Praktikum Algorithmische Anwendungen
WS 2006/07

Thema
Der Advanced Encryption Standard (AES) als symmetrisches Kryptosystem

Gruppe
B_gelb_Ala0607

Teilnehmer
Florian Zaefferer, 11042017, florian.zaefferer@gmx.de
Christian Zaefferer, 11041048, c.zaefferer@gmx.de
Radhouan Jouini, 11033001, jouini@web.de

Inhaltsverzeichnis

1 Zielsetzung

Hier soll kurz erläutert werden, unter welchen Aspekten AES betrachtet werden soll, und wo die Ziele unserer Ausarbeitung liegen.

Sicherheit und Kryptographie im Allgemeinen
Hier soll kurz die Bedeutung der Kryptographie innerhalb der IT-Sicherheit, sowie einige Grundbegriffe aus der IT-Sicherheit erläutert werden. Dies soll einen Blick auf die Anwendungsgebiete des AES ermöglichen und dessen Bedeutung hervorheben.

Die Geschichte des AES
Hier wird auf die Entstehung des Standards eingegangen. Dabei wird auch der Vorgänger von AES, DES (Data Encryption Standard) betrachtet, die grobe Funktionsweise von DES und 3DES erläutert und die Schwächen von DES aufgezeigt, aufgrund dessen der neue Standard entwickelt wurde. Dann wird noch die Entstehung des Standards im Auswahlprozess des NIST erläutert und warum der Rijndael-Algorithmus ausgewählt wurde.

Allgemeine Eigenschaften von AES
Hier sollen die allgemeinen Eigenschaften des AES beschrieben werden. Dabei soll explizit noch nicht auf genauere algorithmische Eigenschaften oder Funktionsweisen eingegangen werden. Es sollen nur die Eigenschaften des Algorithmus betrachtet werden, die für Anwendungen relevant sind.

Algorithmische Funktionsweise
Hier wird die genaue Funktionsweise des Algorithmus analysiert und beschrieben. (Genauere Unterteilung diese Kapitels ist zu diesem Zeitpunkt noch nicht möglich)

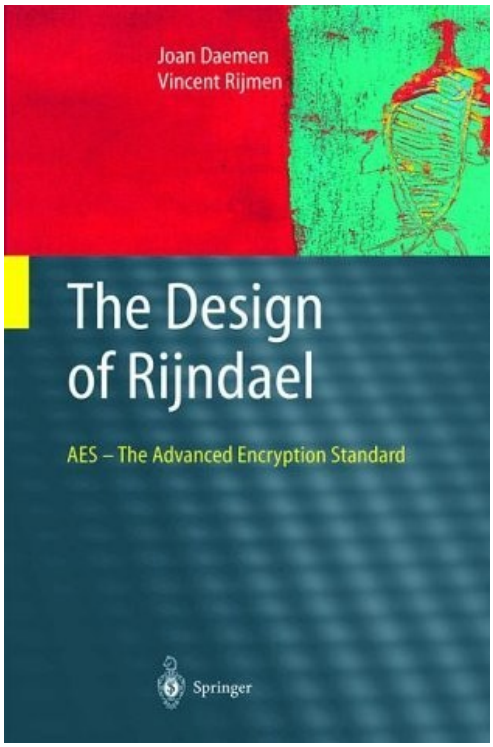
Laufzeitbetrachtungen
Hier sollen Laufzeitbetrachtungen zu dem Algorithmus angestellt werden.

2 Sicherheit und Kryptographie im Allgemeinen

Sicherheit lässt sich nicht pauschal definieren. Der Begriff Sicherheit bedeutet im allgemeinen, dass keine Gefahr besteht, dass nichts Unerwünschtes geschehen kann. Was jedoch erwünscht oder nicht erwünscht ist, hängt von der Situation ab. Somit orientiert sich Sicherheit an den Soll-Vorgaben für unerwünschte oder erwünschte Ereignisse, die für bestimmte Szenarien oder Umgebungen festgelegt wurden. In der IT-Sicherheit können solche Szenarien z.B. sein: ein Homebanking-System - „Kommt das angegebene Zielkonto für eine Transaktion unverändert bei der Bank an?“ oder „Kann niemand außer mir und der Bank in die Daten meiner Transaktion einsehen?“, E-Mail Kommunikation - „Ist mein Kommunikationspartner echt?“, Mehrbenutzer-System - „Habe nur ich Zugriff auf meine Daten?“
Dabei ist zu Beachten das nicht die Daten das Schützenswerte sind, sondern die in den Daten enthaltenen Informationen. Häufig werden z.B. wesentlich mehr Daten verschlüsselt als eigentlich nötig wäre. Aus den in den Daten enthaltenen Informationen ergeben sich

auch die Anforderungen an den Begriff „Sicherheit“ in diesem Szenario. Daher Definiert man verschiedene „Schutzziele“. Schutzziele definieren unerwünschte Ereignisse. Diese Schutzziele variieren von Szenario zu Szenario. Beispiele für Schutzziele sind: Vertraulichkeit (Nur Berechtigte dürfen Zugriff auf Informationen erhalten), Integrität (Daten können nicht unberechtigt verändert oder manipuliert werden), Verfügbarkeit (Ein System oder Daten kann für einen vorgesehenen Zweck tatsächlich genutzt werden), Authentizität (Sicherstellung der Echtheit eines Kommunikationspartners)
Für jedes Szenario lassen sich eines oder mehrere dieser Schutzziele bestimmen. Sie definieren den Begriff „Sicherheit“ in diesem Szenario.
Kryptographie ist eine Maßnahme um die Schutzziele Vertraulichkeit oder Authentizität (->Signatur) zu erfüllen. Die Daten werden verschlüsselt, so dass ohne den Schlüssel keine Einsicht in die Informationen in einer Nachricht genommen werden kann. Aus dem Klartext wird über ein Verfahren und einen geheimen, frei wählbaren Schlüssel ein sog. Chiffrat. Umgekehrt muss es möglich sein, mit dem bei der Verschlüsselung verwendetem Schlüssel und dem Chiffrat mittels des Verfahrens wieder den Klartext zu erhalten. Dies nennt man symmetrische Kryptographie.
Die Verfahren sind heutzutage nicht mehr geheim - im Gegenteil. Sie werden von einer breiten Öffentlichkeit auf ihre Sicherheit geprüft. Es ist nicht selten, dass gerade vorgestellte Verfahren noch während der Präsentation von Zuhörern (in der Regel Experten) widerlegt (d.h. Sicherheitslücken aufgewiesen) werden.
Das Verfahren muss also so gestaltet sein, das es ein Chiffrat erstellt, das ohne den Schlüssel nicht auf seinen Klartext zurückzuführen ist. Außerdem darf aus dem Chiffrat nicht auf den Schlüssel zu schließen sein. Diese Forderungen stellen gewisse Anforderungen an das Verfahren. Es gibt diverse Möglichkeiten der Kryptoanalyse um Verschlüsselungsverfahren zu knacken. Einige davon sind: Brute Force (Durchprobieren aller möglichen Schlüssel), Differentielle Kryptoanalyse (Ermittlung von Unterschieden der Ausgabebits, bei kleinen Variationen der Eingabebits), Lineare Kryptoanalyse (basiert auf linearen Zusammenhängen zwischen Klartext, Chiffretext und Schlüssel)
Gegen diese und andere Analysemethoden muss ein Verschlüsselungsverfahren stand halten, um heutzutage als sicher eingestuft zu werden.

3 Buchvorstellung: Joan Daemen, Vincent Rijmen: The Design of Rijndael



Kapitel:

1. Der Prozess zum AES
2. Vorbereitende Maßnahmen
3. Spezifikationen
4. Aspekte der Implementation
5. Design Philosophie
6. Der DES Standard
7. Lineare Cryptoanalyse
8. Differentiale Cryptoanalyse
9. Wide Trail Strategie
10. Cryptoanalyse
11. Verwandte Blockcodes

Einteilung:
Kapitel 3 ist das wichtigstes Kapitel, wenn man eine eindeutige Beschreibung des Rijndaels Algorthmus sucht. Dafür nützlich ist es Kapitel 2 zu lesen. Fortgeschrittene Aspekte der Implementierung werden in Kapitel 4 beschrieben.
Kapitel 1 beschreibt kurz den Prozess zur Auswahl eines Algorithmus für den AES.

Ein großer Teil des Buches beschäftigt sich mit der Frage, warum der Rijndael so

entworfen wurde wie er ist.
Deshalb wird in Kapitel 5 zunächst die Herangehensweise an einen Blockcode beschrieben und welche Kriterien überhaupt eine Rolle spielen. Da der AES besser sein soll, wird in Kapitel 6 der DES Standard, der Vorgänger zum AES beschrieben und welche Attacken auf ihn ausgeführt werden können. In Kapitel 7 wird dann beschrieben, wie die lineare Kryptoanalyse funktioniert, und im Kapitel 8 dann die Differentiale Kryptoanalyse. In Kapitel 9 wird dann die sich aus den vorherigen Kapiteln ergebende Wide Trail Strategie beschrieben.
Kapitel 10 beschreibt veröffentlichte Angriffe auf den Rijndael. In Kapitel 11 werden zuletzt mit dem Rijndael Algorithmus verwandte Algorithmen vorgestellt.

4 Die Geschichte des AES

Advanced Encryption Standard (AES) beschreibt einen symmetrischen Algorithmus der von Joan Daemen und Vincent Rijmen, zum Verschlüsseln von Daten entwickelt wurde. Er wird auch Rijndael-Algorithmus genannt, da sich diese Bezeichnung aus den Namen der Autoren zusammensetzt (Rij+Dae).

Der Entstehungsgrund des AES lässt sich allerdings auf seinen Vorläufer zurückführen, nämlich dem am häufigsten genutzten symmetrische Algorithmus zur Verschlüsselung von Daten, dem → DES. Da die NSA beim Entwurf des DES beteiligt war, vermutete man Hintertüren der NSA, die aber nie gefunden werden konnten. Entscheidend jedoch waren Zweifel an seiner Sicherheit; da nur ein 56 Bit langer Schlüssel benutzt wurde, und der Algorithmus somit sehr anfällig gegen Brute Force-Angriffe ist. Auch sein Nachfolger 3DES (3-Fache Anwendung des Algorithmus) konnte die Sicherheit nicht mehr garantieren, obwohl die Schlüssellänge somit effektiv auf 112 Bit erhöht wurde.
Dies war Anlass genug um einen neuen Verschlüsselungsstandard zu entwickeln, welche möglichst nicht die Fehler seiner Vorgänger innehat.

Somit wurde dann vom US-amerikanischen National Institue of Standards and Technology, kurz, NIST, ein offener Wettbewerb aufgerufen, dessen Sieger als Advanced Encryption Standard (AES) festgelegt werden soll.

Folgende Kriterien für die Algorithmen sollten dabei berücksichtigt werden:

- AES muss ein symmetrischer Algorithmus sein → und zwar ein Blockalgorithmus
- Die Blockgröße für AES muss 128 Bit betragen, und es müssen Schlüssellängen von 128, 192 oder 256 Bit möglich sein.
- Leichtigkeit für die Implementierung in Hard- und Software.
- Die Performance von AES sollte in der Hard- und Software überdurchschnittlich sein.
- AES sollte eine relative Sicherheit bezüglich der anderen AES-Kandidaten vorweisen.
- Kompatibilität mit verschiedenen Plattformen und eine ausreichende Geschwindigkeit sollte gegeben sein.
- Die Speicheranforderungen sollten möglichst gering gehalten werden, vor allem für den Einsatz von mobilen Geräten.

Dieser Maßstab war ein Grund dafür, dass in der NIST 1998 15 Algorithmen eingegangen sind. Diese wurden nach den oben genannten Kriterien gemessen und öffentlich diskutiert. Im Jahre 1999 hatten es letztlich 5 dieser Algorithmen zur engeren Auswahl geschafft.

- RC6
- MARS
- Rijndael
- Twofish
- Serpent

Diese 5 sind somit in die nächste Runde eingezogen, da Sie allen Anforderungen standgehalten haben.

Aufgrund dieser engeren Wahl beschloss die NIST weitere Kriterien hinzuzufügen, die den letztendlich besten dieser 5 Algorithmen heraus filtern.
D.h., man beschloss theoretische Schwachstellen zu überprüfen, die einen Algorithmus unsicher machen könnten, wenn man an die Rechner-Entwicklung der Zukunft denkt (z.B. mehr Rechenleistung für Brute Force).
Der Rijndael-Algorithmus war der einzige Algorithmus, der in der Hard- und Software-Implementierung sehr schnell abschnitt.
Hinzukamen noch diverse Schwachstellen in unterschiedlichen Bereichen der anderen Kandidaten, was im Jahre 2000 den Rijndael-Algorithmus zum Sieger des Wettbewerb machte.

Der belgische Rijndael-Algorithmus wurde aufgrund seiner Einfachheit in der Implementierung und guten Performance zum neuen Standard gemacht und trägt somit auch den Namen Advanced Encryption Standard (AES).

5 Allgemeine Eigenschaften von AES

Der Ablauf beim Verschlüsseln eines Textes ist folgender:
- Zunächst wird ein geheimer Schlüssel gewählt
- Die zu verschlüsselnden Daten werden in gleich große Blöcke geteilt. Der letzte Block wird mit 1000... () gefüllt. Ist der letzte Block bereits voll, wird ein zusätzlicher Block mit 1000... angehängt.
- Jeder Block wird mit einem festen Algorithmus transformiert und mit dem Schlüssel kombiniert, so dass aus dem neuen transformierten Block ohne die Kenntnis des Schlüssels nicht mehr auf den Klartext geschlossen werden kann.
- Die so entstandenen verschlüsselten Blöcke bilden den Geheimtext. Er ist genauso lang wie der Klartext, da sich die Blockgröße durch das Verschlüsseln nicht ändert.
- Mit dem inversen Algorithmus und dem Schlüssel kann aus dem Geheimtext wieder block weise der Klartext errechnet werden.

Die Schlüsselgröße kann gewählt werden aus den Werten 128, 192 oder 256 Bit. Beim Rijndael Algorithmus kann die Blockgröße 128, 160, 192, 224 und 256 Bit sein, der AES Standard ist dagegen eingeschränkt auf 128 Bit.

Der Advanced Encryption Standard (AES) als symmetrisches Kryptosystem

Der Rijndael Algorithmus musste bestimmt Eigenschaften erfüllen:
Sicherheit:
- Robustheit der Verschlüsselung gegenüber der Kryptoanalyse
- Stichhaltigkeit der mathematischen Basis
- Zufälligkeit der ausgegebenen Geheimtexte
- Relative Sicherheit bezüglich der anderen AES-Kandidaten

Kosten:
- Keine Patentansprüche der Entwickler
- Ausreichende Geschwindigkeit auf verschiedenen Rechnerarchitekturen
- Geringe Speicheranforderungen für die Verwendung in mobilen Geräten

Eigenschaften für die Implementierung:
- Eignung für verschiedenste Hard- und Softwaresysteme
- Einfachheit des Verfahrens
- Flexible Schlüssellänge – 128, 192 und 256 Bit
- Blockgröße mindestens 128 Bit

6 Mathematische Grundlagen

Die Operationen des Algorithmus operieren auf Bytes oder Worten. Ein Byte besteht aus 8 Bit. 4 Bytes bilden ein Wort. Somit sind die Operationen für 8 und 32 Bit ausgelegt. Die gesamte Menge der 256 Zustände eines Bytes wird in der Primärliteratur als $GF(2^8)$ bezeichnet. Ein Byte b wird besteht aus den Bits $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$ und wird als Polynom 7. Grades mit den Koeffizienten {0,1} dargestellt:

$$b= b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0 x^0$$

Beispiel:
Ein Byte „4B“$_{16}$ (= „01001011“$_2$) wird als Polynom durch „$x^6 + x^3 + x + 1$“ dargestellt.

Auch ein Wort kann als Polynom dargestellt werden. Es resultiert ein Polynom 3. Grades. Der 4-Byte-Vektor v besteht aus den Bytes $b_0\ b_1\ b_2\ b_3$ mit den Koeffizienten {00,01,02,...,FF}:

$$v = b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0 x^0$$

Beispiel:

Ein Byte-Vektor $\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} D5 \\ 4\mathrm{B} \\ AF \\ 5E \end{bmatrix}_{16}$ wird als Polynom durch „{5E}$\cdot x^3$ + {AF}$\cdot x^2$ + {4B}$\cdot$ x + {D5}“ dargestellt.

6.1 Addition

Die Addition beim Rijndael-Algorithmus ist eine einfache bitweise XOR-Verknüpfung [1] (⊕).

Beispiel: Addition zweier Bytes

Hexadezimaldarstellung: 4B ⊕ 39 = 72

Polynomdarstellung: $(x^6 + x^3 + x + 1) \oplus (x^5 + x^4 + x^3 + 1) = x^6 + x^5 + x^4 + x$

Binärdarstellung:

$$\begin{array}{ll} & 01001011 \\ \oplus & \underline{00111001} \\ & 01110010 \end{array}$$

Beispiel: Addition zweier 4-Byte Vektoren (Wörter)

Hexadezimaldarstellung:

$$\begin{bmatrix} D5 \\ 4\mathrm{B} \\ AF \\ 5E \end{bmatrix}_{16} \oplus \begin{bmatrix} BC \\ A6 \\ D1 \\ 9\mathrm{A} \end{bmatrix}_{16} = \begin{bmatrix} 69 \\ ED \\ 7E \\ C4 \end{bmatrix}_{16}$$

Polynomdarstellung:

$$\begin{array}{ll} & \{5E\}\cdot x^3 + \{AF\}\cdot x^2 + \{4B\}\cdot x + \{D5\} \\ \oplus & \underline{\{9A\}\cdot x^3 + \{D1\}\cdot x^2 + \{A6\}\cdot x + \{BC\}} \\ & \{C4\}\cdot x^3 + \{7E\}\cdot x^2 + \{ED\}\cdot x + \{69\} \end{array}$$

Binärdarstellung:

$$\begin{array}{ll} & 01011110101011110100101111010101 \\ \oplus & \underline{10011010110100011010011010111100} \\ & 11000100011111101110110101101001 \end{array}$$

6.2 Multiplikation

Die Multiplikation entspricht in der Polynomdarstellung einer Multiplikation der Polynome, modulo ein festes nicht zu vereinfachendes binäres Polynom. Diese Eigenschaft des Polynoms ist wichtig, da es sich somit nur durch sich selbst und 1 teilen lässt. Das garantiert das Entstehen eines Rests. Die Multiplikation basiert dabei auf der oben beschriebenen Addition. Für die Multiplikation zweier Bytes ist das Polynom ein Polynom achten Grades und wird beim Rijndael-Algorithmus m(x) genannt. Es lautet wie folgt:

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

In hexadezimaler Schreibweise „11B“.

---

1 XOR-Rechnung:
1 ⊕ 1 = 0
1 ⊕ 0 = 1
0 ⊕ 1 = 1
0 ⊕ 0 = 0

Beispiel:
Hexadezimal:
$5E_{16} \otimes 9B_{16} = 10_{16}$

1. Schritt: Multiplikation

$$(x^6 + x^4 + x^3) \otimes (x^7 + x^4 + x^3 + x + 1) = x^{13} + x^{10} + x^9 + x^7 + x^6 + x^{11} + x^8 + x^7 + x^5 + x^4 + x^{10} + x^7 + x^6 + x^4 + x^3$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^7 + x^5 + x^3$$

2. Schritt: modulo m(x)

$$x^{13} + x^{11} + x^9 + x^8 + x^7 + x^5 + x^3 \text{ modulo } x^8 + x^4 + x^3 + x + 1 = x^4$$

Division zweier Polynome:

$$(x^{13} + x^{11} + x^9 + x^8 + x^7 + x^5 + x^3) : (x^8 + x^4 + x^3 + x + 1) = x^5 + x^3$$
$$\underline{(x^{13} + x^9 + x^8 + x^6 + x^5)} \quad \text{Rest: } x^4$$
$$(x^{11} + x^7 + x^6 + x^3)$$
$$\underline{(x^{11} + x^7 + x^6 + x^4 + x^3)}$$
$$(x^4)$$

Der Aufwand wird betrieben um als Ergebnis auf jeden Fall ein Polynom kleiner Grad 8 zu erhalten. Denn ein solches wird benötigt um die 1 Byte Größe der Zellen in den Blöcken nicht zu überschreiten.
Die Multiplikation modulo eines Polynoms achten Grades wird auf 8-Bit-Systemen zum Einsatz gebracht. Die Multiplikation ist allerdings auch für 4-Byte-Vektoren (Worte) und somit dem Einsatz auf 32-Bit-Systemen definiert. Hier muss das feste, nicht zu vereinfachende Polynom 4. Grades sein, da das Ergebnis der Multiplikation hier, mit der Modulo-Operation, wieder auf ein Polynom 3. Grades reduziert werden muss. Das Polynom lautet in diesem Fall:

$$m(x) = x^4 + 1$$

7 Arbeitsweise und Ablaufstruktur des AES bei Verschlüsselung

Der Verschlüsselungsvorgang besteht aus mehreren Schritten.
Der erste Schritt ist die Schlüsselexpansion. Während der eigentlichen Verschlüsselung wird ein Schlüssel benötigt, der wesentlich größer ist als der vom Benutzer eingegebene Schlüssel. Daher wird der eingegebene Schlüssel zunächst aufgebläht. Dies geschieht während der gesamten Verschlüsselung eines Textes nur einmal.
Da der AES ein Blockchiffre ist, muss anschließend der zu verschlüsselnde Text in einzelne Blöcke unterteilt werden. Beim Advanced Encryption Standard ist die Blockgröße auf 128 Bit begrenzt, der Rijndael Algorithmus selbst unterstützt auch die Blockgrößen 160, 192, 224 und 256 Bit, diese wurden aber nicht in den Standard übernommen. Jeder Block ist eingeteilt in eine Tabelle aus 4 Spalten und 4 Zeilen, und jede Zelle beinhaltet 1 Byte.

Ist dies geschehen, kann dann jeder Block unabhängig voneinander mit einem festen Algorithmus verschlüsselt werden. Dieser Algorithmus besteht aus abwechselnder Schlüsseladdition und einer Linearen Permutation. Diese Lineare Permutation besteht aus den 3 Schritte ByteSubstitution, ShiftRows und MixColumns. Wie oft diese ausgeführt werden, hängt von der Block- und Schlüsselgröße ab. Da der AES nur eine Blockgröße unterstützt, hängt es hier also nur von der Schlüsselgröße ab:

Schlüsselgröße (in Bit)	128	192	256
Anzahl Runden	10	12	14

Tabelle 1: Anzahl der Runden in Abhängigkeit Schlüssellänge

Die Anzahl der Runden beinhaltet die Schlussrunde. Der Ablauf des Algorithmus ist rechts dargestellt.

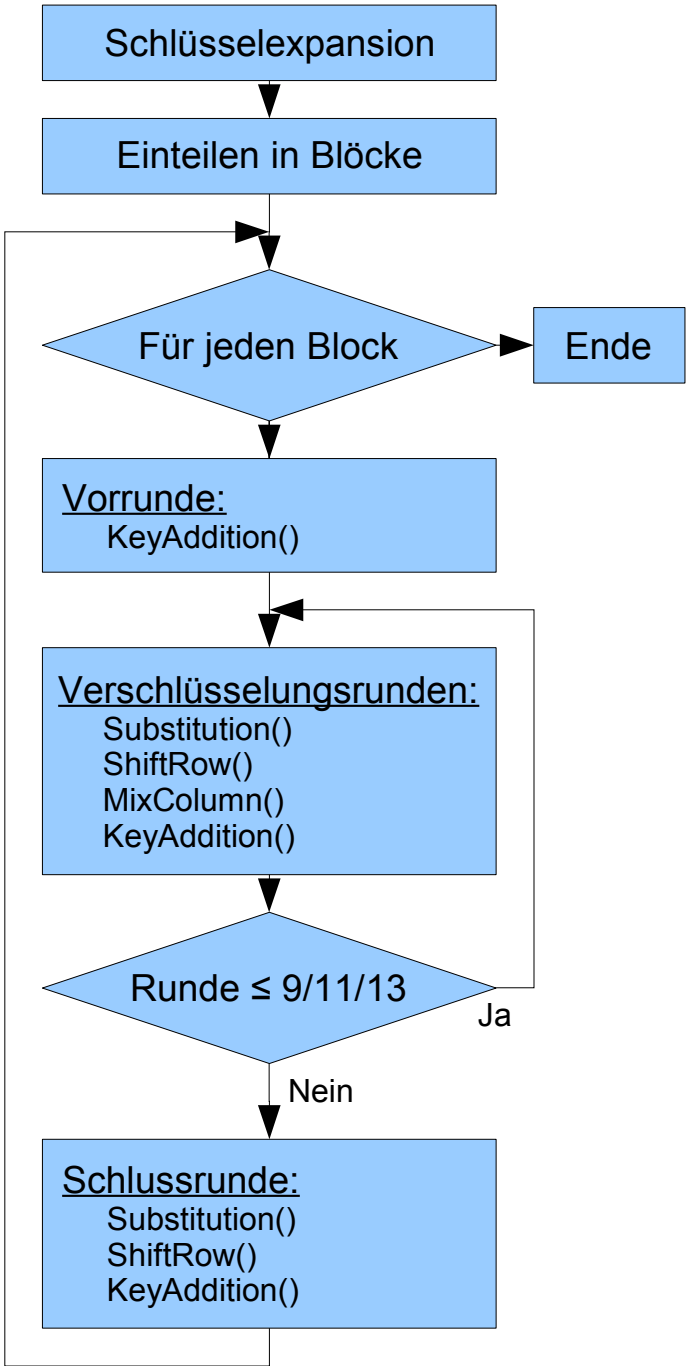


Abbildung 1: Ablauf der Verschlüsselung

7.1 Definition und Spezifikation der einzelnen Funktionen

7.1.1 Schüsselexpansion

Die Funktion KeyAddition addiert auf den zu verschlüsselnden Block einen Schlüssel (siehe nächstes Kapitel). Da man aber dort nicht jedes Mal den gleichen Schlüssel verwenden möchte, ist es nötig, weitere Schlüssel (sogenannte Rundenschlüssel) anzulegen. Dies geschieht durch die Schlüsselexpansion zu Beginn der Verschlüsselung eines Textes einmal.
Da die Funktion KeyAddition während der Vorrunde, jeder Verschlüsselungsrunde und der Nachrunde ausgeführt wird, werden insgesamt 12, 14 oder 16 Rundenschlüssel (zukünftig AR genannt) der Länge 128 Bit benötigt. Für alle Rundenschlüssel wird eine gemeinsame Tabelle angelegt der Größe 128 Bit * AR. Diese Tabelle besteht aus 4 Zeilen, AR * 4 Spalten, und jede Zelle enthält 1 Byte. Die Tabelle wird spaltenweise von links nach rechts gefüllt. Zunächst wird der vom Benutzer eingegebene Schlüssel eingefügt. Dahinter werden alle Werte dann rekursiv anhand der vorheriger Werte berechnet. Durch diese Struktur ist es auch zum einen möglich, Schlüssel, die größer oder kleiner als die angegebene Schlüssellänge sind, zu verwenden. Zum anderen sind dadurch alle Rundenschlüssel abhängig vom eingegebenen Schlüssel. Außerdem ist es dadurch möglich, dass die gewählte Schlüsselgröße auch anders ist als die gewählte Blockgröße, da der erweiterte Schlüssel aufgeteilt wird in Rundenschlüssel mit der gleichen Größe eines zu verschlüsselnden Blockes.

Die Zellen der ersten drei Spalten eines Rundenschlüssels werden durch die Addition des Wertes links der Zelle und des Wertes 4 Spalten nach links berechnet. Die vierte Spalte wird etwas anders berechnet, damit lineare Abhängigkeiten aufgelöst werden.
Der erste Wert der Spalte berechnet sich aus drei Werten:

- aus dem Werte der Zelle 4 Spalten nach links
- aus dem Wert der Zelle eins nach links und eins nach unten, ersetzt durch die S-Box
- einer speziellen Rundenkonstante

Durch diese Struktur ist der der erste Wert jeder vierten Spalte nicht linear abhängig vom eingegebenen Schlüssel.
Die restlichen 3 Werte werden durch 2 Werte berechnet:

- aus dem Wert der Zelle 4 Spalten nach links und
- aus dem Wert der Zelle eins nach links und eins nach unten (bzw. dem obersten Wert in der letzten Zeile)

Dies dient dazu, den nicht linear abhängigen Wert in der ersten Zeile in den folgenden Runden in die nächsten Zeilen zu übernehmen.
Durch diese Struktur werden dann auch ungünstige Schlüssel (wie z.B. ein Schlüssel nur aus Nullen) nach einigen Runden aufgelöst und werden dann übergehen in heterogene Schlüssel.

7.1.2 KeyAddition

In einer sog. Vorrunde wird das erste Mal die KeyAddition ausgeführt. Hier wird einfach eine XOR-Verknüpung zwischen der block-enthaltenden Tabelle und der Tabelle mit dem

aktuellen Rundenschlüssel durchgeführt. Dabei wird immer die jeweils äquivalente Zelle (a[0][0] XOR b[0][0],a[0][1] XOR b[0][1], usw.) beider Tabellen Verknüpft.

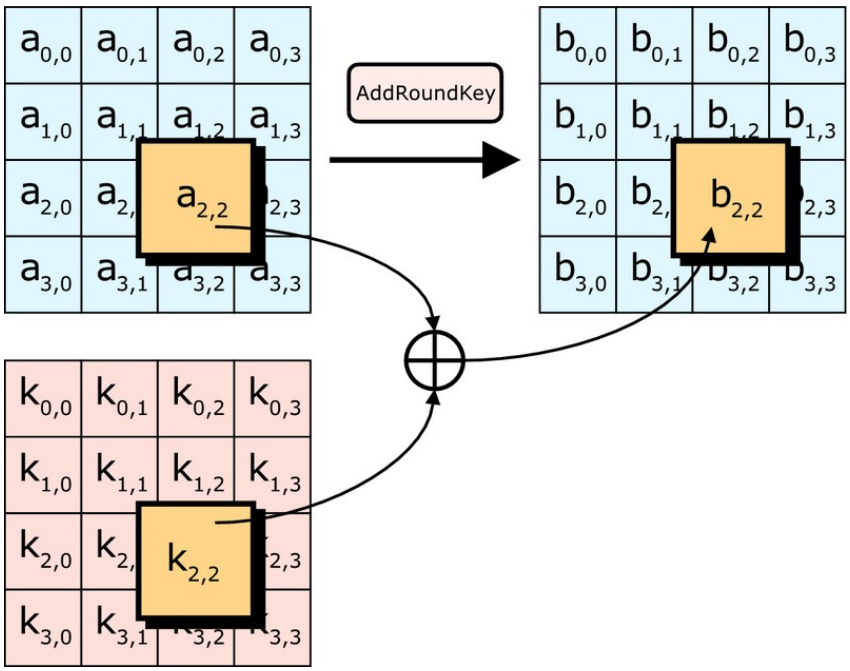


Abbildung 2: Schema der XOR-Verknüpfung des aktuellen Blocks und des aktuellen Rundeschlüssels
(Quelle: http://de.wikipedia.org/wiki/Advanced_Encryption_Standard)

Die KeyAddition wird auch in jeder weiteren Runde sowie der Schlussrunde ausgeführt. Es ist die einzige Funktion, die das Ergebnis abhängig vom eingegebenen Schlüssel macht.

7.1.3 Substitution

In diesem Schritt wird die aktuelle Tabelle über eine sog. S-Box (Substitutionsbox) monoalphabetisch verschlüsselt. Die Substitutionsbox besteht aus einer Tabelle mit festgelegten Werten. Die Eingabe-Bytes werden dabei einfach durch bestimme Bytes aus der S-Box ersetzt. Dabei handelt es sich somit um eine statische S-Box. Die Tauschungen stehen also immer fest und sind fest im Algorithmus implementiert.

7.1.4 ShiftRow

ShiftRow ist die zweite Operation, die in jeder Runde ausgeführt wird. Dabei werden in der Tabelle, die den aktuellen Block enthält, auf die Zeilen bestimmte Verschiebungen ausgeführt.

v	b=128	b=192	b=256
Zeile 0	0	0	0
Zeile 1	1	1	1
Zeile 2	2	2	3
Zeile 3	3	3	4

Tabelle 2: Anzahl der Links-Verschiebungen v, in Abhängigkeit von der Blockgröße b und der jeweiligen Zeile.

Abhängig von der jeweiligen Zeile werden die Zellen um 0-3 Spalten nach links verschoben. Links „herausfallende“ Zellen werden in der selben Zeile rechts wieder angesetzt.

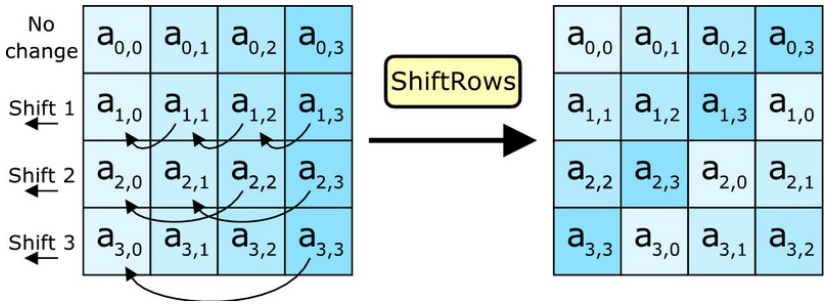


Abbildung 3: Schema der ShiftRow-Operation im aktuellen Block (Quelle: http://de.wikipedia.org/wiki/Advanced_Encryption_Standard)

Durch diese Zeilenverschiebung wird in Kombination mit der MixColumns-Operation eine horizontale Abhängigkeit der Zellen geschaffen. Verstärkt wird dieser Effekt dadurch, dass die Funktion bei 128Bit Blockgröße und 128Bit Schlüssellänge z.B. 10 mal ausgeführt wird. Die Bytes werden also durch die Kombination der Funktionen ShiftRow und MixColumn in ausreichender Weise vermischt (durch ShiftRow horizontal , durch MixColumn vertikal).

7.1.5 MixColumn

In der MixColumn-Funktion werden die Spalten des aktuellen Blocks als Polynome über $GF(2^8)$ betrachtet und modulo x4 + 1 mit einem festen Polynom c (x) multipliziert.
Das Polynom lautet:

$$c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

Dieses Polynom ist nicht durch das Polynom $x^4 + 1$ teilbar und außerdem invertierbar.

Diese Eigenschaften sind wichtig, damit erstens das Ergebnis der Modulo-Operation nicht 0 ist und zweitens, damit die Mix-Column-Operation umkehrbar ist. Die Rechnung modulo $x^4 + 1$ ist notwendig, da die Multiplikation zweier Polynome 3. Grades ein Polynom ≤ 6. Grades ergeben kann. Das Ergebnis muss allerdings eine neue Spalte im Ergebnis-Block sein, also auch wieder ein Polynom 3. Grades. Daher reduziert man das Ergebnis durch Modulo $x^4 + 1$.

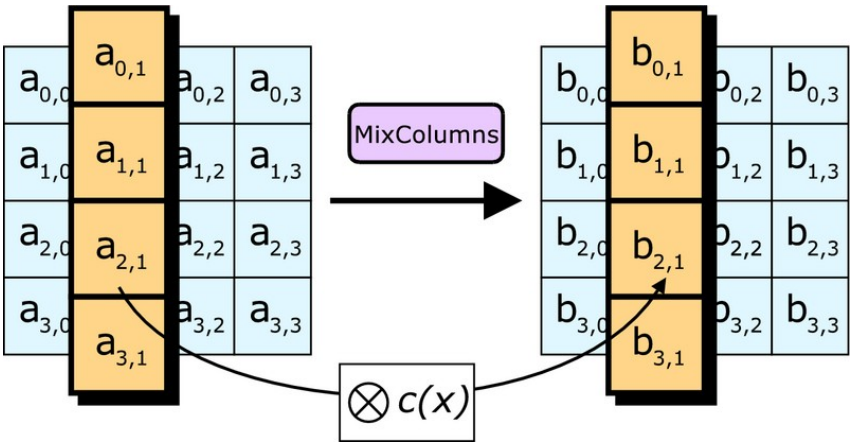


Abbildung 4: Schema der MixColumn-Operation. c(x) bezeichnet das feststehendes Polynom $\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$
(Quelle: http://de.wikipedia.org/wiki/Advanced_Encryption_Standard)

Die wortweise Multiplikation hat man eingeführt um die Operation für 32-Bit-Systeme zu optimieren. Natürlich lässt sich die Operation auch byteweise durchführen. Dann ist das feste Polynom für die Modulo-Operation allerdings $m(x) = x^8 + x^4 + x^3 + x + 1$, wie in den mathematischen Grundlagen bereits beschrieben. Dies entspräche dann der Berechnung auf einem 8-Bit-System.

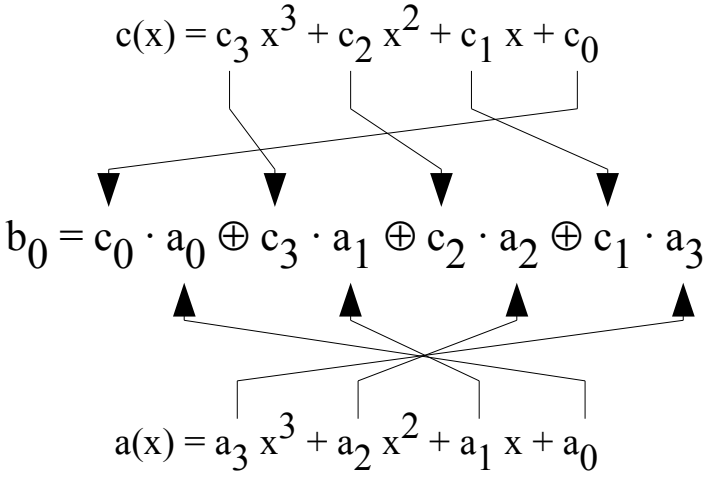
Die Multiplikation mit c(x) wird in der Literatur als Matrix-Multiplikation dargestellt:

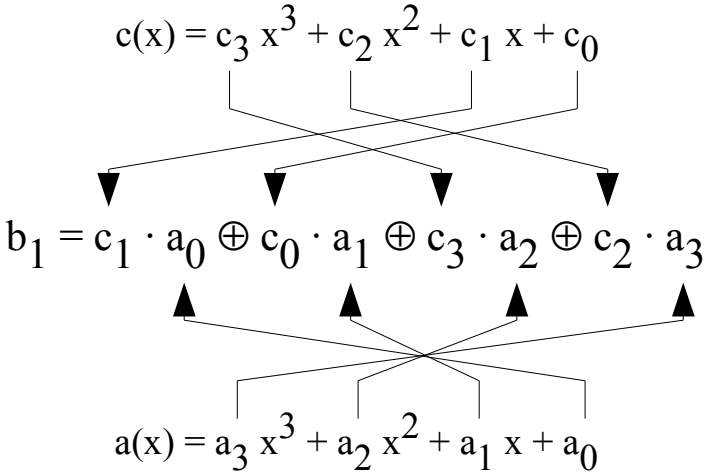
Sei b der Ergebnis-Bytevektor, a der Bytevektor des „Ursprungs-Blocks“ und c(x) das oben beschriebene Polynom $\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.
Dann gilt:

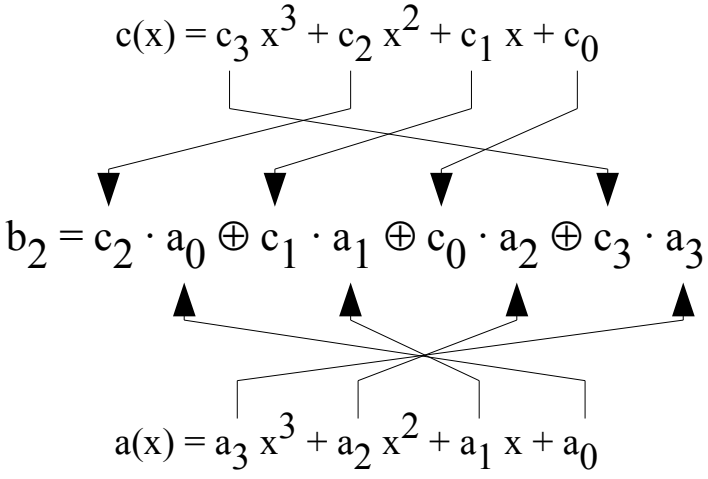
Polynomdarstellung:

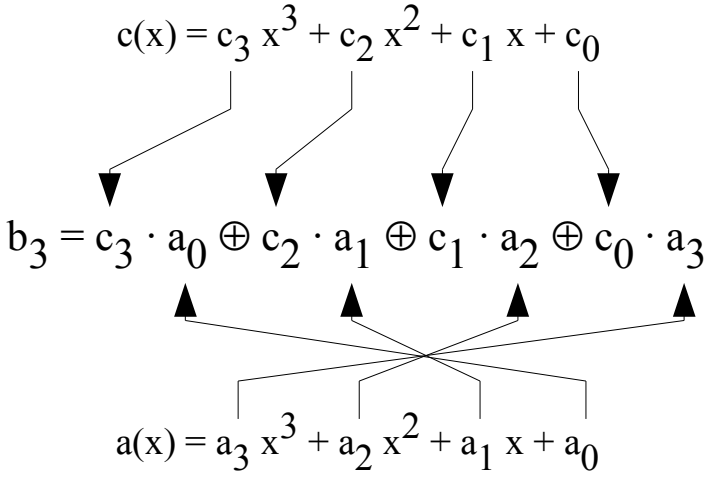
$$a(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0$$
$$c(x) = c_3 x^3 + c_2 x^2 + c_1 x + c_0$$
$$b(x) = b_3 x^3 + b_2 x^2 + b_1 x + b_0$$
$$M(x) = x^4 + 1$$

$$b(x) = a(x) \oplus c(x) \bmod M(x)$$


c(x) = c3 x3 + c2 x2 + c1 x + c0
b0 = c0 · a0 ⊕ c3 · a1 ⊕ c2 · a2 ⊕ c1 · a3
a(x) = a3 x3 + a2 x2 + a1 x + a0


c(x) = c3 x3 + c2 x2 + c1 x + c0
b1 = c1 · a0 ⊕ c0 · a1 ⊕ c3 · a2 ⊕ c2 · a3
a(x) = a3 x3 + a2 x2 + a1 x + a0


c(x) = c3 x3 + c2 x2 + c1 x + c0
b2 = c2 · a0 ⊕ c1 · a1 ⊕ c0 · a2 ⊕ c3 · a3
a(x) = a3 x3 + a2 x2 + a1 x + a0



Diese Zusammenstellung garantiert eine optimale Verteilung des Ergebnisses.
Dadurch, dass die Koeffizienten in jedem Byte anders kombiniert werden, ist es möglich eine Matrix aufzustellen:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

da $c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$, ergibt sich:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

ausgeschrieben:

$b_0 = \{02\} \cdot a_0 \oplus \{03\} \cdot a_1 \oplus \{01\} \cdot a_2 \oplus \{01\} \cdot a_3$
$b_1 = \{01\} \cdot a_0 \oplus \{02\} \cdot a_1 \oplus \{03\} \cdot a_2 \oplus \{01\} \cdot a_3$
$b_2 = \{01\} \cdot a_0 \oplus \{01\} \cdot a_1 \oplus \{02\} \cdot a_2 \oplus \{03\} \cdot a_3$
$b_3 = \{03\} \cdot a_0 \oplus \{01\} \cdot a_1 \oplus \{01\} \cdot a_2 \oplus \{02\} \cdot a_3$

Beispiel:
Der Einfachheit halber wird anstatt auf Wort-Basis, auf Byte-Basis gerechnet.
($M(x) = x^8 + x^4 + x^3 + x + 1$)

$$MixColumns\begin{pmatrix} D5 & C8 & 56 & D5 \\ 4B & A6 & E8 & DD \\ AF & 59 & F0 & 82 \\ 5E & 0B & EE & 4F \end{pmatrix} = \begin{pmatrix} 9D & 28 & 91 & 00 \\ F7 & 7F & 78 & A6 \\ 39 & C1 & 6C & C6 \\ 3C & AA & 25 & A5 \end{pmatrix}$$

Erste Spalte:

$$a(x) = \begin{bmatrix} D5 \\ 4\mathrm{B} \\ AF \\ 5E \end{bmatrix} = \begin{bmatrix} 11010101 \\ 01001011 \\ 10101111 \\ 01011110 \end{bmatrix} \equiv \begin{bmatrix} x^7+x^6+x^4+x^2+1 \\ x^6+x^3+x+1 \\ x^7+x^5+x^3+x^2+x+1 \\ x^6+x^4+x^3+x^2+x \end{bmatrix}$$

$$c(x) = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 10 & 11 & 01 & 01 \\ 01 & 10 & 11 & 01 \\ 01 & 01 & 10 & 11 \\ 11 & 01 & 01 & 10 \end{bmatrix} \equiv \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix}$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} \cdot \begin{bmatrix} x^7+x^6+x^4+x^2+1 \\ x^6+x^3+x+1 \\ x^7+x^5+x^3+x^2+x+1 \\ x^6+x^4+x^3+x^2+x \end{bmatrix}$$

Erste Zelle:

$$\begin{aligned} b0 &= x(x^7 + x^6 + x^4 + x^2 + 1) \oplus \\ &\quad (x + 1)(x^6 + x^3 + x + 1) \oplus \\ &\quad (x^7 + x^5 + x^3 + x^2 + x + 1) \oplus \\ &\quad (x^6 + x^4 + x^3 + x^2 + x) \\ &= ((x^8 + x^7 + x^5 + x^3 + x) \bmod (x^8 + x^4 + x^3 + x + 1)) \oplus \\ &\quad ((x^7 + x^6 + x^4 + x^3 + x^2 + 1) \bmod (x^8 + x^4 + x^3 + x + 1)) \oplus \\ &\quad ((x^7 + x^5 + x^3 + x^2 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1)) \oplus \\ &\quad ((x^6 + x^4 + x^3 + x^2 + x) \bmod (x^8 + x^4 + x^3 + x + 1)) \\ &= (x^7 + x^4 + x^3 + x^2 + 1) \\ &= (9D)_{16} \end{aligned}$$

...

Die MixColumns-Operation schafft somit eine vertikale Abhängigkeit der einzelnen Zellen, da alle 4Bytes einer Spalte in die Berechnung jedes einzelnen Ergebnis-Bytes mit einfließen. In Kombination mit der Verschiebung von ShiftRows ergibt sich somit eine horizontale und vertikale Abhängigkeit. Das bedeutet: Wenn man einen Klartext verschlüsselt und einen zweiten Klartext mit dem selben Schlüssel verschlüsselt, allerdings 1 Bit im Klartext ändert, so hat dies Auswirkungen auf die Werte aller Bytes des Chiffrats im Vergleich zum ersten Chiffrat.

7.2 Warum diese Struktur?

Die verschiedenen Funktionen lassen sich in 2 Gruppen teilen: ByteSubstitution, ShiftRows und MixColumns sorgen für eine lineare Permutation des Textes. Alleine sind diese aber zurückrechenbar. Deshalb wird zusätzlich die Funktion KeyAddition benötigt, die dann die gesamte Verschlüsselung vom Benutzerschlüssel abhängig macht.

7.2.1 Lineare Permutation

Bei der linearen und differentiellen Kryptoanalyse (s.u.) wird beobachtet, wie sich die Ausgabe der Verschlüsselung ändert, wenn sich die Eingabe minimal ändert. Die Lineare Permutation sorgt unter anderem dafür, gegen dieses Verfahren immun zu sein.
Die drei Funktionen sorgen dafür, dass jede Zeile sehr stark mit allen anderen Zellen des Blockes gemischt wird.

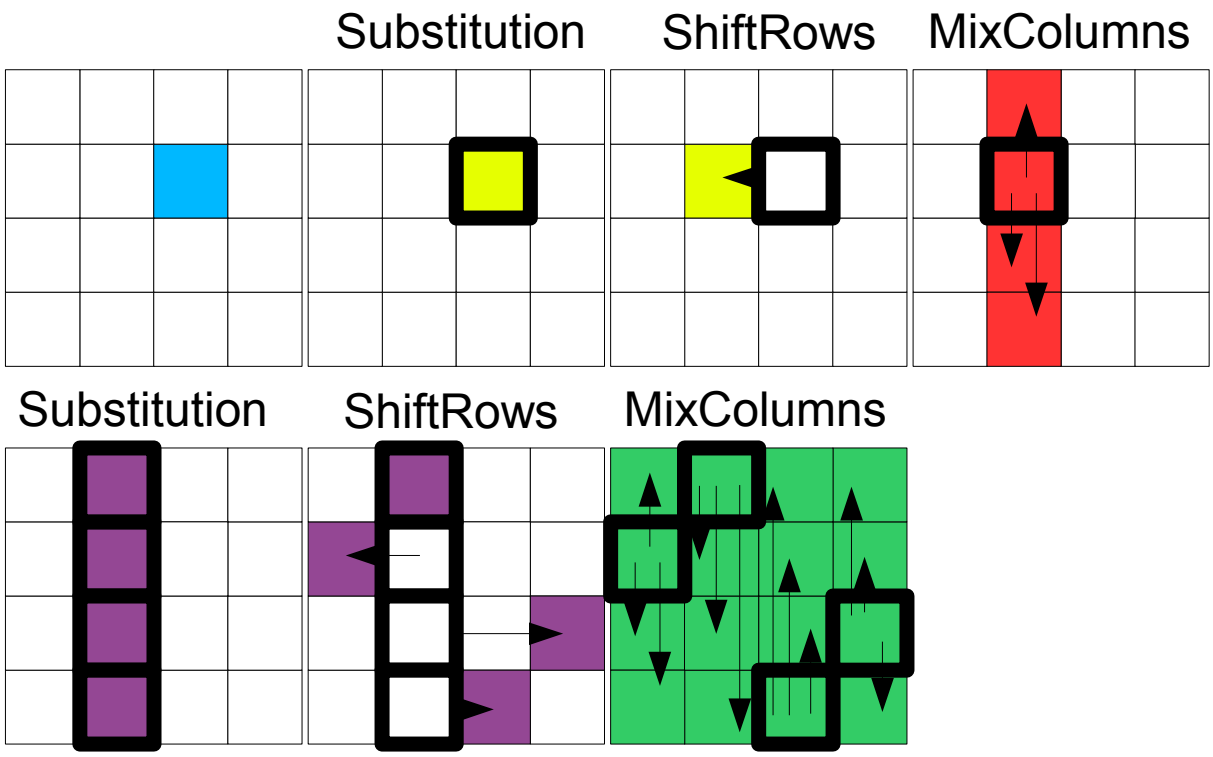


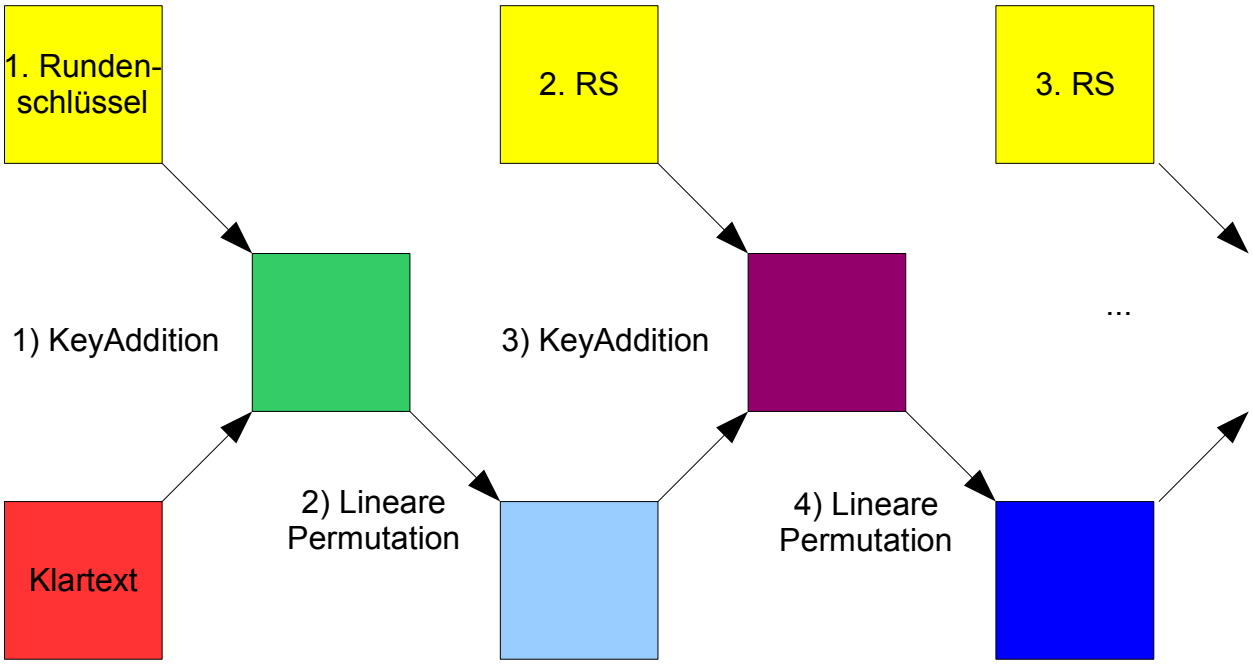
Abbildung 5: Ablauf der Linearen Permutation

Beobachtet man nur die Zellen, die direkte Auswirkungen durch Anwenden der drei Funktionen auf eine Zelle haben, so sieht man, dass nach der ersten Runde die Ursprungszelle bereits auf 4 Zellen verteilt ist. Nach der zweiten Runde hat sie Auswirkungen auf alle Zellen im Block. Da dies nicht nur mit einer Zelle über 2 Runden geschieht, sondern auf allen Zellen über 10 bis 14 Runden, sieht man, dass alle Werte sehr gut gemischt werden.

7.2.2 Schlüsseladdition

Da die lineare Permutation komplett zurückrechenbar ist, wird zwischen jeder Runde die Schlüsseladdition ausgeführt. Diese ist nicht zurückrechenbar ohne den Schlüssel, und

schottet die lineare Permutation ab, wodurch auch diese nicht zurückrechenbar ist ohne Schlüssel. Zusammen ergibt sich folgende Struktur:



8 Arbeitsweise und Ablaufstruktur des AES bei Entschlüsselung

Um ein Chiffrat wieder zu Entschlüsseln, wird das gesamte Verschlüsselungsverfahren einfach rückwärts durchlaufen. Da es sich um ein symmetrisches Verfahren handelt, wird der selbe Schlüssel wie bei der Verschlüsselung verwendet. Die Reihenfolge der Funktionen muss also genau umgekehrt, wie bei der Verschlüsselung sein. Damit sich die Funktionen umkehren lassen, sind einige Funktionen stellenweise verändert gegenüber den Verschlüsselungs-Operationen. Diese Funktionen tragen dann das Präfix „Inv“ für Invers.

Der grobe Ablauf des Algorithmus während der Entschlüsselung ist rechts dargestellt.

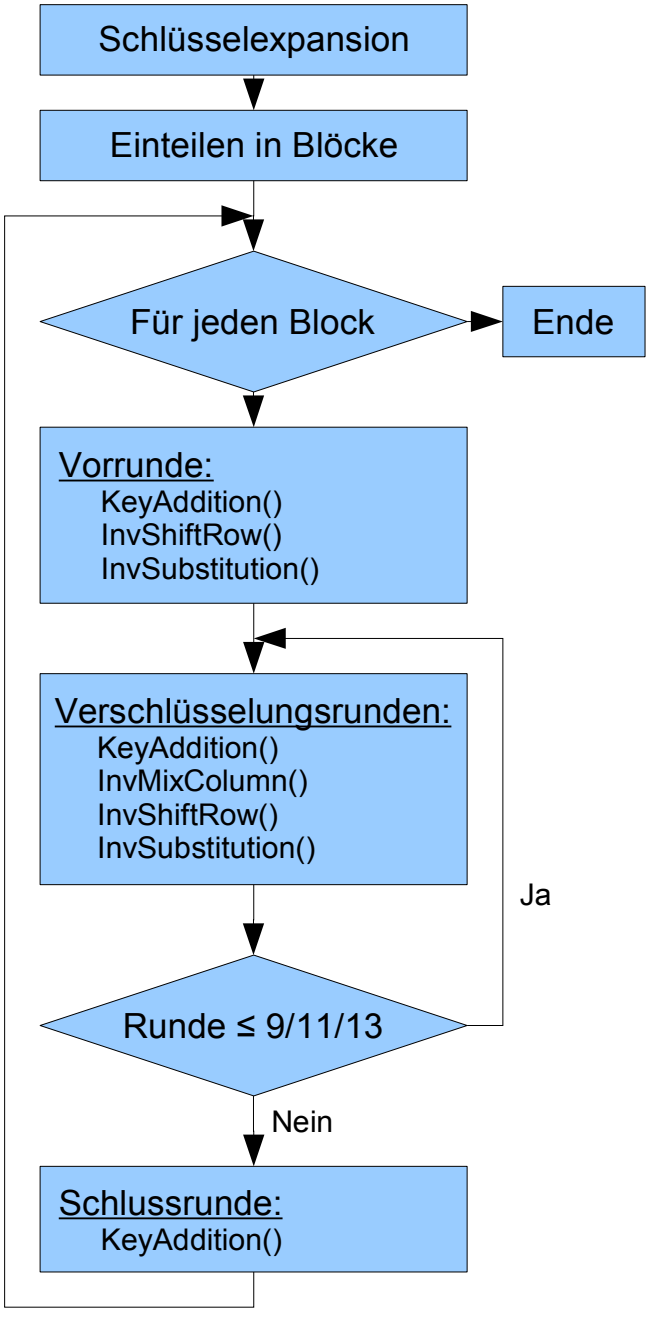


Abbildung 6: Ablauf der Entschlüsselung

8.1 Definition und Spezifikation der einzelnen inversen Funktionen

8.1.1 Schüsselexpansion

Die Schlüsselexpansion läuft bei der Entschlüsselung exakt so ab wie bei der Verschlüsselung. Auch die sich ergebenden Rundenschlüssel sind die selben, da es sich ja um ein symmetrisches Verfahren handelt. Zu beachten ist nur, dass nicht mit dem ersten Rundenschlüssel begonnen wird, sondern logischerweise mit dem letzten.

8.1.2 KeyAddition

Auch die KeyAddition läuft exakt so ab wie bei der Verschlüsselung. Eine Eigenschaft der XOR Operation ist, wenn 2 der 3 Komponenten Operand A, Operand B und Ergebnis E einer XOR-Operation vorhanden sind, dass sich durch erneute XOR-Operation der fehlende Teil rekonstruieren lässt. Somit ist es möglich mit dem Schlüssel und dem Chiffrat den Klartext wieder herzustellen.

8.1.3 InvSubstitution

Die Substitution verläuft nur vom Prinzip her genau so wie bei der Verschlüsselung. Um die umgekehrte Zuordnung wieder herzustellen, wird eine inverse S-Box verwendet. Es wäre natürlich auch möglich mit der selben S-Box zu arbeiten, nur müsste dann jedes Mal das entsprechende Element unter allen 256 Elementen linear gesucht werden, was grobe Performance-Einbußen zur Folge hätte.

8.1.4 InvShiftRow

Auch die ShiftRow-Operation ist es leicht umzukehren. Die Verschiebungen werden nach dem selben Schema wie bei der Verschlüsselung durchgeführt, nur wird nach rechts statt nach links verschoben.

8.1.5 InvMixColumn

Die InvMixColumn-Operation verläuft vom Schema her ebenfalls gleich der MixColumn-Operation. Um die Operation umkehren zu können, muss das Polynom (konstant) mit dem ein Byte-Vektor multipliziert werden, um den Ergebnis-Byte-Vektor zu erhalten, invertiert werden.
Somit gilt:

$$c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$
$$c(x)^{-1} = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$$

Somit bildet sich natürlich auch eine andere Multiplikations-Matrix.

$$c(x) = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$$

Beispiel:
Der Einfachheit halber wird anstatt auf Wort-Basis, auf Byte-Basis gerechnet.
($M(x) = x^8 + x^4 + x^3 + x + 1$)

$$InvMixColumns\begin{pmatrix} 9D & 28 & 91 & 00 \\ F7 & 7F & 78 & A6 \\ 39 & C1 & 6C & C6 \\ 3C & AA & 25 & A5 \end{pmatrix} = \begin{pmatrix} D5 & C8 & 56 & D5 \\ 4B & A6 & E8 & DD \\ AF & 59 & F0 & 82 \\ 5E & 0B & EE & 4F \end{pmatrix}$$

Erste Spalte:

$$a(x) = \begin{bmatrix} 9D \\ F7 \\ 39 \\ 3C \end{bmatrix} = \begin{bmatrix} 10011101 \\ 11110111 \\ 00111001 \\ 00111100 \end{bmatrix} \equiv \begin{bmatrix} x^7+x^4+x^3+x^2+1 \\ x^7+x^6+x^5+x^4+x^2+x+1 \\ x^5+x^4+x^3+1 \\ x^5+x^4+x^3+x^2 \end{bmatrix}$$

$c(x)^{-1} = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$

$$c(x)^{-1} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} = \begin{bmatrix} 1110 & 1011 & 1101 & 1001 \\ 1001 & 1110 & 1011 & 1101 \\ 1101 & 1001 & 1110 & 1011 \\ 1011 & 1101 & 1001 & 1110 \end{bmatrix}$$

$$\equiv \begin{bmatrix} x^3+x^2+x & x^3+x+1 & x^3+x^2+1 & x^3+1 \\ x^3+1 & x^3+x^2+x & x^3+x+1 & x^3+x^2+1 \\ x^3+x^2+1 & x^3+1 & x^3+x^2+x & x^3+x+1 \\ x^3+x+1 & x^3+x^2+1 & x^3+1 & x^3+x^2+x \end{bmatrix}$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} x^3+x^2+x & x^3+x+1 & x^3+x^2+1 & x^3+1 \\ x^3+1 & x^3+x^2+x & x^3+x+1 & x^3+x^2+1 \\ x^3+x^2+1 & x^3+1 & x^3+x^2+x & x^3+x+1 \\ x^3+x+1 & x^3+x^2+1 & x^3+1 & x^3+x^2+x \end{bmatrix} \cdot \begin{bmatrix} x^7+x^4+x^3+x^2+1 \\ x^7+x^6+x^5+x^4+x^2+x+1 \\ x^5+x^4+x^3+1 \\ x^5+x^4+x^3+x^2 \end{bmatrix}$$

Erste Zelle:

$b0 = (x^3 + x^2 + x)(x^7 + x^4 + x^3 + x^2 + 1) \oplus$
$(x^3 + x + 1)(x^7 + x^6 + x^5 + x^4 + x^2 + x + 1) \oplus$
$(x^3 + x^2 + 1)(x^5 + x^4 + x^3 + 1) \oplus$
$(x^3 + 1)(x^5 + x^4 + x^3 + x^2)$
$= ((x^{10} + x^9 + x^8 + x^7 + x^5 + x^2 + x) \bmod (x^8 + x^4 + x^3 + x + 1)) \oplus$

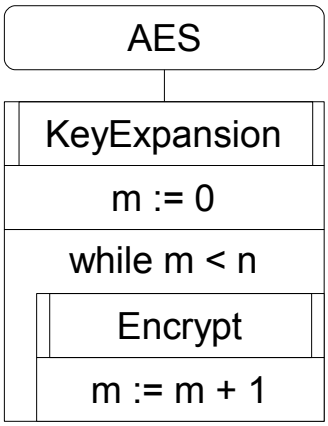
Der Advanced Encryption Standard (AES) als symmetrisches Kryptosystem

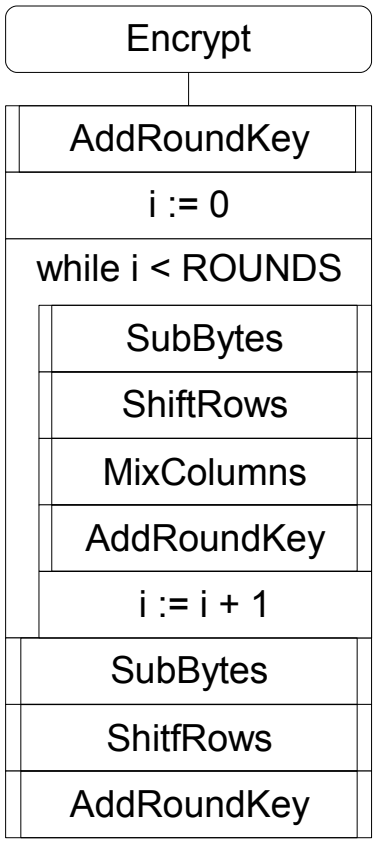
$$
\begin{aligned}
& ((x^{10} + x^9 + x^7 + x^5 + 1) \bmod (x^8 + x^4 + x^3 + x + 1)) \oplus \\
& ((x^8 + x^4 + x^2 + 1) \bmod (x^8 + x^4 + x^3 + x + 1)) \oplus \\
& ((x^8 + x^7 + x^6 + x^4 + x^3 + x^2) \bmod (x^8 + x^4 + x^3 + x + 1)) \\
= \quad & (x^7 + x^6 + x^4 + x^2 + 1) \\
= \quad & (D5)_{16}
\end{aligned}
$$

...

9 Asymptotische Analyse

Bei der Asymptotischen Analyse gibt es 2 Werte zu beachten: Zum einen „n“, das die Anzahl der zu verschlüsselnden Blöcke beinhaltet. Dieses ist das Wachstumskriterium der Analyse. Zum anderen „ROUNDS“, das die Anzahl der Runden während der Verschlüsselung wiedergibt. Dieser Wert liegt bei 9, 11 oder 13, abhängig von der Länge des Benutzerschlüssels.
Der Ablauf des Algorithmus sieht so aus:





Die Laufzeiten der einzelnen Funktionen im Überblick:

9.1 KeyExpansion

Bei der Schlüsselexpansion müssen im Best Case 176 Bytes errechnet werden, im Worst Case 224 Bytes. Der Wert ist aber nicht abhängig von n, daher ist die Laufzeit dieser Funktion konstant.

9.2 AddRoundKey

Hier werden alle Bytes des Rundenschlüssels auf den aktuellen Block addiert. Im Best Case sind dies 16 Operationen, im Worst Case 32. Da dieser Wert nicht von n abhängig ist, ist die Laufzeit dieser Funktion konstant.

9.3 SubBytes

Hier werden alle Bytes im aktuellen Block anhand der S-Box ersetzt, wie bei der Funktion AddRoundKey sind dies im Best Case 16, im Worst Case 32 Operationen, also ist die Laufzeit auch hier konstant.

9.4 ShiftRows

Hier werden die Werte aus drei der vier Zeilen des Blockes verschoben. Da die Anzahl der zu verschiebenden Zellen nicht von n abhängt, ist die Laufzeit auch hier konstant.

9.5 MixColumns

Hier werden die vier Werte jeder Spalte des Blockes gemischt. Im Best Case sind das vier Spalten, im Worst Case 8. Da dies aber nicht von n abhängt ist die Laufzeit konstant.

9.6 Schleife while i < ROUNDS

Während der Schleife werden die vier Funktionen AddRoundKey, SubBytes, ShiftRows und MixColumns aufgerufen. Die Laufzeit dieser Funktionen ist konstant. Wie oft dies geschieht, hängt vom Wert ROUNDS ab. Da dieser Wert konstant 9 bis 13 ist, ist die Laufzeit der gesamten Schleife weiterhin konstant.

9.7 Encrypt

Da alle Funktionen, die in Encrypt aufgerufen werden, sowie die Schleife alle konstante Laufzeit haben, ist die Laufzeit auch hier weiterhin konstant.

9.8 while m < n

In der Schleife wird die Funktion Encrypt aufgerufen. Die Laufzeit dieser Funktion ist konstant, aber da die Schleife n-mal aufgerufen wird, ist die Laufzeit direkt von n abhängig. Diese Schleife hat also eine linear wachsende Laufzeit.

9.9 AES

Zusätzlich zur Schleife wird während der Verschlüsselung die Funktion KeyExpansion einmal aufgerufen. Diese hat zwar konstante Laufzeit, aber da die folgende Schleife linear wachsende Laufzeit hat, wirkt sich diese nicht aus. Die Laufzeit der gesamten Verschlüsselung ist also linear wachsend, abhängig von der Anzahl der zu verschlüsselnden Blöcke.

10 Angriffe

Bisher sind noch keine erfolgreichen Angriffe auf den AES erfolgt. Da der Algorithmus aber während der Ausschreibung (und natürlich auch danach) ausgiebig getestet wurde, sind natürlich einige Ansätze vorhanden.

10.1 Brute Force

Dies ist wohl der einfachste Ansatz. Es werden einfach alle möglichen Schlüssel sukzessive durchprobiert. Die einfachste Lösung dagegen ist es, so viele verschiedene Schlüssel zu erlauben, dass das Durchprobieren nicht in brauchbarer Zeit möglich ist. Vergleicht man den Vorgänger DES und den AES auf seine Schlüssellängen, so sieht man, dass der Schlüssel des AES mindestens 72 Bit länger ist, wodurch sich 2^{72} mehr Kombinationen ergeben, und man entsprechend auch 2^{72} mal mehr Zeit benötigt

Beim schnellsten Angriff auf den DES wurden 22 Stunden benötigt, rechnet man dies hoch auf den AES so würden derzeit 11 Trillionen Jahre benötigt, und auch wenn der DES in einer Sekunde zu knacken wäre, würden beim AES hoch gerechnet immer noch 150 Billionen Jahre benötigt. Heute ist es daher praktisch unmöglich den AES auf diese Weise zu knacken, allerdings galten für den DES vor 20 Jahren wohl ähnliche Zahlen.

10.1.1 Reduktion der möglichen Permutationen

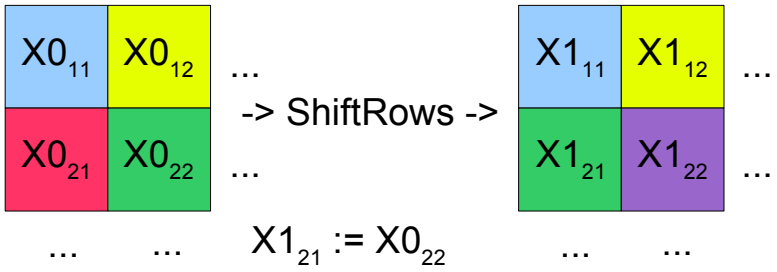
Dieser Angriff gehört mit zur Brute Force Attacke und hat es unter anderem möglich gemacht den DES zu knacken.
Es geht dabei darum, die möglichen Permutationen, die durch verschiedene Schlüssel möglich sind, zu reduzieren. So wurde beispielsweise beim DES entdeckt, dass zu jedem Schlüssel ein inverser Schlüssel existiert, der auch den inversen Geheimtext liefert. Dadurch brauchen also die Hälfte aller Schlüssel nicht ausprobiert werden. Zusammen mit einigen anderen Spezialfällen konnte der DES dadurch soweit vereinfacht werden, dass es möglich wurde den DES zusammen mit der gesteigerten Rechenleistung zu knacken. Beim AES wurden solche Fälle bisher aber noch nicht entdeckt.

10.2 Lineare und Differentielle Kryptoanalyse

Bei der Linearen und Differentiellen Kryptonalayse geht es darum, die Ausgabe eines Verschlüsselungsalgorithmus zu beobachten, wenn sich an der Eingabe eine Kleinigkeit ändert. Dieses Fachgebiet der Kryptographie wurde entdeckt, nachdem Verschlüsselungsalgorithmen wie der DES öffentlich diskutiert wurden. Wie mittlerweile bekannt wurde, waren aber Geheimdiensten wie der NSA diese Verfahren bereits vorher bekannt. Da dieses Verfahren bereits bekannt war, als der Rijndael Algorithmus entwickelt wurde, wurde er speziell dagegen geschützt durch die so genannte Wide-Trail Strategie. Wie schon vorher gezeigt, wird der zu verschlüsselnde Block sehr stark ineinander gemischt, so dass jedes Eingabebyte Auswirkungen auf jedes Byte der Ausgabe hat. Ändert man also auch nur eine Kleinigkeit an der Eingabe, so ändert sich im Optimalfall die gesamte Ausgabe.
Um auf einen Schlüssel Rückschlüsse ziehen zu können, braucht man derzeit so viele Klar- und Geheimtexte, dass kein Rechner diese Datenmenge verarbeiten könnte.

10.3 XSL-Angriff

Dieser Angriff wurde speziell für den AES entwickelt. Der gesamte Algorithmus basiert auf Mathematik, und mittlerweile ist es vollständig gelungen, diese Mathematik als ein System aus 8000 Gleichungen und 1600 Variablen darzustellen.



Beispiel: Beim ShiftRows verschiebt sich die zweite Zeile um eine Zelle nach links. Eine (einfache) Formel könnte also lauten: $X1_{21} := X0_{22}$
Führt man dies sukzessive durch, so erhält man 8000 Gleichungen.

Dieses System aus Gleichungen lässt sich vereinfachen, so dass die Lösungen des Systems theoretisch schneller möglich ist als eine reine Brute Force Attacke. Praktisch ist aber auch dieser Angriff derzeit noch nicht in brauchbarer Zeit möglich.

11 Literatur

Primärliteratur:

- J. Daemen, V. Rijmen, „The Design of Rijndael“ - „AES – The Advanced Encryption Standard“

Allgemeine Recherche:

- http://de.wikipedia.org/wiki/Advanced_Encryption_Standard (Achtung: nicht fehlerfrei)

Visualisierung

- http://home.datacomm.ch/th.aes/