorberechneten Ergebnisse der verschiedenen Transformationen ermöglichen. Zum Vergleich sind in der folgenden Tabelle Verschlüsselungs-Rechenzeiten von RC6, Rijndael und Twofish sowohl für den 8051-Controller als auch für die Advanced Risc Machine (ARM) als moderner 32-Bit-Smart Card Controller angegeben (für 128 Bit lange Schlüssel):

	8051 (3,57 MHz)	ARM (28,56 MHz)
RC6	165	151.260
Rijndael	3.005	311.492
Twofish	500*	56.289

Tab. 5: Rijndael-Performanz im Vergleich (in Byte/sec nach [KOEU], \*interpoliert)

Aufgrund der komplexeren `InvMixColumns`-Operation können je nach Implementierung die Rechenzeiten für die Ver- und Entschlüsselung unterschiedlich ausfallen, wobei dieser Effekt durch die Verwendung von Tabellen gänzlich kompensiert werden kann. Zum Vergleich: Auf einem Pentium III/200 MHz ist für 128 Bit-Schlüssel in beiden Richtungen ein Durchsatz von über 8 MByte/sec erzielbar [GLAD]. Natürlich hängen die Rechenzeiten auch von der jeweiligen Schlüssellänge bzw. von der Anzahl der zu durchlaufenden Runden ab (vgl. Tab. 1).

### 3.9 Betriebsarten

Neben den klassischen Betriebsarten für Blockchiffren, wie etwa Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB) (vgl. [FIPS81]) oder Message Authentication Codes (MAC), werden zur Zeit einige neue Vorschläge für weitere Betriebsarten unter den Gesichtspunkten Si-

cherheit, Fehlertoleranz, Synchronisationseigenschaften und Implementierbarkeit diskutiert. Neue Vorschläge bieten zusätzlichen Integritätsschutz oder setzen AES als Schlüsselstromgenerator ein.

Ebenfalls in die Diskussion einbezogen wird die Verwendung der Betriebsarten in der Internet-Kommunikation, etwa im Rahmen von IPSEC und IKE (Internet Key Exchange-Protocol).

Im weiteren Vorgehen wird NIST wohl erst einmal einige der Standardbetriebsarten (u.a. ECB, CBC, CFB) im Rahmen eines eigenen FIPS-Standard für die AES-Betriebsarten festlegen, zusätzliche Modi werden möglicherweise später dazukommen. Im August 2001 findet zunächst ein zweiter öffentlicher Workshop statt, der weitere Klarheit zum diesem Thema schaffen soll.

## 4 Fazit und Ausblick

„... a fast, flexible and elegant cipher“ – so positiv fasst das IBM MARS-Team die Eigenschaften von Rijndael zusammen. Angesichts der Tatsache, dass die fünf AES-Kandidaten hinsichtlich der meisten Anforderungen sehr dicht beieinander liegen, kann dies als eine hohe Anerkennung gelten. Rijndael ist ein modernes Verschlüsselungssystem, das allen aktuellen Sicherheitsanforderungen genügt und vielfältige Einsatzmöglichkeiten bietet:

- Rijndael zeigt über alle Plattformen hinweg eine gleichmäßig exzellente Performanz,
- Rijndael erlaubt eine besonders schnelle Erzeugung von Rundenschlüsseln, wobei die Schlüsselerzeugung sowohl separat als auch „on-the-fly“, d. h. innerhalb jeder einzelnen Runde möglich ist,
- Rijndael benötigt nur geringe Ressourcen an RAM und ROM,
- Schutzmaßnahmen gegen Power-Analysis-Attacks (SPA/DPA) wirken sich bei Rijndael aufgrund der verwendeten Operationen (Shift, XOR und Tabellen) nicht besonders ungünstig auf die Performanz aus (was fairerweise auch für Serpent und Twofish festzuhalten ist),
- Rijndael besitzt insgesamt eine hervorragende Eignung für Chipkarten.

Bruce Schneier hat geschrieben, das einzige Problem an Rijndael sei die Schwierigkeit der Aussprache des Namens. Auf einen Nenner gebracht: Rijndael macht einfach Spaß.

Die Zeitplanung von NIST sieht vor, das FIPS-Dokument für den AES bis zum Sommer 2001 fertig zu stellen und bis Oktober 2001 als Standard zu verabschieden und in Kraft zu setzen. Mit einer schnellen Einführung des neuen Standards in der Praxis ist allerdings eher nicht zu rechnen, da die Migration auf das neue Verfahren die Umstellung der Blocklänge von 8 auf 16 Byte beinhaltet. Nachdem jahrzehntelang die Blocklänge auf 8 Byte festgeschrieben war, dürfte die Anpassung auf 16 Byte umfangreiche Software-Änderungen in zahlreichen (zahllosen?) Applikationen nach sich ziehen. Die Schlüssellängen des AES und das hierfür notwendige Schlüsselmanagement dürften nicht ganz so umfangreiche Modifikationen erfordern, da bereits seit einiger Zeit für den Triple-DES 128- und in Ausnahmefällen auch 192-Bit-Schlüssel eingesetzt werden. Insgesamt ist jedenfalls eine längere Migrationsdauer zu erwarten, an deren Ende allerdings der allgemeine Umstieg auf ein modernes, flexibles und hoffentlich auf Jahre hin für alle möglichen Anwendungen ausreichend sicheres Verschlüsselungssystem vollzogen sein wird.

Für weitere Einzelheiten, Untersuchungen zur Sicherheit und zur Kryptoanalyse, Rechenzeiten und aktuelle Information zu AES und Rijndael wird auf die zitierte Literatur sowie auf die Internetseiten von NIST und von Vincent Rijmen verwiesen, die jeweils viele Links zu weiteren Informationsquellen anbieten:

[http://csrc.nist.gov/encryption/aes/aes\\_home.htm](http://csrc.nist.gov/encryption/aes/aes_home.htm)  
<http://www.esat.kuleuven.ac.be/~rijmen/rijndael>

## Literatur

- [AES] National Institute of Standards and Technology: „Announcing the ADVANCED ENCRYPTION STANDARD (AES)“, Draft Federal Information Processing Standard, NIST, 2001.
- [CHAR] Chari, Suresh, Charanjit Jutla, Josyula R. Rao, Pankaj Rohatgi: „A Cautionary Note Regarding Evaluation of AES Candidates on Smart Cards“, <http://csrc.nist.gov/encryption/aes/round1/conf2/papers/chari.pdf>, 1999.
- [DARI] Daemen, Joan, Vincent Rijmen: „AES-Proposal: Rijndael“, Doc. Vers. 2.0, <http://www.nist.gov/encryption/aes>, Sept. 1999.
- [FIPS81] National Institute of Standards and Technology: „DES Modes of Operation“, Federal Information Processing Standard 81, NIST, 1980.
- [GLAD] Gladman, Brian: „A Specification for Rijndael, the AES Algorithm“, [http://fp.gladman.plus.com/cryptography\\_technology/rijndael/](http://fp.gladman.plus.com/cryptography_technology/rijndael/), 2001.
- [GOBA] Goubin, Louis, Jacques Patarin: „DES and Differential Power Analysis“, in Proceedings of CHES '99, Lecture Notes in Computer Science, Vol. 1717, Springer-Verlag, 1999.
- [IBM] IBM Mars Team: „Mars and the AES selection criteria“, IBM, 2000.
- [KOEU] Koeune, F., G.Hachez, J.-J. Quisquater: „Implementation of Four AES Candidates on Two Smart Cards“, UCL Crypto Group, 2000.
- [KOJJ] Kocher, Paul, Joshua Jaffe, Benjamin Jun: „Introduction to Differential Power Analysis and Related Attacks“, 1998, <http://www.cryptogrphy.com/dpa/technical/>
- [MESS] Messerges, Thomas S.: „Securing the AES Finalists Against Power Analysis Attacks“, Fast Software Encryption Workshop 2000, Lecture Notes in Computer Science, Springer-Verlag.
- [NIST] Nechvatal, James, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, Edward Roback: „Report on the Development of the Advanced Encryption Standard“, National Institute of Standards and Technology, 2000.
- [SQUA] Daemen, Joan, Lars Knudsen, Vincent Rijmen: „The Block Cipher Square“, Fast Software Encryption, Lecture Notes in Computer Science 1267, Springer-Verlag, 1997, S. 149–165.
- [SCHN] Schneier, Bruce, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson: „The Twofish Encryption Algorithm“, Wiley Computer Publishing, 1999.
- [WELS] Welschenbach, Michael: „Cryptography in C and C++“, Springer-Verlag New York, APress CA, 2001.