10 Der Advanced Encryption Standard (AES)

Das momentan bedeutendste symmetrische Verschlüsselungsverfahren ist der **Advanced Encryption Standard** (**AES**). Unter dem Namen Rijndael hat dieses Verfahren den Wettbewerb um die Nachfolge des DES als US-Verschlüsselungsstandard gewonnen (siehe Abschnitt 23.5.2). Seit November 2001 ist es in den USA offiziell standardisiert und wird dort seitdem für die Verschlüsselung staatlicher Dokumente verwendet. Zwei Jahre später ließ die US-Regierung den AES sogar für Daten bis zur höchsten Geheimhaltungsstufe (Top Secret) zu, was beim Vorgänger DES nicht der Fall gewesen war. Zum ersten Mal ist damit ein Verschlüsselungsverfahren, das für derart wichtige Informationen eingesetzt wird, öffentlich bekannt. Zahlreiche staatliche Institutionen in anderen Ländern nutzen den AES ebenfalls. Gleiches gilt für Hersteller von Hard- und Software in aller Welt. Das Verfahren ist nicht patentiert und damit frei verwendbar.

Entwickelt wurde der AES (unter der Bezeichnung Rijndael) von den beiden belgischen Kryptografen Vincent Rijmen und Joan Daemen (letzterer ist übrigens ein Mann, auch wenn der Vorname diesbezüglich in die Irre führt). Die Bezeich-

nung »Rijndael« ist an die Nachnamen der beiden Erfinder angelehnt. Die korrekte Aussprache lautet »Reindahl«. Für die Namenswahl nannten Rijmen und Daemen einen einleuchtenden Grund: Sie wollten, dass ihre Nachnamen endlich einmal richtig ausgesprochen werden. Wer sich intensiv mit dem AES beschäftigen will, sollte sich das Buch besorgen, das Daemen und Rijmen darüber geschrieben haben [DaeRij01].

10.1 Funktionsweise des AES

Der Funktionsweise des AES liegen einige Überlegungen zu Grunde, die nur mit mathematischem Grundlagenwissen zu verstehen sind. Dabei spielen endliche Körper (namentlich der Körper GF(2^8), der auch bei Twofish zum Einsatz kommt) eine zentrale Rolle. Endliche Körper sind ansonsten vor allem in der asymmetrischen Kryptografie von Bedeutung und werden daher im späteren Verlauf dieses Buchs noch näher betrachtet. Um den Ablauf einer AES-Verschlüsselung nachzuvollziehen, muss man diese mathematischen Grundlagen jedoch nicht unbedingt kennen. Ich werde die Funktionsweise des AES daher zunächst erklären, ohne auf endliche Körper einzugehen. In Abschnitt 10.2 gibt es dann einige Erklärungen zur Mathematik des Verfahrens.

Die Blocklänge des AES beträgt 128 Bit. Das Verfahren Rijndael unterstützt zwar auch 192 und 256 Bit, doch diese Blocklängen wurden nicht in die AES-Standardisierung aufgenommen (genau genommen muss man den AES daher als Spezialfall von Rijndael bezeichnen). Die AES-Schlüssellänge kann – wie bei allen AES-Kandidaten – wahlweise auf 128, 192 oder 256 Bit festgelegt werden. Die Rundenzahl ist, wie in der folgenden Tabelle dargestellt, von der Schlüssellänge abhängig:

Schlüssellänge	Rundenzahl
128 Bit	10
192 Bit	12
256 Bit	14

Der AES ist eine SP-Chiffre. Das Verfahren sieht im Wechsel Schritte zur Substitution und zur Permutation vor, ohne dass es dabei die von Feistel-Chiffren bekannte Aufteilung in linke und rechte Hälfte gibt. Ein Klartextblock (128 Bit) ist stets in Form einer 4×4-Matrix aus Byte-Variablen gegeben, wobei die 16 Bytes eines Blocks spaltenweise in die Matrix geschrieben werden. Ein Klartext (a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3, c_0, c_1, c_2, c_3, d_0, d_1, d_2, d_3) wird daher wie folgt in eine Matrix umgewandelt:

$$\begin{matrix} a_0 & b_0 & c_0 & d_0 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{matrix}$$

10.1.1 Rundenaufbau

Der AES besteht aus vier Funktionen, die jeweils mehrfach durchlaufen werden: SubBytes, ShiftRow, MixColumn und AddRoundKey. Alle vier Funktionen bearbeiten eine 4×4-Matrix, in der am Anfang des Verfahrens der Klartext und am Ende der Geheimtext steht. Der erste Schritt einer AES-Verschlüsselung besteht aus einem Aufruf von AddRoundKey. Anschließend werden alle Runden bis auf die letzte (also 9 bzw. 11 bzw. 13 Runden) nach folgendem Schema abgearbeitet:

1. *SubBytes*
2. *ShiftRow*
3. *MixColumn*
4. *AddRoundKey*

In der letzten Runde fällt der dritte Schritt (MixColumn) weg, ansonsten verläuft sie gleich. Man kann das Verfahren übrigens auch anders erklären: Alle Runden bis auf die letzte bestehen aus den Schritten 1) AddRoundKey 2) SubBytes 3) ShiftRow 4) MixColumn. In der letzten Runde wird MixColumn durch ein weiteres AddRoundKey ersetzt.

SubBytes

Die Funktion *SubBytes* realisiert eine S-Box. Sie führt also eine nichtlineare Substitution durch. Nach den im Chiffren-Design üblichen Grundsätzen (Abschnitt 7.2) ist SubBytes damit für die Konfusion zuständig. Die Arbeitsweise von SubBytes ist recht einfach: Jedes der 16 Bytes in der 4×4-Matrix wird nach derselben Ersetzungstabelle (S-Box) durch ein neues ersetzt. Die Ersetzungstabelle hat demnach für jeden Bytewert von 0 bis 255 einen Eintrag und sieht so aus (Einträge in hexadezimaler Schreibweise):

00-0F	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
10-1F	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
20-2F	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
30-3F	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
40-4F	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
50-5F	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
60-6F	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8

→→

70-7F	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
80-8F	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
90-9F	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A0-AF	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B0-BF	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C0-CF	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D0-DF	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E0-EF	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F0-FF	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Der Aufbau der S-Box ist so gestaltet, dass der AES gegenüber differenzieller und linearer Kryptoanalyse nicht anfällig ist. Eine Besonderheit der AES-S-Box besteht darin, dass man jeden der 256 Einträge durch eine einfache mathematische Formel berechnen kann. Mehr dazu gibt es im Unterkapitel über die mathematischen Hintergründe des AES (Abschnitt 10.2).

ShiftRow

In *ShiftRow* werden die Zeilen der 4×4-Matrix durcheinandergemischt. Der Sinn dieser Funktion dürfte klar sein: Nachdem SubBytes als nichtlineare Funktion für Konfusion zuständig war, sorgt ShiftRow für Diffusion durch eine (lineare) Permutation. Die folgende Tabelle zeigt links die Matrix vor dem Mischen und rechts danach:

$$\begin{matrix} a_0 & b_0 & c_0 & d_0 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{matrix} \quad => \quad \begin{matrix} a_0 & b_0 & c_0 & d_0 \\ b_1 & c_1 & d_1 & a_1 \\ c_2 & d_2 & a_2 & b_2 \\ d_3 & a_3 & b_3 & c_3 \end{matrix}$$

In Worten beschrieben bedeutet dies: Die erste Zeile wird um null, die zweite um eine, die dritte um zwei und die vierte um drei Stellen nach links rotiert. An dieser Stelle ist zu beachten, dass Rijndael bei einer Blocklänge von 192 oder 256 Bit geringfügig andere Verschiebungen vorsieht. Da wir jedoch nur den AES betrachten, der eine Blocklänge von 128 Bit vorschreibt, soll uns das nicht weiter interessieren.

MixColumn

Die Funktion *MixColumn* trägt wie ShiftRow zur Diffusion bei und ist wie diese linear. Dabei gibt es eine klare Aufteilung: Während ShiftRow für eine Durchmischung der Zeilen sorgt, führt MixColumn ein spaltenweises Mischen durch. Im Vergleich zum sehr simplen ShiftRow ist MixColumn etwas komplexer. Um es

zu verstehen, benötigen wir eine spezielle Form der Multiplikation, die wir mit dem Zeichen »•« notieren. Ist b ein Byte, dann ist diese Multiplikation wie folgt definiert:

$2 \bullet b$:= Linksshift von b, falls $b<128$
:= (Linksshift von b) $\oplus$ 00011011, falls $b>=128$
$3 \bullet b := (2 \bullet b) \oplus b$

Damit haben wir zwar nur die Multiplikation eines Bytes mit den Zahlen 2 und 3 definiert, doch das reicht aus. Beachten Sie, dass ein Linksshift keine Linksrotation ist; das erste Bit eines Bytes geht bei einem Linksshift verloren und wird nicht in das letzte Bit geschoben (stattdessen wird das letzte Bit Null gesetzt). Zur Erklärung von MixColumn gehen wir davon aus, dass a_0, a_1, a_2 und a_3 die erste Spalte der 4×4-Matrix vor dem MixColumn-Aufruf bilden. Nach Abarbeiten der Funktion bezeichnen wir die entsprechenden Werte als a_0', a_1', a_2' und a_3'. MixColumn sieht folgende Formeln vor:

$a_0' = (2 \bullet a_0) \oplus (3 \bullet a_1) \oplus a_2 \oplus a_3$
$a_1' = a_0 \oplus (2 \bullet a_1) \oplus (3 \bullet a_2) \oplus a_3$
$a_2' = a_0 \oplus a_1 \oplus (2 \bullet a_2) \oplus (3 \bullet a_3)$
$a_3' = (3 \bullet a_0) \oplus a_1 \oplus a_2 \oplus (2 \bullet a_3)$

Die entsprechenden Formeln für die zweite, dritte und vierte Spalte sehen genau gleich aus, nur dass Sie den Buchstaben a durch b, c bzw. d ersetzen müssen. Die vier MixColumn-Formeln sind im Übrigen nach einem mathematischen Modell so gewählt, dass sie eine maximale Durchmischung der vier Matrix-Spalten garantieren. Mehr dazu gibt es im Unterkapitel über die mathematischen Grundlagen des AES.

Wie oben beschrieben, ist MixColumn die einzige AES-Funktion, die nicht in allen Runden vorkommt (sie fehlt in der letzten Runde). Dafür gibt es eine einfache Begründung: Die beiden Funktionen MixColumn und AddRoundKey kann man in der Reihenfolge vertauschen, ohne dass sich am Ergebnis der Verschlüsselung etwas ändert. Gäbe es in der letzten Runde ein MixColumn, dann könnte man dieses mit dem letzten AddRoundKey vertauschen, wodurch der letzte Schritt der Verschlüsselung nicht vom Schlüssel abhängen würde. Es wäre damit kryptografisch wertlos und würde unnötige Rechenzeit verbrauchen.

AddRoundKey

In der Funktion *AddRoundKey* wird ein Subschlüssel zur 4×4-Matrix addiert (bitweises exklusives Oder). Da sowohl der erste als auch der letzte Schritt einer AES-Verschlüsselung aus einem AddRoundKey besteht, ist ein Whitening gegeben.

10.1.2 Entschlüsselung mit dem AES

Bei einer SP-Chiffre ist die Entschlüsselung nie so elegant wie bei einer Feistel-Chiffre. Dies gilt auch für den AES. Eine AES-Entschlüsselung sieht notwendigerweise vor, dass dieselben Schritte wie bei der Verschlüsselung durchlaufen werden, nur invertiert und in umgekehrter Reihenfolge. Bei ShiftRow und AddRoundKey ist die Invertierung trivial. Bei SubBytes muss Alice eine inverse Ersetzungstabelle aufbauen, was ebenfalls kein Problem darstellt, weil die S-Box entsprechend konstruiert wurde. Für MixColumn ist die Invertierung nicht ganz so naheliegend, aber durch das zu Grunde liegende mathematische Modell ebenfalls nicht schwierig. Die Umkehrung der MixColumn-Formeln für eine Spalte lautet:

$$a_0' = 14 \bullet a_0 \oplus 11 \bullet a_1 \oplus 13 \bullet a_2 \oplus 9 \bullet a_3$$
$$a_1' = 9 \bullet a_0 \oplus 14 \bullet a_1 \oplus 11 \bullet a_2 \oplus 13 \bullet a_3$$
$$a_2' = 13 \bullet a_0 \oplus 9 \bullet a_1 \oplus 14 \bullet a_2 \oplus 11 \bullet a_3$$
$$a_3' = 11 \bullet a_0 \oplus 13 \bullet a_1 \oplus 9 \bullet a_2 \oplus 14 \bullet a_3$$

Da wir die Multiplikation »•« bisher nur für die Zahlen 2 und 3 definiert haben, benötigen wir nun einige weitere Werte:

$$9 \bullet b := 2 \bullet (2 \bullet (2 \bullet b)) \oplus b$$
$$11 \bullet b := 8 \bullet b \oplus 2 \bullet b \oplus b$$
$$13 \bullet b := 8 \bullet b \oplus 4 \bullet b \oplus b$$
$$14 \bullet b := 8 \bullet b \oplus 4 \bullet b \oplus 2 \bullet b$$

10.1.3 Schlüsselaufbereitung

Wie man leicht nachrechnet, benötigt der AES einen Subschlüssel mehr, als es Runden gibt. Jeder Subschlüssel besteht aus 16 Bytes und hat dieselbe Form wie die 4×4-Matrix, die von den vier AES-Funktionen verarbeitet wird. Wir gehen an dieser Stelle von 128 Schlüssel-Bits und damit von 10 Runden aus. Wir benötigen 11 Subschlüssel zu je 16 Bytes, insgesamt also 176 Bytes.

Die AES-Schlüsselaufbereitung lässt sich am einfachsten an einem Schaubild erklären (siehe Abbildung 10–1). Die 128 Schlüssel-Bits werden zunächst in vier 32-Bit-Variablen aufgeteilt (*A*, *B*, *C* und *D*). Der erste Subschlüssel ist mit dem AES-Schlüssel identisch. Danach wird in zehn Runden jeweils ein weiterer Subschlüssel generiert. Wie in der Abbildung ersichtlich, kommen dabei zwei Funktionen (*SubWord*, *RotWord*) und eine Konstante (*Rcon*) zur Anwendung. Die beiden Funktionen nehmen jeweils vier Bytes (also 32 Bit) entgegen und geben vier Bytes wieder aus. Hier die Beschreibung der Funktionen und der Konstante:

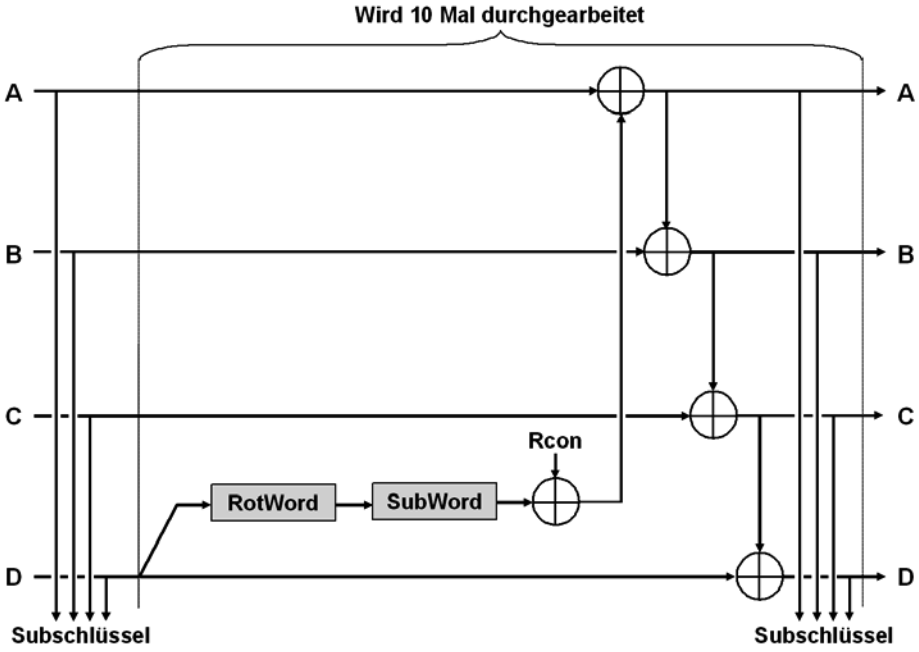

Abb. 10–1 *Die AES-Schlüsselaufbereitung generiert für die 128-Bit-Variante 11 Subschlüssel.*

- *SubWord*: Diese Funktion ersetzt jedes der vier Bytes durch den entsprechenden Eintrag in der S-Box des AES (dies ist die Ersetzungstabelle, die auch in SubBytes zum Einsatz kommt).
- *RotWord*: Diese Funktion führt eine byteweise Linksrotation der vier Bytes durch. Lautet die Eingabe b_0,b_1,b_2,b_3, dann ist die Ausgabe b_1,b_2,b_3,b_0.
- *Rcon*: Diese Konstante besteht aus vier Bytes, von denen die letzten drei stets Null sind (nur das erste Byte enthält also einen Wert). Der Wert des ersten Bytes hängt von der Runde ab (das Zeichen »•« bezeichnet die oben definierte Multiplikation):

Runde	Wert
1	1
2	2
3	2•2
4	2•2•2
5	2•2•2•2
...	...

Auch für diese Vorgehensweise gibt es mathematische Hintergründe, die Sie im folgenden Unterkapitel nachlesen können.

10.2 Mathematische Betrachtung des AES

Die Funktionsweise des AES wirkt an einigen Stellen recht willkürlich. Wenn man jedoch weiß, dass dem Verfahren mathematische Überlegungen zu Grunde liegen, auf die ich bisher nicht eingegangen bin, lässt sich alles schlüssig erklären. Um die mathematischen Grundlagen des AES zu verstehen, benötigen Sie ein Grundwissen zum Thema Galois-Felder (siehe Abschnitt 13.1.1). Ein Galois-Feld GF(p) ist ein Körper, der die ganzen Zahlen zwischen 0 und p-1 als Elemente hat. p muss eine Primzahl sein, ansonsten sind nicht alle Voraussetzungen gegeben, die ein Körper per Definition erfüllen muss. Man kann beweisen, dass es (bis auf Isomorphie) jeweils nur einen Körper mit p Elementen gibt. Man kann außerdem zeigen, dass auch zu jeder Zahl p^n genau ein Körper mit dieser Elemente-Anzahl existiert. Körper mit einer Anzahl von Elementen, die weder einer Primzahl noch einer Primzahl-Potenz entsprechen, gibt es nicht.

In GF(2^n) ist das Rechnen etwas komplizierter als in GF(p). Die Elemente in GF(2^n) werden üblicherweise nicht als natürliche Zahl, sondern als Binärzahl oder als Polynom mit den Koeffizienten 0 und 1 geschrieben. Es folgen einige Beispiele aus GF(2^4):

$$0101 \equiv x^2+1$$
$$1111 \equiv x^3+x^2+x+1$$
$$1010 \equiv x^3+x$$

Die Addition in GF(2^n) ist als bitweise Exklusiv-oder-Verknüpfung zweier Binärzahlen definiert. Bei der Polynomschreibweise entspricht dies einer Polynomaddition mit Koeffizienten aus GF(2). Hier einige Beispiele aus GF(2^4):

$$1001+1111 = 0110 \equiv (x^3+1) + (x^3+x^2+x+1) = x^2+x$$
$$1000+0001 = 1001 \equiv (x^3) + (1) = x^3 + 1$$

Mit Polynom ist im Folgenden stets ein Polynom mit Koeffizienten aus GF(2) gemeint. Die Multiplikation in GF(2^n) ist als Polynommultiplikation modulo einem irreduziblen Polynom vom Grad n definiert. Irreduzibel ist ein Polynom dann, wenn es sich nicht als Produkt aus Polynomen kleineren Grades darstellen lässt. Es gibt verschiedene Algorithmen, mit denen festgestellt werden kann, ob ein Polynom irreduzibel ist. Bei unserer Betrachtung genügt es jedoch, ein Polynom des gewünschten Grades einer Tabelle zu entnehmen. Es gibt zu jedem Grad n>1 irreduzible Polynome. In den folgenden Beispielen wird das irreduzible Polynom x^4+x+1 verwendet, um die Multiplikation in GF(2^4) zu definieren:

$$(x^3+1)\cdot(x^3+x^2+1) = x^6+x^5+x^2+1 = x^3+x^2+x+1 \pmod{x^4+x+1},$$
daher $1001 \cdot 1101 = 1111$

Für die Funktionsweise des AES spielt der endliche Körper GF(2^8) eine wichtige Rolle. Die Elemente von GF(2^8) lassen sich mit den 256 Werten eines Bytes iden-

tifizieren. Ein Byte $b_0,b_1,b_2,b_3,b_4,b_5,b_6,b_7$ kann man auch als Polynom der Form $b_0x^7+b_1x^6+b_2x^5+b_3x^4+b_4x^3+b_5x^2+b_6x+b_7$ schreiben. Um im Körper GF(2^8) multiplizieren zu können, benötigen wir ein irreduzibles Polynom vom Grad 8. Die AES-Entwickler haben $m(x)=x^8+x^4+x^3+x+1$ hierfür vorgesehen. Eine Multiplikation zweier Bytes entspricht daher immer der Multiplikation der beiden zugehörigen Polynome modulo $m(x)$. Eine Byte-Multiplikation modulo $m(x)$ wird mit dem Zeichen »•« notiert. Damit wäre geklärt, wie die in MixColumn und Rcon verwendete Multiplikation zustande kommt. Das Praktische daran ist, dass man für die Durchführung einer solchen Multiplikation keine Modulo-Polynom-Rechnung durchführen muss. Dies liegt daran, dass die AES-Entwickler das Polynom $m(x)$ so gewählt haben, dass man das Multiplizieren mit der oben beschriebenen Methode elegant abkürzen kann.

Auch die Funktion SubBytes lässt sich mithilfe der Polynom-Multiplikation modulo $m(x)$ beschreiben. Ist b der Bytewert aus der 4×4-Matrix, dann hat der zugehörige Tabelleneintrag den Wert $A\cdot b^{-1}\oplus 01100011$. Hierbei ist b^{-1} das inverse Element zu b in GF(2^8), es gilt also: $b\bullet b^{-1}=1$. A ist eine Matrix mit folgendem Inhalt:

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

Beachten Sie, dass sowohl die Matrix-Multiplikation als auch die Addition der Konstante 01100011 lineare Operationen sind. Nichtlinear ist ausschließlich die Multiplikation mit b^{-1}. Wenn Sie mehr über das S-Box-Design des AES wissen wollen, empfehle ich Ihnen einen Blick in [DaeRij01].

10.3 Sicherheit des AES

Angesichts einer Schlüssellänge von mindestens 128 Bit ist eine vollständige Schlüsselsuche beim AES ein aussichtsloses Unterfangen. Dafür war die relativ geringe Rundenzahl (je nach Schlüssellänge 10, 12 oder 14) von Anfang an umstritten. Bekannt sind Angriffe, die gegen sieben Runden bei 128 Bit Schlüssellänge, gegen acht Runden bei 192 Bit Schlüssellänge sowie gegen neun Runden bei 256 Bit Schlüssellänge funktionieren [KLSSWW]. Der Sicherheitspuffer beträgt also drei bis fünf Runden. Für ein Verfahren, das einige Jahrzehnte lang verwendet werden soll, ist dies zweifellos etwas knapp bemessen. Auch die Tatsache, dass

der AES einige damals noch wenig erforschte Designprinzipien (z. B. durch eine Formel berechenbare S-Boxen) verwendet, sorgte für Bedenken. Knapp 20 Jahre später ist jedoch nach wie vor keine Schwäche des AES bekannt, die auch nur annähernd eine praktische Bedeutung hat. Dafür sind einige recht interessante theoretische Angriffe bekannt geworden, die wir uns im Folgenden genauer anschauen wollen.

10.3.1 AES als algebraische Formel

Ein Kritikpunkt ist das mathematische Modell, das dem AES zu Grunde liegt. Dieses macht das Verfahren zwar sehr elegant, es hat aber auch Nachteile. 2001 zeigten die drei Kryptografen Niels Ferguson, Richard Schroeppel und Doug Whiting, dass sich der AES vergleichsweise gut als algebraische Formel darstellen lässt [FeScWh]. Bei einer Schlüssellänge von 128 Bit enthält die entsprechende Formel 2^{50} Bestandteile, bei 256 Schlüssel-Bits sind es 2^{70}. Diese Zahlen sind zwar nicht gerade klein. Es gibt jedoch kein anderes modernes symmetrisches Verschlüsselungsverfahren, für das eine vergleichbare Darstellung bekannt ist. 2^{50} ist eine Größenordnung, die man mit sehr viel Computeraufwand gerade noch in den Griff bekommen kann, zumal die Leistungsfähigkeit der verfügbaren Computer ständig zunimmt. Außerdem könnten neue Erkenntnisse dafür sorgen, dass man Vereinfachungsmöglichkeiten entdeckt.

Die entscheidende Frage lautet nun: Gibt es eine Möglichkeit, die algebraische AES-Formel mit vertretbarem Aufwand nach dem Schlüssel aufzulösen? Wäre dies der Fall, dann würde möglicherweise schon ein einziges Klartext-Geheimtext-Paar genügen, um den Schlüssel zu ermitteln. Bisher lautet die Antwort jedoch eindeutig nein. Noch gibt es nicht einmal ansatzweise eine Methode, um die AES-Formel zu lösen. Die Sicherheit des AES ist von dieser Seite also derzeit nicht gefährdet. Dennoch ist Vorsicht geboten. Die Lösung der AES-Formel betrifft Teilgebiete der Mathematik, die bisher keine Berührungspunkte mit der Kryptografie hatten. Es ist daher nicht auszuschließen, dass eines Tages ein kluger Kopf, der mit Kryptografie bis dahin nichts am Hut hatte, die passende Lösungsmethode präsentiert. Gut möglich, dass dieser kluge Kopf auf mathematische Erkenntnisse zurückgreift, die derzeit keiner mit diesem Problem in Verbindung bringt. Bisher hat sich allerdings noch kein kluger Kopf gemeldet.

$$a_{i,j}^{(3)} = k_{i,j}^{(2)} + \sum_{\substack{e_2 \in \mathcal{E} \\ d_2 \in \mathcal{D}}} \frac{w_{i,e_2,d_2}}{\left(k_{e_2,e_2+j}^{(1)} + \sum_{\substack{e_1 \in \mathcal{E} \\ d_1 \in \mathcal{D}}} \frac{w_{e_2,e_1,d_1}}{(a_{e_1,e_1+e_2+j}^{(1)})^{2^{d_1}}} \right)^{2^{d_2}}} \tag{4}$$

and the three-round version is

$$a_{i,j}^{(4)} = k_{i,j}^{(3)} + \sum_{\substack{e_3 \in \mathcal{E} \\ d_3 \in \mathcal{D}}} \frac{w_{i,e_3,d_3}}{\left(k_{e_3,e_3+j}^{(2)} + \sum_{\substack{e_2 \in \mathcal{E} \\ d_2 \in \mathcal{D}}} \frac{w_{e_3,e_2,d_2}}{\left(k_{e_2,e_2+e_3+j}^{(1)} + \sum_{\substack{e_1 \in \mathcal{E} \\ d_1 \in \mathcal{D}}} \frac{w_{e_2,e_1,d_1}}{(a_{e_1,e_1+e_2+e_3+j}^{(1)})^{2^{d_1}}} \right)^{2^{d_2}}} \right)^{2^{d_3}}}$$

We can now write the five-round formula

$$a_{i,j}^{(6)} = K + \sum_{\substack{e_5 \in \mathcal{E} \\ d_5 \in \mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_4 \in \mathcal{E} \\ d_4 \in \mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_3 \in \mathcal{E} \\ d_3 \in \mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_2 \in \mathcal{E} \\ d_2 \in \mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_1 \in \mathcal{E} \\ d_1 \in \mathcal{D}}} \frac{C}{K^* + p_*^*}}}}} \tag{6}$$

Abb. 10–2 *Der AES lässt sich (in sehr aufwendiger Form) als mathematische Formel darstellen. Die Abbildung zeigt lediglich einen Auszug. Noch hat niemand eine Möglichkeit gefunden, diese Formel nach dem Schlüssel aufzulösen.*

10.3.2 Quadratische Kryptoanalyse

Die einfache mathematische Darstellbarkeit des AES macht das Verfahren besonders interessant für die quadratische Kryptoanalyse (XSL-Angriff). Diese 2002 von Nicolas Courtois und Josef Pieprzyk veröffentlichte Methode sieht vor, ein Verschlüsselungsverfahren mithilfe eines quadratischen Gleichungssystems nachzubilden und dieses anschließend zu lösen (beispielsweise mit dem sogenannten XSL-Verfahren). In der Tat lässt sich der AES besser mit quadratischen Gleichungen nachbilden als die meisten anderen symmetrischen Verschlüsselungsverfahren [CouPie]. Bei einer Schlüssellänge von 128 Bit sind 8.000 Gleichungen mit 1.600 Variablen notwendig. Der Bedarf an Klartext-Geheimtext-Paaren ist gering.

Auch die quadratische Kryptoanalyse gibt jedoch bisher noch keinen Anlass zur Panik. Noch ist völlig unklar, ob die Methode überhaupt so funktioniert, wie es sich die beiden Erfinder vorstellen (nicht wenige Fachleute zweifeln daran).

Falls ja, ist man bisher noch weit davon entfernt, ein Gleichungssystem dieser Komplexität lösen zu können. Bisher ist die quadratische Kryptoanalyse daher ein sehr spekulatives Thema, das noch weiterer Forschung bedarf. AES-Miterfinder Vincent Rijmen ließ sich jedenfalls erst einmal nicht beeindrucken. Sein Kommentar: »Der XSL-Angriff ist kein Angriff. Er ist ein Traum.«

10.3.3 Biclique-Kryptoanalyse

Den bisher besten Angriff auf den AES fanden Andrey Bogdanov, Dmitry Khovratovich und Christian Rechberger im Jahr 2011 – also 14 Jahre nach Veröffentlichung des Verfahrens [BoKhRe]. Der Angriff ist eine Biclique-Kryptoanalyse, für den Mallory nur einige wenige Klartext-Geheimtext-Paare benötigt. Er ist etwa um das Vierfache schneller als die vollständige Schlüsselsuche. Präziser ausgedrückt benötigt Mallory für den Angriff $2^{126,1}$ Schritte bei einem 128-Bit-Schlüssel, $2^{189,7}$ Schritte bei 192 Bits und $2^{254,4}$ Schritte bei 256 Bits.

Die vollständige Schlüsselsuche ist somit nicht mehr der beste bekannte Angriff auf den AES. Müssen wir uns deshalb Sorgen machen? Zunächst einmal nicht, denn der Angriff von Bogdanov, Khovratovich und Rechberger ist noch weit davon entfernt, praktisch verwertbar zu sein.

10.3.4 Weitere Angriffe

In den letzten Jahren wurden einige weitere theoretische Angriffe auf den AES veröffentlicht. Dazu gehört insbesondere eine Related-Key-Attacke aus dem Jahr 2009 von Alex Biryukov and Dmitry Khovratovich [BirKho]. Dieser Angriff betrifft die 192-Bit und die 256-Bit-Version des AES. Er benötigte zunächst einen Aufwand von 2^{119} Schritten, wurde jedoch auf $2^{99.5}$ verbessert. Zwar sind diese Angriffe nicht praxisrelevant, sie sorgten jedoch wieder einmal für Diskussionen über das zwar elegante, aber nicht mit viel Sicherheitsspielraum ausgestattete Design des AES. In diesem Fall war die vergleichsweise einfache Schlüsselaufbereitung des AES die Ursache.

10.4 Bewertung des AES

Kein Zweifel, der Sieg von Rijndael beim AES-Wettbewerb kam nicht von ungefähr. Das Verfahren verfügt über ein elegantes Design, offenbarte in der Wettbewerbsphase keine Sicherheitslücken, und die Verschlüsselungsgeschwindigkeit ließ auf unterschiedlichen Plattformen nichts zu wünschen übrig. So setzte sich Rijndael in der Gunst der Experten gegen 14 Konkurrenten durch. Kaum jemand stellte diesen Sieg infrage. Allerdings bestand Einigkeit darüber, dass die Verfahren Serpent und Twofish ebenfalls würdige Gewinner gewesen wären. Am Ende war es eher eine Frage des Geschmacks, welchen der drei Algorithmen man

bevorzugte. Serpent galt als eher konservativ – ohne neue Designelemente, mit einem großen Sicherheitspuffer ausgestattet, dafür relativ langsam. Rijndael wirkte im Vergleich dazu deutlich innovativer und eleganter, verzichtete auf eine allzu große Sicherheitsmarge und überzeugte durch eine hohe Verschlüsselungsgeschwindigkeit. Twofish lag zwischen diesen beiden Extremen.

15 Jahre nach Ende des AES-Wettbewerbs stellt sich die Situation etwas anders dar. Zwar gilt der AES nach wie vor als sehr sicher. Doch seit Bekanntwerden der Biclique-Kryptoanalyse weiß man, dass die vollständige Schlüsselsuche nicht die beste Methode ist, um ihn zu knacken. Zusammen mit den Überlegungen zur quadratischen Kryptoanalyse und der Darstellbarkeit als algebraische Formel hätte dies vermutlich ausgereicht, um den Sieg von Rijndael zu verhindern.

Interessant ist nun folgende Frage: Haben die Experten die Sicherheit von Rijndael überschätzt, als sie das Verfahren zum AES kürten, oder sind die vielen Kryptoanalyse-Ergebnisse darauf zurückzuführen, dass sich so viele mit dem Verfahren beschäftigten. Zweifellos spielt der letztgenannte Aspekt eine wichtige Rolle, denn seit Ende des Wettbewerbs haben Heerscharen von Kryptografen mit großem Aufwand nach Schwachstellen im AES gesucht. Niemand weiß, wie Serpent und Twofish heute dastehen würden, wenn sie so intensiv untersucht worden wären. Andererseits ist zu beachten: Serpent und Twofish haben ein konservativeres Design als der AES. Es ist daher durchaus möglich, dass diese beiden Verfahren den Kryptoanalytikern weniger Angriffsfläche geboten hätten. Interessant ist auf jeden Fall eine Lehre, die man aus der Geschichte des AES ziehen kann: Selbst wenn sich zahlreiche, weltweit führende Experten über 15 Jahre lang mit einem Verfahren beschäftigt haben, können noch interessante Dinge entdeckt werden.