

Seminararbeit

Thema: Kryptographische Verfahren

**Irina Tsalman
Maxim Mariach
Eugen Hofmann**

WS 2002

Inhaltsverzeichnis

1 Vorwort	3
2 Grundbegriffe	4
3 Kryptoanalyse.....	6
4 Grundlegende Verschlüsselungsmethoden	8
4.1 Transpositionsmethoden.....	8
4.1.1 Einfache Blocktransposition.....	8
4.1.2 Zeilentransposition.....	8
4.2 Substitutionsmethoden	8
4.2.1 Monoalphabetische Substitutionsmethoden.....	9
4.2.2 Homophonische Substitutionsmethoden.....	9
4.2.3 Polyalphabetische Substitutionsmethoden.....	9
4.2.4 „Sichere“ Substitutionsmethoden.....	9
5 Symmetrische Kryptosysteme	10
5.1 Vor- und Nachteile	10
5.2 DES-Algorithmus	12
5.2.1 Arbeitsweise von DES	13
5.2.2 Betriebsmodi von DES	15
5.2.2.1 ECB	15
5.2.2.2 CBC	15
5.2.2.3 CFB.....	16
5.2.2.4 OFB	16
5.2.3 Triple DES (3DES).....	17
5.2.4 Sicherheit und Schwächen von DES	18
5.3 AES	19
5.3.1 Arbeitsweise von AES	20
5.3.2 Kryptographische Stärke von AES	21
5.3.3 Kryptoanalyse von AES	22
5.4 IDEA.....	23
5.4.1 Arbeitsweise von IDEA.....	24
5.4.2 Sicherheit von IDEA	25
6 Asymmetrische Kryptosysteme	26
6.1 Vor- und Nachteile	26
6.2 RSA	27
6.2.1 Schlüsselerzeugung	27
6.2.2 Verschlüsselung.....	28
6.2.3 Entschlüsselung.....	28
6.2.4 Kryptoanalyse	28
6.2.5 Wahl von p und q	29
7 Hybride Kryptosysteme	30
8 Hashfunktionen.....	31
8.1 Grundbegriffe	31
8.2 Sicherheitsrisiken - Geburtstags-Attacke	32
8.3 MD5	32
8.3.1 Arbeitsweise	33
8.3.2 Beispiel einer Kollision.....	35
8.4 SHA-1.....	36
8.4.1 Arbeitsweise	36
8.4.2 Sicherheit von SHA.....	39

9 Kryptographische Anwendungen	40
9.1 Digitale Signatur.....	40
9.1.1 RSA-Signaturen.....	41
9.1.1.1 Schlüsselerzeugung	41
9.1.1.2 Erzeugung der Signatur	41
9.1.1.3 Verifikation.....	41
9.1.2 Angriffe	42
9.2 Blinde Unterschrift	42
9.3 Digitales Geld.....	43
Literatur	44

1 Vorwort

Seit Menschen sprechen und schreiben können, wünschen sie, Nachrichten so übertragen zu können, dass Außenstehende den Inhalt weder verstehen noch unbemerkt verändern können. Kryptographie und Kryptologie stellen die Lehre vom geheimen Schreiben dar.

Anfänge der Kryptologie sind aus Ägypten und Mesopotamien bekannt. Von dort sind uns Funde überliefert, die in verschlüsselter Form Anweisungen zur Bearbeitung von Tontöpfen enthalten.

Auch von Staaten wurde das geheime Schreiben für diplomatische und militärische Zwecke früh benutzt. Aus Griechenland und Rom sind dazu einige Beispiele bekannt.

Nach dem Verfall des römischen Reiches und dessen Kultur wurde die Kryptologie bis in das Mittelalter hinein nicht mehr verwendet. Erst seit der Renaissancezeit wird die Kryptologie weiter entwickelt. Es geht dabei immer um das Verschlüsseln von Nachrichten und ihre Entschlüsselung, die in ständigem Wettstreit liegen.

In der heutigen Welt spielt die Information und der Informationsaustausch eine zentrale Rolle. Der Schutz von Informationen und das Wissen über Kryptologie wird immer wichtiger.

Heute kommt der Kryptologie durch die weite Verbreitung und Vernetzung von Computern eine immer stärkere Bedeutung zu. So wie man Briefe zuklebt, die empfangenen Briefe und Akten wegschließt, so möchte man in ähnlicher Weise elektronische Daten vor unerlaubter Einsicht schützen.

Wir wollen in dieser Seminararbeit nach einführenden Begriffen erst einige ältere Verfahren der Kryptographie beschreiben, um danach auf verschiedene neuere Verfahren einzugehen, wie sie heute in der Praxis benutzt werden.

2 Grundbegriffe

In diesem Kapitel führen wir Grundbegriffe ein, die die Beschreibung von Verschlüsselungsverfahren ermöglichen.

Kryptologie

Der Begriff *Kryptologie* stammt aus dem Griechischen und setzt sich aus „kryptós“ für geheim, verborgen, versteckt und „lógos“ für Wort, Wissen zusammen.

Die *Kryptologie* ist ein Teilgebiet der Mathematik, das sich mit dem Verbergen von Informationen beschäftigt. Sie umfasst die Kryptografie und die Kryptoanalyse.

Kryptographie

Die *Kryptografie* beschäftigt sich mit der Entwicklung von Algorithmen zur Verschlüsselung von Informationen. Als Wissenschaft befasst sich die *Kryptografie* dazu mit der Entwicklung von Kryptosystemen bzw. den Verfahren zur Verschlüsselung und (befugten) Entschlüsselung von Daten.

Kryptoanalyse

Die *Kryptoanalyse* ist das Teilgebiet der Kryptologie, das sich mit der Kompromittierung kryptografischer Verfahren beschäftigt. Kryptoanalytische Methoden werden auch verwendet, um die Sicherheit kryptografischer Verfahren zu testen.

Kryptosystem

Kryptosystem ist ein System, das es ermöglicht, Nachrichten zu verschlüsseln und entschlüsseln.

Mathematisch gesehen ist ein Kryptosystem ein 5-Tupel (P, C, K, E, D) , wobei:

- P ist ein Klartextrraum (engl. *Plaintext Space*). Seine Elemente heißen Klartexte.
- C ist ein Schlüsseltextrraum (engl. *Cipher text Space*). Seine Elemente heißen Chiffretexte.
- K ist ein Schlüsselraum (engl. *Key Space*). Seine Elemente heißen Schlüssel.
- E ist eine Familie von Funktionen $E_K: P \rightarrow C$. Ihre Elemente heißen Verschlüsselungsfunktionen.
- D ist eine Familie von Funktionen $D_K: C \rightarrow P$. Ihre Elemente heißen Entschlüsselungsfunktionen.
- Für jedes $e \in K$ gibt es ein $d \in K$, so dass für alle $p \in P$ die Gleichung $D_d(E_e(p)) = p$ gilt.

Ein „gutes“ Kryptosystem soll folgende Eigenschaften besitzen:

- Gegeben E_e und p , soll es einfach sein $c = E_e(p)$ zu berechnen.
- Gegeben D_d und c , soll es einfach sein $p = D_d(c)$ zu berechnen.
- Der Chiffretext $c = E_e(p)$ soll nicht viel länger sein als der Klartext p .

- Geheimhaltungsanforderungen: Es darf nicht einfach möglich sein, dass ein Angreifer
 1. systematisch die Entschlüsselungstransformation D_d aus Chiffretext c bestimmt, selbst wenn Klartext p bekannt ist.
 2. systematisch Klartext p aus Chiffretext c bestimmt.
- Authentizitätsanforderungen: Es darf nicht einfach möglich sein, dass ein Angreifer
 1. systematisch die Entschlüsselungstransformation E_e aus Chiffretext c bestimmt, selbst wenn Klartext p bekannt ist.
 2. systematisch Chiffretext c' findet, so dass $D_d(c')$ gültiger Klartext aus P ist.

Prinzip von Kerckhoffs

Die Sicherheit eines Kryptosystems hat nicht von der Geheimhaltung der Verschlüsselungstransformationen und Entschlüsselungstransformationen E und D abzuhängen, sondern ausschließlich von der Geheimhaltung der verwendeten Schlüssel $k \in K$.

3 Kryptoanalyse

Kryptoanalyse ist die Lehre von den Angriffen auf Verschlüsselungsverfahren. In diesem Kapitel werden diese Angriffe grob klassifiziert.

Um Angriffe auf Verschlüsselungsverfahren zu erschweren, kann man das verwendete Verfahren geheim halten. Die Sicherheit ist aber sehr zweifelhaft. Ein Angreifer hat viele Möglichkeiten, zu erfahren, welches Kryptosystem verwendet wird. Aus diesen Gründen bezieht die moderne Kryptoanalyse einen anderen Standpunkt. Sie geht davon aus, dass jeder weiß, welches Kryptosystem verwendet wird. Nur der verwendete Schlüssel und die verschlüsselten Klartexte werden als geheim angenommen. Das Ziel des Angreifers ist es, die Klartexte oder den Schlüssel zu finden. Durch mathematische Analyse des Verschlüsselungsverfahrens versucht man herauszufinden, ob das schwierig ist oder nicht.

Folgende Angriffstypen lassen sich grundsätzlich unterscheiden:

- *Ciphertext-Only-Attacke*: Der Angreifer kennt nur Chiffretexte und versucht daraus, die zugehörigen Klartexte oder Schlüssel zu bestimmen. Die „Ciphertext-only“-Angriffe stellen die schwächsten kryptoanalytischen Angriffe dar, sie finden sich aber in der Praxis am häufigsten.
- *Known-Plaintext-Attacke*: Der Angreifer kennt einige Klartext-Chiffretext-Paare und sucht den verwendeten Schlüssel oder versucht, andere Chiffretexte zu entschlüsseln. „Known-plaintext“-Angriffe stellen in vernetzten und verteilten Systemen eine ernstzunehmende Gefahr dar. Zum einen sind Format und Kodierung von Protokolldateneinheiten meist standardisiert und öffentlich bekannt, und zum anderen lassen sich die einzelnen Nachrichtenkomponenten relativ leicht aus diesen Dateneinheiten extrahieren.
- *Chosen-Plaintext-Attacke*: Der Angreifer kann die Chiffretexte zu selbst gewählten Klartexten erzeugen, kennt aber den Schlüssel nicht. Er sucht den verwendeten Schlüssel oder versucht, andere Chiffretexte zu entschlüsseln. „Chosen-plaintext“-Angriffe sind z.B. dann möglich, wenn Anfragen automatisch in die entsprechenden Antworten integriert und mitchiffriert werden.
- *Chosen-Ciphertext-Attacke*: Der Angreifer kann selbst gewählte Chiffretexte entschlüsseln, ohne den Schlüssel zu kennen. Er sucht den verwendeten Schlüssel. „Chosen-ciphertext“-Angriffe spielen eigentlich nur im Zusammenhang mit asymmetrischen Kryptosystemen eine Rolle.

Weitere Angriffsarten sind in der Regel mathematisch komplizierter, zu nennen wären:

- *Differentielle Kryptoanalyse*: Dieses Verfahren basiert auf der Untersuchung sehr ähnlicher Klartexte und den Unterschieden in den dazugehörigen Verschlüsselungen, um damit Rückschlüsse auf den Verschlüsselungsalgorithmus ziehen zu können.
- *Lineare Kryptoanalyse*: Dieses Verfahren basiert auf linearen, statistischen Relationen zwischen Klartext, Geheimtext und Schlüssel. Der Angreifer verfügt über beliebig viele (gestohlene) Klartexte und die zugehörigen, mit dem unbekanntem Schlüssel chiffrierten (abgefangenen) Geheimtexte.

- *Timing Attack*: Dieser Angriff benötigt eine Zeitmessung für kryptographische Operationen, aus dem Zeitbedarf lassen sich Rückschlüsse auf die Schlüssellänge und -beschaffenheit ziehen.
- *Power Analysis-Attacken*: Diese Angriffe 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.

Ein Kryptosystem wird als *bedingungslos sicher* bezeichnet, wenn unter Zuhilfenahme eines beliebig langen Chiffretextes eine Kryptoanalyse nicht möglich ist. Ein solches System weist die Eigenschaft perfekter Geheimhaltung auf. Leider ist nur ein Kryptosystem bekannt, das sicher ist. Es handelt sich um den One-Time-Pad, der später vorgestellt wird.

In der Praxis begnügt man sich mit *berechenbar sicheren* Kryptosystemen, bei denen die bestmöglichen Angriffe die Möglichkeiten eines Kryptoanalisten in jedem Fall übersteigen.

Man bezeichnet ein Kryptosystem als *praktisch sicher*, wenn der bestbekannteste Angriff nicht einfacher ist, als die vollständige Schlüsselsuche.

4 Grundlegende Verschlüsselungsmethoden

Als einfache Verschlüsselungsverfahren werden in den beiden Unterkapiteln Transpositionsmethoden und Substitutionsmethoden vorgestellt.

4.1 Transpositionsmethoden

Im Rahmen einer *Transpositionsmethode* werden die Zeichen des Klartextes umgeordnet.

4.1.1 Einfache Blocktransposition

Im Rahmen einer *Blocktransposition* wird ein Klartext $P = p_1 \dots p_d p_{d+1} \dots p_{2d}$ in Blöcke der Länge d aufgeteilt. Wenn π eine Permutation der Zahlen $\{1, 2, \dots, d\}$ darstellt, kann das Paar $k=(d, \pi)$ als Schlüssel zur Verschlüsselung von P herangezogen werden. Den Chiffretext bekommt man dann durch Anwendung der Permutation π auf den Klartext, d.h.

$$C = E_{(d, \pi)}(P) = p_{\pi(1)} \dots p_{\pi(d)} p_{\pi(d+1)} \dots p_{\pi(2d)}.$$

Die Entschlüsselung wird durch eine zu π inverse π^{-1} festgelegt.

4.1.2 Zeilentransposition

Klartext wird zeilenweise in eine Matrix geschrieben.

Chiffretext erhält man durch Spalten in einer bestimmten Folge.

Die Kryptoanalyse einer Zeilentransposition ist relativ einfach, weil benachbarte Klartextzeichen im Chiffretext einen konstanten Abstand haben. Man versucht mit Hilfe der Häufigkeitsverteilung von Zeichenpaaren die Zeichen an ihre ursprünglichen Positionen zurückzuspeichern (*Anagramming*).

4.2 Substitutionsmethoden

Im Rahmen einer *Substitutionsmethode* werden die Zeichen eines Klartextalphabetes durch die Zeichen eines (oder mehrere) Chiffretextalphabetes ersetzt.

4.2.1 Monoalphabetische Substitutionsmethoden

Bei einer *monoalphabetischen Substitutionsmethode* werden die Zeichen eines Klartextalphabetes durch die Zeichen eines einzigen Chiffretextalphabetes ersetzt.

Die Cäsar-Verschlüsselung ist wohl die älteste monoalphabetische Substitutionsmethode. Diese Methode verschlüsselt einen Klartext, indem sie jedes Klartextzeichen durch das in diesem Alphabet drei Position weiter liegende Chiffretextzeichen ersetzt, d.h. $c = E_3(p) = p + 3 \bmod 26$. Den Klartext erhält man durch Anwendung $D_3(c) = c - 3 \bmod 26$.

4.2.2 Homophonische Substitutionsmethoden

Bei einer *homophonischen Substitutionsmethode* werden die Zeichen eines Klartextalphabetes durch paarweise disjunkte Mengen von Chiffretextzeichen ersetzt. Bei der Verschlüsselung werden dann zufällig Elemente aus den jeweiligen Homophonmengen herausgegriffen und als Chiffretextzeichen eingesetzt.

4.2.3 Polyalphabetische Substitutionsmethoden

Bei einer *polyalphabetischen Substitutionsmethode* werden mehrere voneinander unabhängige Chiffretextalphabete eingesetzt, um die Häufigkeitsverteilungen von Buchstaben und Buchstabengruppen in einem Chiffretext auszugleichen.

4.2.4 „Sichere“ Substitutionsmethoden

Das bekannteste Kryptosystem, dessen perfekte Sicherheit man beweisen kann, ist das *Vernam-One-Time-Pad*. Sei n eine natürliche Zahl. Das Vernam-One-Time-Pad verschlüsselt Bitstrings der Länge n . Es ist $P = C = K = \{0, 1\}^n$.

Die Verschlüsselung zum Schlüssel $k \in \{0, 1\}^n$ ist:

$$E_k: \{0, 1\}^n \rightarrow \{0, 1\}^n, \quad p \rightarrow p \oplus k.$$

Die Entschlüsselungsfunktion zum Schlüssel k sieht genauso aus.

Diese Idee, Texte zu verschlüsseln wurde 1917 von Gilbert Vernam erfunden und patentiert. Aber erst 1949 bewies Shannon, dass das Vernam-One-Time-Pad perfekt sicher ist.

Leider ist dieses Verfahren nicht effizient. Kommunikationspartner müssen für jeden neuen Block der Länge n einen neuen Schlüssel der Länge n zufällig erzeugen und austauschen.

5 Symmetrische Kryptosysteme

In einem *symmetrischen Kryptosystem* verwenden der Sender und der Empfänger den gleichen Schlüssel, den sie vor Beginn der eigentlichen Kommunikation ausgetauscht haben. Wenn Sender und Empfänger zwar über unterschiedliche Schlüssel verfügen, diese aber in einer einfachen funktionalen Beziehung stehen, dann spricht man ebenfalls von einem symmetrischen Kryptosystem.

Der Sender benutzt den geheimen Schlüssel, um die Nachricht zu verschlüsseln, und der Empfänger, um diese zu entschlüsseln.

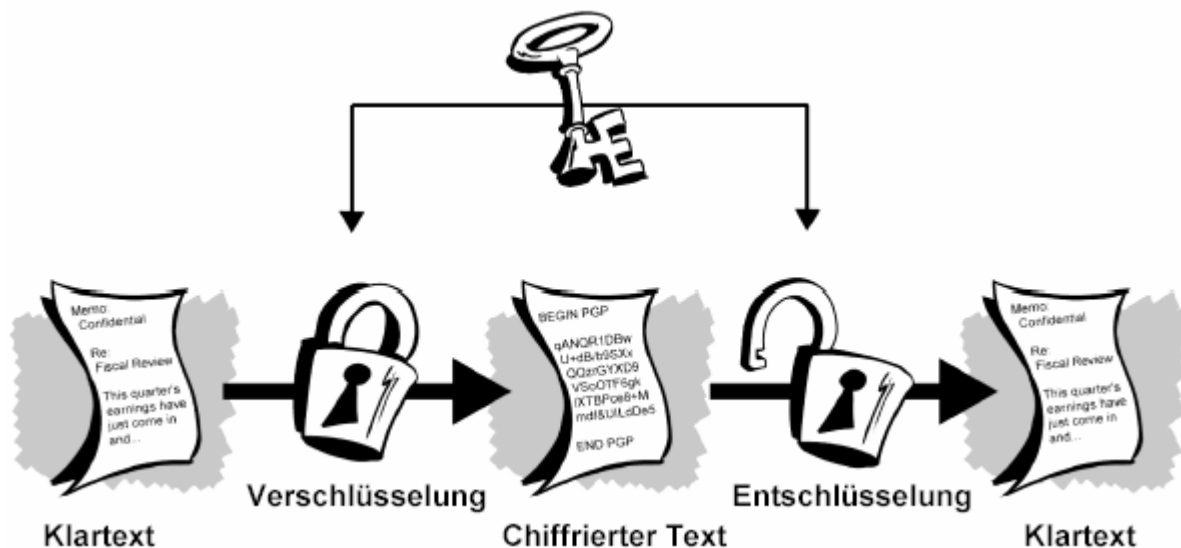


Abbildung 1 (Quelle : PGP, Einführung in die Kryptographie)

5.1 Vor- und Nachteile

Vorteile von symmetrischen Algorithmen sind die hohe Geschwindigkeit, mit denen Daten ver- und entschlüsselt werden. Symmetrische Verschlüsselungsverfahren sind leicht zu implementieren. Sie basieren auf der wiederholten Anwendung einfacher Methoden wie Permutation, Substitution und Transposition.

Weiterhin sind die Schlüssellängen deutlich kleiner, als bei asymmetrischen Verfahren. Schon mit kleinen Schlüssellängen kann eine hohe Sicherheit erreicht werden.

Ein Nachteil ist das Schlüsselmanagement. Um miteinander vertraulich kommunizieren zu können, müssen Sender und Empfänger vor Beginn der eigentlichen Kommunikation über einen sicheren Kanal einen Schlüssel ausgetauscht haben. Spontane Kommunikation zwischen Personen, die sich vorher noch nie begegnet sind, scheint so nahezu unmöglich. Soll in einem Netz mit n Teilnehmern jeder mit jedem zu jeder Zeit spontan kommunizieren können, so muss jeder Teilnehmer vorher mit jedem anderen der $n-1$ Teilnehmer einen

Schlüssel ausgetauscht haben. Insgesamt müssen also $n(n - 1)/2$ Schlüssel ausgetauscht werden. Einen Ausweg kann die Einrichtung eines zentralen Schlüsselservers darstellen. Allerdings sind damit wieder viele weitere Probleme verbunden. Für digitale Signaturen sind symmetrische Verfahren kaum verwendbar.

5.2 DES-Algorithmus

Der DES-Algorithmus wurde erstmals 1974 von der US Regierung veröffentlicht. DES basiert auf dem zu Anfang der 70er Jahre von IBM entwickeltem Verschlüsselungsverfahren *Lucifer*. Nach Änderungen (z.B. die Verkürzung des ursprünglich 128-Bit langen Schlüssels auf 64 Bit) durch die National Security Agency (NSA), wurde DES 1976 zum offiziellen Standard (ANSI X3.92-1981) für amerikanische Regierungsbehörden ernannt.

Vom mathematischen Standpunkt betrachtet ist der DES ein einfaches Verfahren, das symmetrisch mit der so genannten Produktverschlüsselung arbeitet, bei der als elementare Verschlüsselung Substitutionen und Transpositionen (Permutationen) verwendet werden. Aus den 64 Bit großen Blöcken des Klartextes erzeugt der Algorithmus 64 Bit große Chiffretexte. Der dazu verwendete Schlüssel ist ebenfalls 64 Bit lang, allerdings ist jedes achte Bit ein Paritätsbit, so dass nur 56 Bit zur Ver- und Entschlüsselung benutzt werden.

DES lässt sich sowohl hardwaremäßig, wie auch softwaremäßig implementieren. Bei den neuesten Hardware-Implementierungen liegen die Verschlüsselungsraten schon im Bereich GByte/s.

Software-Implementierungen arbeiten zwar deutlich langsamer, doch sind auch hier noch Verschlüsselungsraten von mehreren 100 kbps möglich.

5.2.1 Arbeitsweise von DES

In Abbildung 2 ist der DES-Algorithmus schematisch dargestellt.

Ein Klartext wird in 64-Bit lange Blöcke P_i aufgeteilt.

Im ersten Schritt wird auf P_i eine initiale Permutation IP angewandt. Dies ist eine für das Verfahren fest gewählte, vom Schlüssel unabhängige, Bitpermutation auf Bitvektoren der Länge 64.

Das Ergebnis wird dann in zwei 32-Bit große Blöcke L_0 und R_0 aufgeteilt. Daraufhin folgen 16 Runden, wobei für die i -Runde gilt:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i) \end{aligned}$$

Dabei ist K_i der Rundenschlüssel, der durch die Schlüsselexpansion aus dem geheimen Schlüssel abgeleitet wird.

Nach den 16 Runden werden L_{16} und R_{16} wieder vertauscht und zusammengefügt. Daraufhin wird die zu IP inverse Permutation IP^{-1} angewandt.

Für das Dechiffrieren wird der gleiche Algorithmus angewandt, wobei die Teilschlüssel K_i jetzt aber in umgekehrter Reihenfolge zum Einsatz kommen.

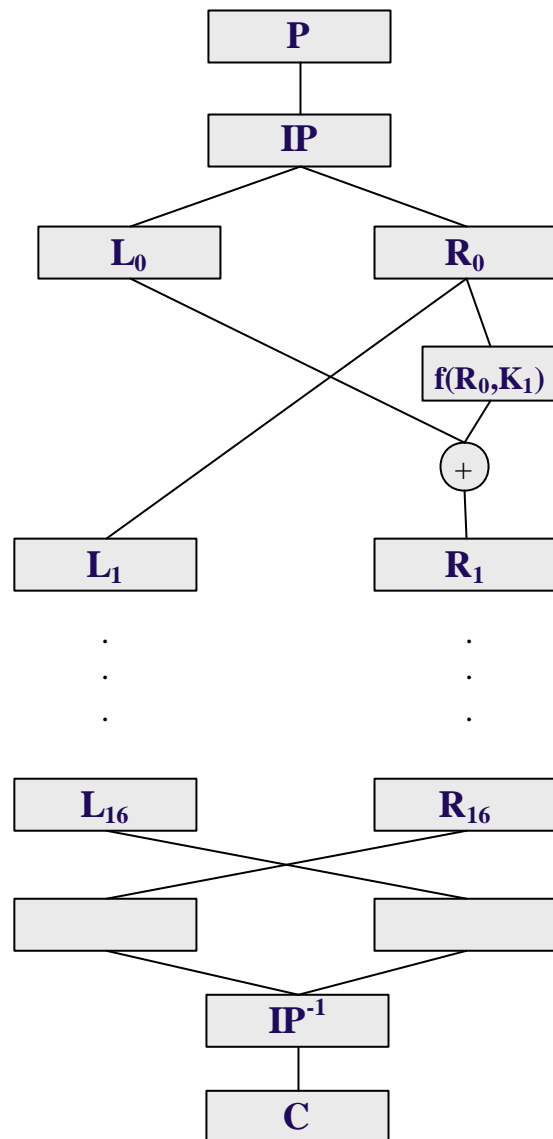
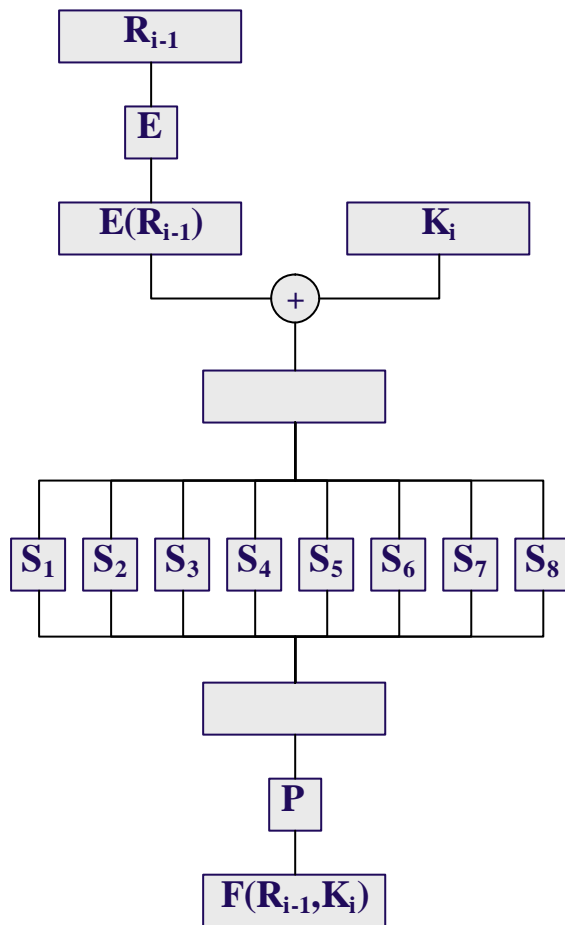


Abbildung 2

In Abbildung 3 ist die Rundenfunktion f schematisch dargestellt.



Die Rundenfunktion setzt sich aus elementaren Funktionen zusammen.

Zuerst wird der R-Block einer Expansion auf 48 Bit unterworfen und dann mit einem Rundenschlüssel mittels der XOR Operation verknüpft.

Der 48 Bit lange Block wird in acht 6 Bit lange Blöcke zerlegt.

Diese acht Werte dienen als Eingabe für acht S-Boxen.

Die S-Boxen S_1 bis S_8 ersetzen jeweils 6 Bit durch 4 Bit entsprechend dem folgenden Schema: das erste und das sechste Bit selektieren die Reihe der Tabelle, die mittleren 4 Bit selektieren die Spalte der Tabelle, als Ausgabe dient der entsprechende Tabelleneintrag.

Insgesamt entstehen acht 4 Bit lange Blöcke, die abschließend zusammengefasst werden und auf die eine Permutation angewandt wird.

Abbildung 3

5.2.2 Betriebsmodi von DES

Für den DES wurden 4 Betriebsmodi spezifiziert (FIPS Publication 81):

- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)

Diese Betriebsarten unterscheiden sich sehr stark in ihren Eigenschaften und in ihren Anwendungsbereichen.

ECB und CBC sind blockorientiert, CFB und OFB sind zeichenorientiert.

5.2.2.1 ECB

Wird der DES blockweise auf jeweils 64 Bit lange Klartextblöcke angewandt, dann betreibt man ihn im ECB-Modus.

Da jeder Klartextblock mit demselben Schlüssel verschlüsselt wird, gestattet es einem Angreifer viele Rückschlüsse auf den Klartext zu machen.

Ein weiterer Nachteil ist die Möglichkeit der Umordnung der einzelnen Blöcke, ohne dass der Empfänger eine Chance hat, dies zu bemerken. Denn es besteht keinerlei Abhängigkeiten zwischen den einzelnen verschlüsselten Blöcken.

5.2.2.2 CBC

Um diese Nachteile zu beseitigen, wurde CBC-Verfahren erfunden.

Das CBC-Verfahren arbeitet mit einer Verkettung, sodass eine Umordnung möglich, aber nicht unbemerkt bleibt.

Der aktuelle Klartextblock wird vor seiner Verschlüsselung mit dem chiffrierten Vorgängerblock mittels der XOR Operation verknüpft. Damit hängt das Ergebnis nicht mehr ausschließlich vom aktuellen Klartextblock, sondern von seinen Vorgängern ab.

Die Verschlüsselung ist also kontextabhängig. Gleiche Muster in unterschiedlichem Kontext werden verschieden verschlüsselt.

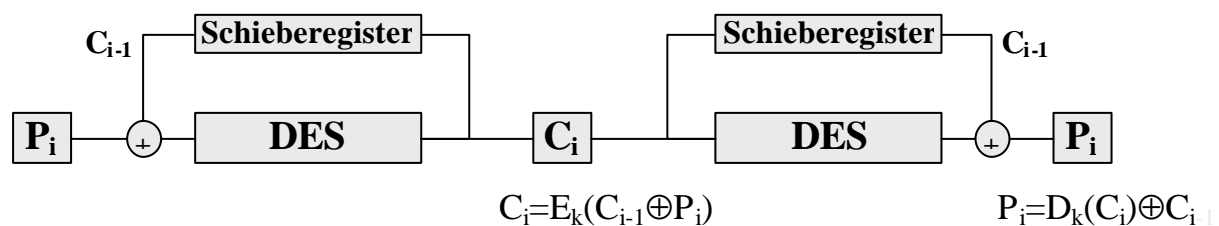


Abbildung 4

Beim CBC-Modus beeinflusst jede Bitmodifikation in einem chiffrierten Block den vollen Klartextblock und zusätzlich korrespondierende Bit in seinem Nachfolger. Auf weitere Blöcke hat diese Bitmodifikation dann aber keinen Einfluss mehr.

5.2.2.3 CFB

Der CBC-Verfahren ist zum Verschlüsseln langer Nachrichten gut geeignet. Es kann jedoch Effizienzprobleme geben, weil der Empfänger immer abwarten muss, bis der Sender einen ganzen Schlüsseltextblock erzeugt und diesen versandt hat, bevor er mit der Entschlüsselung des Blocks beginnen kann.

Das CFB-Verfahren bietet die Möglichkeit, Dateneinheiten zu verschlüsseln, deren Bit-Anzahl kleiner als die Blocklänge der verwendeten Chiffre ist.

Der DES wird als Pseudozufallszahlengenerator für die Erzeugung einer Schlüsselreihe eingesetzt.

Die eigentliche Verschlüsselung erfolgt dann zeichenweise mit Hilfe einer Addition Modulo 2: die ersten 8 Bit des Klartextes werden mit den linken 8 Bit der Schlüsselreihe verknüpft. Dadurch entsteht das Chiffre, welches an den Empfänger übertragen und in das Schieberegister geschoben wird. Das Schieberegister wird mit DES verschlüsselt und die linken acht Bit werden zur bitweisen Addition Modulo 2 mit den nächsten acht Bit des Klartextes herangezogen.

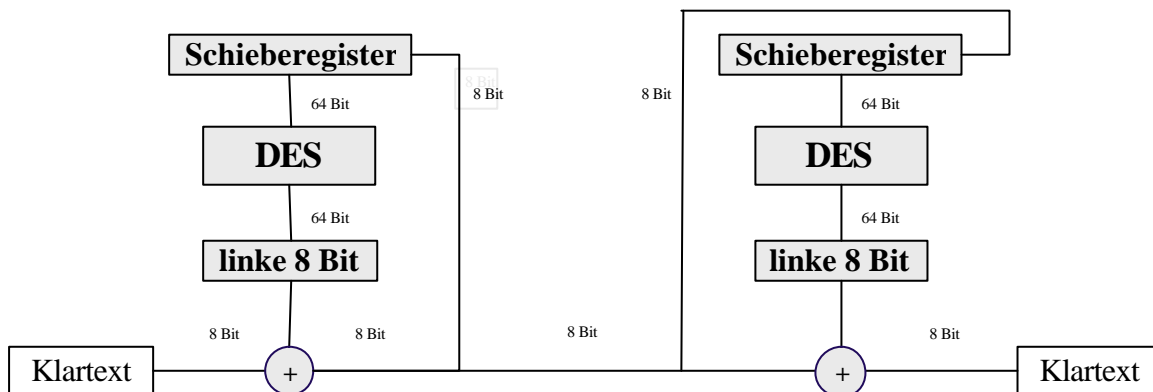


Abbildung 5

5.2.2.4 OFB

Das OFB-Verfahren ist dem CFB-Verfahren sehr ähnlich, die beiden Verfahren unterscheiden sich in der Art der Rückkopplung. Anstelle des Chiffretextzeichens werden die linken 8 Bit der Schlüsselreihe in das Schieberegister geschoben.

Diese Verfahren hat die gravierende Schwäche, dass ein Fehler im Schieberegister den gesamten verschlüsselten Text unbrauchbar macht.

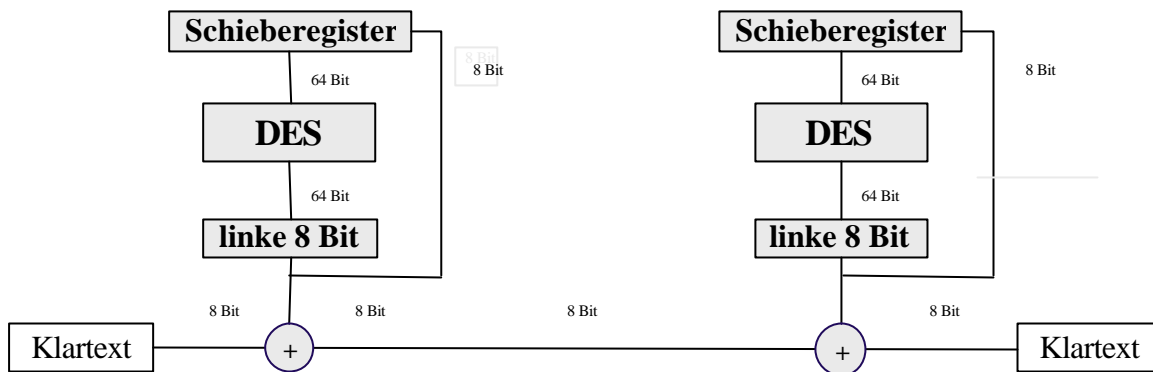


Abbildung 6

5.2.3 Triple DES (3DES)

Beim Triple DES wird das Problem der relativ kurzen Schlüssel durch Mehrfachverschlüsselung gelöst. Es gibt verschiedene Formen des Triple DES. Einige nutzen zwei Schlüssel, andere setzen drei verschiedene 56 Bit Schlüssel ein. Werden beispielsweise zwei Schlüssel benutzt, wird zunächst der Klartext mit dem ersten Schlüssel verschlüsselt, mit dem zweiten entschlüsselt und wieder mit dem ersten verschlüsselt.

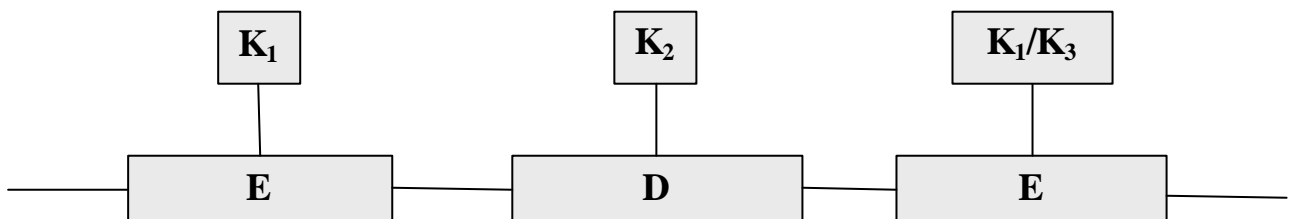


Abbildung 7

Das Verfahren gilt als sicher, ist aber nicht besonders performant.

5.2.4 Sicherheit und Schwächen von DES

Seit seiner Veröffentlichung wird der DES-Algorithmus sehr intensiv untersucht, da es sich beim DES um den ersten von der National Security Agency mitentwickelten Algorithmus handelte, der veröffentlicht wurde.

Vor allem die mysteriösen Konstanten der acht S-Boxen erweckten Misstrauen, da eine Hintertür in ihnen vermutet wurde. Verschiedene Versuche, die Konstanten durch andere Werte zu ersetzen, zeigten jedoch, dass die Original-Werte sehr gut gewählt sind; die Alternativvorschläge führten alle zu schwächeren Algorithmen. Besonders interessant ist in dieser Hinsicht, dass die Konstanten so gewählt wurden, dass die Chiffre einem Angriff mittels der so genannten differentiellen Kryptoanalyse bestmöglich widerstehen kann, was bedeutet, dass der NSA dieses Verfahren bereits 1974 bekannt war - die internationale Fachwelt „entdeckte“ die differentielle Analyse erst 1990.

Es sind nur drei denkbare Angriffsmöglichkeiten gegen DES bekannt:

1. Mit Hilfe von differentieller Kryptoanalyse kann ein Kryptoanalyst im Rahmen eines adaptiven „Chosen-plaintext“-Angriffs einen DES-Schlüssel aus 2^{47} Klar- und Chiffretextpaaren ermitteln.
2. Demgegenüber stellt die lineare Kryptoanalyse einen „Known-plaintext“-Angriff dar, bei dem 2^{43} Klar- und Chiffretextpaaren ausreichen, um einen DES-Schlüssel zu ermitteln.

Weil einem Kryptoanalyst 2^{47} oder 2^{43} Paare nur selten zur Verfügung stehen, sind die beiden Angriffsverfahren nur von theoretischem Interesse.

3. Der effektivste Angriff besteht aus dem Durchprobieren aller möglichen Schlüssel. Die kurze Schlüssellänge von 56 Bit bleibt der größte Kritikpunkt an DES. Da es nur 2^{56} DES-Schlüssel gibt, kann man diese alle durchprobieren. Es gilt auch zu beachten, dass in einem 2^{56} großen Schlüsselraum im Durchschnitt 2^{55} Schlüssel zu testen sind, um den richtigen zu finden.

Zudem ist DES invariant unter der binären Komplementbildung, so dass sich der Aufwand noch einmal halbiert.

Es gibt so genannte schwache und semi-schwache Schlüssel, bei denen zweimaliges Verschlüsseln mit demselben Schlüssel einer Entschlüsselung gleichkommt bzw. bei denen ein Schlüssel als Inverses eines anderen fungiert. Mit modernen Computern ist es heute möglich, DES-verschlüsselte Dokumente in wenigen Sekunden zu entschlüsseln.

DES kann heute nur noch als sicher gelten, wenn die in Unterkapitel 5.2.3 vorgestellte Dreifachverschlüsselung verwendet wird.

5.3 AES

Das US-amerikanische National Institute of Standards and Technology (NIST) hat Anfang 1997 einen Wettbewerb ausgeschrieben, um einen DES-Nachfolger zu finden.

Nach der Ankündigung des NIST im Januar 1997, wurden in zwei Workshops im April und Juni 1997 Anforderungen erarbeitet. Diese zu erfüllen, sollte für einen Algorithmus Voraussetzung sein, um als Kandidat für den AES in Frage zu kommen.

Als Minimalanforderungen wurden folgende Kriterien aufgestellt, die von den Algorithmen zu erfüllen sind:

- AES muss ein symmetrischer Blockalgorithmus sein
- AES muss frei von patentrechtlichen Ansprüchen sein
- AES muss mindestens 128 Bit lange Blöcke verwenden und Schlüssel von 128, 192 und 256 Bit Länge einsetzen können
- AES soll gleichermaßen leicht in Hard- und Software zu implementieren sein
- AES soll in Hard- wie Software eine überdurchschnittliche Performance haben
- Speziell für den Einsatz in Smartcards sollen geringe Ressourcen erforderlich sein
- AES soll allen bekannten Methoden der Kryptanalyse widerstehen können

Im Mai 2000 wurden die Analysen und öffentlichen Diskussionen abgeschlossen und schließlich am 2. Oktober 2000 der Sieger bekannt gegeben: Der belgische Algorithmus Rijndael wurde neuer Standard. Unter den 15 Kandidaten des Wettbewerbs um die DES-Nachfolge erschien er als geeignetster: außerordentlich schnell, einfach, Ressourcen sparend und hoffentlich jahrzehntelang sicher genug für alle denkbaren Anforderungen.

Rijndael ist ein symmetrischer Blockverschlüsselungsalgorithmus mit variabler Block- und Schlüssellänge: Er kann Blöcke zu 128, 192 und 256 Bit und ebensolche Schlüssellänge 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 in mehreren Runden behandelt. Die Anzahl der Runden ist von der Block- und Schlüssellänge abhängig:

r	b=128	b=192	b=256
k=128	10	12	14
k=192	12	12	14
k=256	14	14	14

Tabelle 1

5.3.1 Arbeitsweise von AES

Für die Ver- und Entschlüsselung muss für jede Runde ein Rundenschlüssel erzeugt werden. Hierzu wird der geheime Schlüssel des Anwenders expandiert, indem rekursiv abgeleitete 4-Byte-Wörter an diesen Anwenderschlüssel angehängt werden.

Der Algorithmus arbeitet auf einer Matrix von Bytes, dem so genannten State. Der Klartextblock wird dann spaltenweise abgebildet.

Hierauf wird der AES-Verschlüsselungsalgorithmus angewandt, der aus den folgenden Transformationen besteht:

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

Jede 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 Blockes werden in der so genannten ShiftRows-Transformation permutiert
3. **Diffusion**
Die Bytes der permutierten Matrix des Blockes werden durch die MixColumns-Transformation nochmals spaltenweise permutiert
4. **Verknüpfung mit dem Schlüssel**
Der jeweilige Rundenschlüssel wird mit Block durch XOR verknüpft

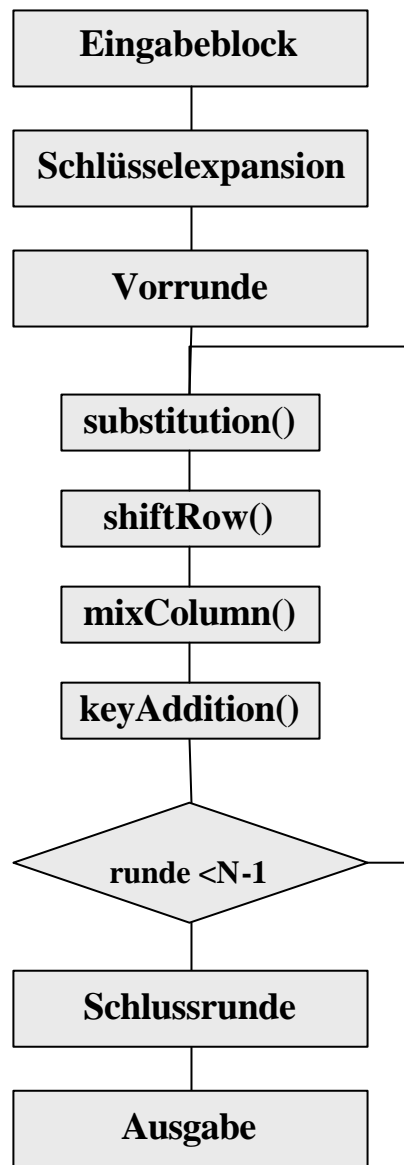


Abbildung 8

Einfluss des Schlüssel

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 stark nichtlineare Operation. Die Konstruktion der S-Box sorgt für nahezu idealen Schutz vor differentieller und linearer Kryptoanalyse.

Lineare Schicht

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

Entschlüsselung

Bei der Entschlüsselung von Daten wird genau rückwärts vorgegangen. Die Daten werden zunächst wieder in zweidimensionale Tabellen gelesen und die Rundenschlüssel generiert. Allerdings wird nun mit der Schlussrunde angefangen und alle Funktionen in jeder Runde in der umgekehrten Reihenfolge aufgerufen. Durch die vielen XOR-Verknüpfungen unterscheiden sich die meisten Funktionen zum Entschlüsseln nicht von denen zum Verschlüsseln. Jedoch muss eine andere S-Box genutzt werden (die sich aus der originalen S-Box berechnen lässt) und die Zeilenverschiebungen erfolgen in die andere Richtung.

5.3.2 Kryptographische Stärke von AES

Wie bei allen praktischen kryptografischen Verfahren, kann die Sicherheit jedoch nicht bewiesen werden. Man kann nur testen, ob die bisher bekannten Angriffsmethoden der Kryptanalyse versagen.

Beim Entwurf von Rijndael wurde auf alle bekannten Attacken, wie lineare und differenzielle Kryptoanalyse eingegangen, so dass ein Angriff mit allen herkömmlichen Verfahren nicht effizienter sein sollte als ein Brute-Force-Angriff. Mit den Schlüssellängen von 128 Bit ergibt sich ein Schlüsselraum von 3.4×10^{38} , mit 192 Bit 6.2×10^{57} und mit 256 Bit 1.1×10^{77} .

Wenn man nun annimmt, dass es eine Maschine gibt, die den ganzen DES Schlüsselraum in einer Sekunde durchsucht, dann würde es bei Rijndael ungefähr 149'000 Billionen Jahre dauern. Man bedenke, dass es das Universum seit weniger als 20 Billionen Jahren geben soll.

AES erwies sich als resistent gegen alle möglichen Angriffe.

Implementierungen von Rijndael können im Vergleich zu den anderen Kandidaten mit dem geringsten Aufwand gegen Angriffe geschützt werden, die auf Messungen von Änderungen der Stromaufnahme beruhen (s.g. Power Analysis-Attacken).

5.3.3 Kryptoanalyse von AES

Im Mai 2001 gelang es Ferguson, Schroepel und Whiting, den Algorithmus als geschlossene Formel darzustellen - allerdings mit 2^{50} Termen, also etwa eine Billion Summanden. Niemand weiß, ob daraus jemals ein sinnvoller Angriff auf AES konstruiert werden kann, doch bisher ließ sich kein anderes sicheres Verfahren in solch einer „einfachen“ Form darstellen.

Ein Qualitätssprung war jedoch eine Arbeit von Courtois und Pieprzyk. Die Mathematiker beschrieben ganze Klassen von Chiffrierungen mittels sehr großer Systeme quadratischer Gleichungen, zum Beispiel 128-Bit-AES als System von 8000 Gleichungen mit 1600 Variablen. Derartige Systeme lassen sich im Allgemeinen nicht mit vertretbarem Rechenaufwand lösen, doch in diesem Fall sind starke Vereinfachungen mittels der so genannten XSL-Methode möglich. Sie nutzt aus, dass die Gleichungssysteme mehr Gleichungen als Unbekannte enthalten (sie sind überbestimmt), die meisten Koeffizienten Null sind (schwach besetzt) und dass sie eine besonders reguläre Struktur haben. Dank der Weiterentwicklung der XSL-Methode erscheint mittlerweile ein Angriff auf AES mit „lediglich“ 2^{100} Rechenoperationen denkbar.

Diese Ergebnisse stellen einen Qualitätssprung in der Kryptoanalyse dar. Die bisherigen Methoden ließen Schlüssel ermitteln, allerdings nur mittels enormer Mengen von Geheim- und meist auch Klartext. Der XSL-Angriff hingegen könnte die Rekonstruktion des Schlüssels aus kleinen Mengen Geheimtext ermöglichen. Es überrascht auch, dass die Sicherheit eines Blockalgorithmus gegen diesen Angriff offenbar nicht mehr exponentiell mit der Rundenzahl steigt.

Aber trotzdem ist AES nicht unsicher. Viele Abschätzungen des Arbeitsaufwands basieren noch auf Vermutungen und 2^{100} Rechenschritte sind eine sehr große Zahl.

Die genannten Angriffe sind also rein akademischer Natur!

5.4 IDEA

1990 entwickelten Xuejia Lai und James L. Massey einen Algorithmus namens Proposed Encryption Standard (PES), der sich aber nach der kurz darauf entwickelten differentiellen Kryptoanalyse als unsicher erwies. Zusammen mit S. Murphy überarbeiteten Lai und Massey ihren Algorithmus und nannten ihn fortan IPES (Improved Proposed Encryption Standard). 1992 wurde dieser verbesserte Vorschlag eines Standards in die bekannte Bezeichnung IDEA umbenannt und von Ascom Tech patentiert. Für nicht kommerzielle Anwendungen darf der Algorithmus ohne Lizenz eingesetzt werden.

IDEA ist ein Verschlüsselungsalgorithmus, der DES ähnlich ist, mit einem 128-Bit langen Schlüssel arbeitet und trotz höherer Sicherheit viel schneller als DES ist. Die jeweils 64 Bit langen Datenblöcke durchlaufen eine 8 Runden langen Verschlüsselungsoperation (plus einer Ausgabetransformation), bei der IDEA sich auf Operationen, wie XOR, Addition modulo 2^{16} sowie Multiplikation modulo $2^{16}+1$ beschränkt und auf Permutationen verzichtet, um eine schnelle Implementierung in eine Software zu gewährleisten.

IDEA wäre nach dem heutigen Wissensstand der ideale Ersatz für DES. Zum einen sind die Operationen leicht sowohl in Hardware, als auch in Software zu implementieren. Zum anderen ist IDEA fast doppelt so schnell wie DES. Leider unterliegt IDEA dem Patentschutz, wodurch die Einsetzbarkeit eingeschränkt ist. Dennoch kann IDEA als verbreitetes Verfahren angesehen werden, da es bei PGP eingesetzt wird.

5.4.1 Arbeitsweise von IDEA

IDEA basiert auf drei einfachen, leicht zu implementierenden Operationen:

- XOR (\oplus)
- Addition modulo 2^{16} (+)
- Multiplikation modulo $2^{16}+1$ (\otimes)

Sowohl bei der Addition modulo 2^{16} als auch bei der Multiplikation modulo $2^{16}+1$ bleiben Überläufe unberücksichtigt.

In IDEA existieren acht Runden. Der Klartextblock wird in vier je 16 Bit breite Teilblöcke zerteilt. Diese werden in der Runde mit sechs 16-Bit-Teilschlüsseln, die für jede Runde aus dem 128-Bit-Schlüssel generiert werden, durch die oben genannten Operationen verknüpft. Jede Runde liefert vier 16-Bit-Blöcke als Ausgabe, die als Eingabe für die nächste Runde dienen. Zum Abschluss des Algorithmus werden die vier Blöcke (Resultat der letzten Runde) in einer Ausgabetransformation mit vier Teilschlüsseln kombiniert.

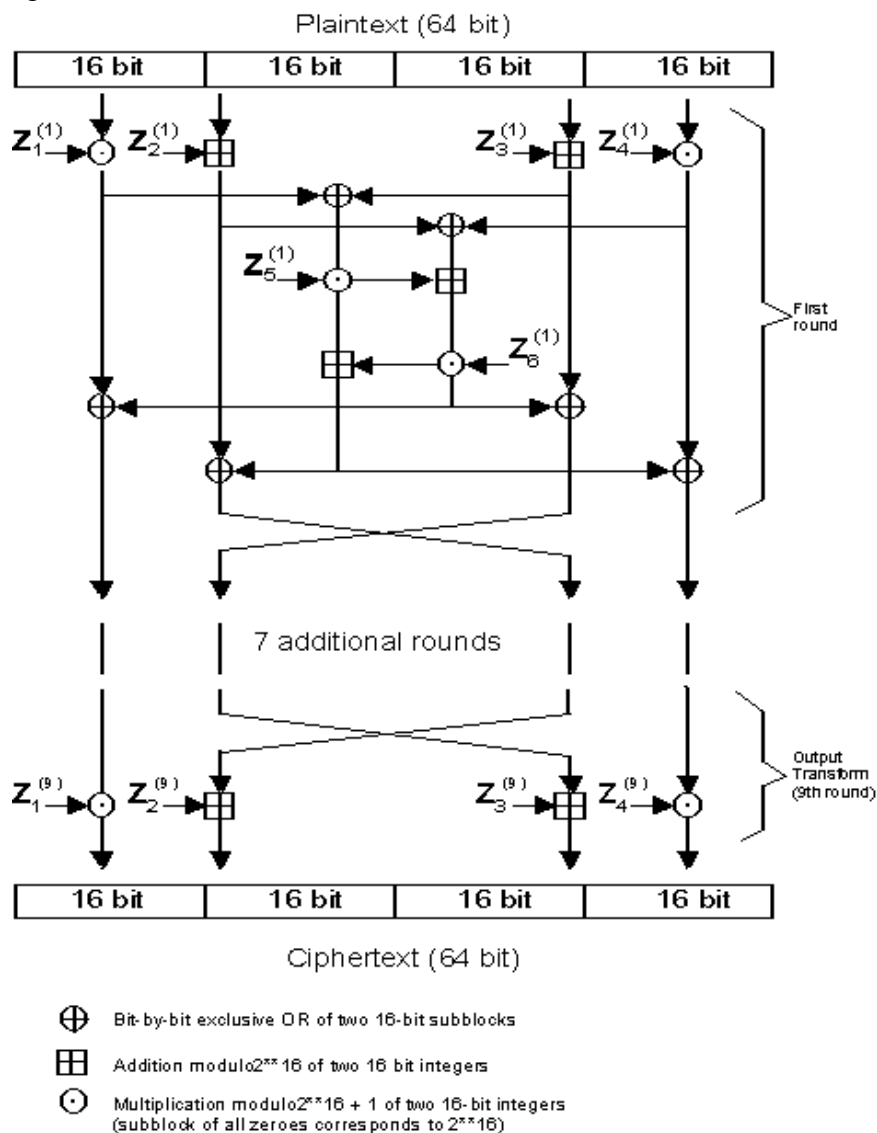


Abbildung 9 (Quelle: Ascom Tech)

Jede Runde des Algorithmus (Abbildung 9) besteht aus 14 Schritten:

1. Multiplikation des ersten Teilblock mit dem ersten Teilschlüssel
2. Addition des zweiten Teilblock mit dem zweiten Teilschlüssel
3. Addition des dritten Teilblock mit dem dritten Teilschlüssel
4. Multiplikation des vierten Teilblock mit dem vierten Teilschlüssel
5. XOR-Verknüpfung der Resultate der Schritte 1 und 2
6. XOR-Verknüpfung der Ergebnisse der Schritte 3 und 4
7. Multiplikation des Ergebnisses aus Schritt 5 mit dem fünften Teilschlüssel
8. Addition der Resultate der Schritte 6 und 7
9. Multiplikation des Resultats aus Schritt 8 mit dem sechsten Teilschlüssel
10. Addition der Ergebnisse der Schritte 7 und 9
11. XOR-Verknüpfung der Resultate der Schritte 1 und 9
12. XOR-Verknüpfung der Ergebnisse der Schritte 3 und 9
13. XOR-Verknüpfung der Resultate der Schritte 2 und 10
14. XOR-Verknüpfung der Ergebnisse der Schritte 4 und 10

Die vier Ausgabeblocke bilden die Ergebnisse der Schritte 11 bis 14, die die Eingabe für die nächste Runde darstellen.

Die Ausgabetransformation besteht aus 4 Schritten:

1. Multiplikation des ersten Teilblock mit dem ersten Teilschlüssel
2. Addition des zweiten Teilblock mit dem zweiten Teilschlüssel
3. Addition des dritten Teilblock mit dem dritten Teilschlüssel
4. Multiplikation des vierten Teilblock mit dem vierten Teilschlüssel

Die Ergebnisse der Ausgabetransformation, d. h. die resultierenden vier 16-Bit-Blöcke, werden zu dem 64-Bit-Chiffreblock zusammengefügt. Zur Entschlüsselung kommt der gleiche Algorithmus wie zur Verschlüsselung zum Einsatz. Der Unterschied besteht darin, dass die Teilschlüssel in umgekehrter Reihenfolge angewendet werden. Manche der Teilschlüssel werden durch ihre additiven - oder multiplikativen Inversen ersetzt.

5.4.2 Sicherheit von IDEA

Der IDEA hat sich als resistent gegenüber allen heute bekannten kryptanalytischen Angriffen erwiesen. Insbesondere ist gezeigt worden, dass bereits nach vier von acht Runden eine Differential-Kryptanalyse nicht mehr greifen kann.

Die Schlüssellänge von 128 Bit macht einen direkten Angriff fast unmöglich.

Ein Ansatz wären schwache Schlüssel, die auch für IDEA existieren. Diese sind jedoch nicht wie bei DES im Sinne des Algorithmus schwach, sondern relativ leicht aus einem "chosen-plaintext"-Angriff zu ermitteln. Die Chance, einen solchen schwachen Schlüssel zu erzeugen, ist relativ gering.

Bis heute gilt IDEA als einer der sichersten symmetrischen Algorithmen überhaupt.

6 Asymmetrische Kryptosysteme

In einem asymmetrischen Kryptosystem verwenden der Sender und der Empfänger zwei unterschiedliche Schlüssel, einen zum Verschlüsseln, den anderen zum Entschlüsseln einer Nachricht.

Mit einem *öffentlichen Schlüssel* werden Daten verschlüsselt, und mit dem zugehörigen *privaten* oder *geheimen Schlüssel* werden Daten entschlüsselt. Der öffentliche Schlüssel ist allen bekannt, der private Schlüssel dagegen bleibt geheim.

Rein rechnerisch ist es unmöglich, den privaten Schlüssel aus dem öffentlichen Schlüssel abzuleiten. Jeder mit einem öffentlichen Schlüssel kann Daten zwar verschlüsseln, aber nicht entschlüsseln. Nur die Person mit dem entsprechenden privaten Schlüssel kann die Daten entschlüsseln.

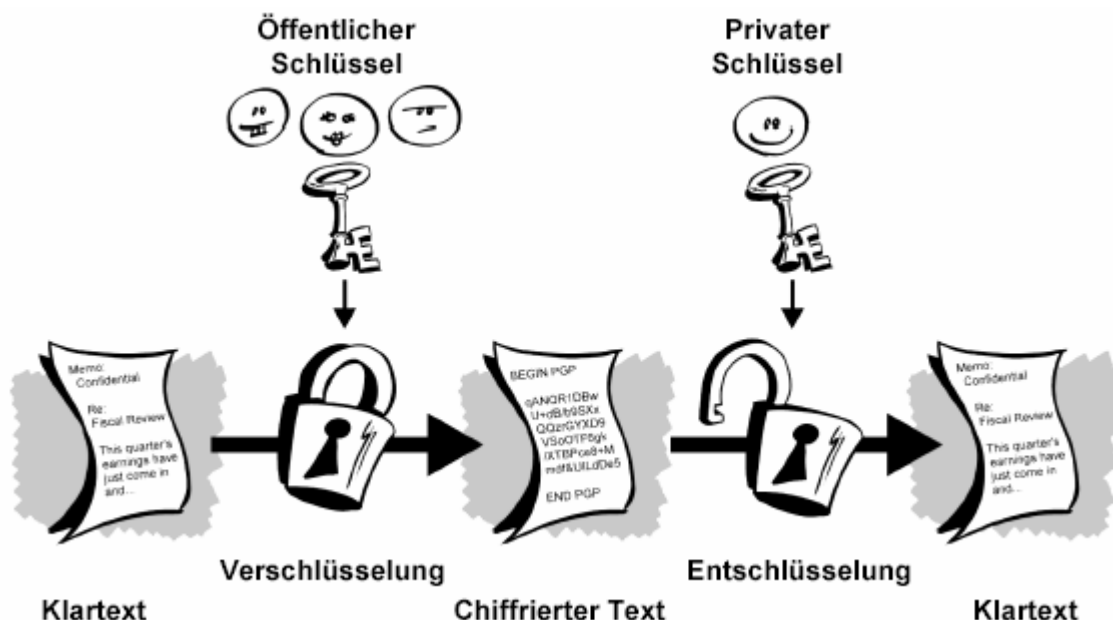


Abbildung 10 (Quelle : PGP, Einführung in die Kryptographie)

Will Bob verschlüsselte Nachrichten empfangen, so macht er den Verschlüsselungsschlüssel bekannt, hält aber den Entschlüsselungsschlüssel geheim. Jeder kann den öffentlichen Schlüssel benutzen, um geheime Nachrichten an Bob zu schicken.

6.1 Vor- und Nachteile

Das Übertragen von geheimen Schlüsseln zwischen Absender und Empfänger über einen sicheren Kanal ist nicht mehr notwendig. Für jede Kommunikation sind nur noch öffentliche Schlüssel erforderlich.

Ein weiterer wichtiger Vorteil von asymmetrischer Kryptographie ist die Möglichkeit der elektronischen Unterschrift.

Ein Nachteil der asymmetrischen Verfahren ist allerdings die Tatsache, dass ihre Anwendung deutlich langsamer als bei symmetrischen Verfahren ist.

Es ist notwendig, relativ lange Schlüssel zu verwenden, da bei kurzen Schlüsseln eine Rückrechnung der unvollkommenen Einwegfunktion durch einfaches "Durchprobieren" gerade mit leistungsstarken Rechnersystemen nicht ausgeschlossen werden kann. Durch die langen Schlüssel erhöht sich teilweise die Datenmenge enorm.

Schlüsselverwaltung bringt Komplikationen: öffentliche Schlüssel müssen authentisch sein. Die Sicherheit beruht "nur" auf der vermuteten, aber von der Fachwelt anerkannten, algorithmischen Schwierigkeit eines mathematischen Problems.

6.2 RSA

RSA ist ein asymmetrisches Kryptosystem, das 1977 von Ron Rivest, Adi Shamir und Len Adleman am Massachusetts Institute for Technology (MIT) entwickelt und 1978 publiziert worden ist. Das Akronym RSA setzt sich aus den Anfangsbuchstaben der Namen der drei Entwickler zusammen.

RSA basiert auf der Arithmetik in einem Restklassenring Modulo n . Die Sicherheit des Systems beruht auf der Annahme, dass das modulare Potenzieren mit einem festen Exponenten und einem festen Modul eine Falltür-Einwegfunktion darstellt, bzw. dass das Faktorisierungsproblem ein schwieriges Problem ist. So ist es relativ leicht, zwei große Primzahlen miteinander zu multiplizieren, ihr Produkt aber wieder in seine Faktoren zu zerlegen, ist eine aus mathematischer Sicht äußerst schwierige Aufgabe.

6.2.1 Schlüsselerzeugung

Die Schlüsselerzeugung funktioniert folgendermaßen:

Man wählt nacheinander zufällig zwei große Primzahlen p und q und berechnet das Produkt:

$$n = p \cdot q$$

Als öffentlichen Schlüssel wählt man dann eine Zahl e , die relativ prim zu $j(n) = (p-1)(q-1)$ ist, d.h. eine Zahl e , die mit $j(n)$ den größten gemeinsamen Teiler 1 hat.

Ist e gewählt, dann muss der Benutzer noch einen privaten Schlüssel d als Inverse von e modulo $j(n)$ ermitteln, d.h. d hat die Gleichung $e \cdot d = 1 \pmod{j(n)}$ zu erfüllen.

Da der größte gemeinsame Teiler von e und $j(n)$ gleich 1 ist, gibt es eine solche Zahl d tatsächlich. Sie kann mit dem erweiterten euklidischen Algorithmus berechnet werden. Der öffentliche Schlüssel besteht aus dem Paar (n, e) . Der private Schlüssel ist (n, d) . Die Zahl n heißt RSA-Modul, e heißt Verschlüsselungsexponent und d heißt Entschlüsselungsexponent.

6.2.2 Verschlüsselung

Ein Klartext m wird verschlüsselt zu $c = m^e \bmod n$. Jeder, der den öffentlichen Schlüssel (n, e) kennt, kann die Verschlüsselung durchführen. Bei der Berechnung von $m^e \bmod n$ wird schnelle Exponentiation verwendet.

6.2.3 Entschlüsselung

Die Entschlüsselung von RSA beruht auf folgendem Satz.

Sei (n, e) ein öffentlicher und (n, d) der entsprechende private Schlüssel im RSA-Verfahren. Dann gilt

$$(m^e)^d \bmod n = m$$

für jede natürliche Zahl m mit $0 \leq m < n$.

Wurde als c wie in (6.2.2) berechnet, kann man m mittels

$$M = c^d \bmod n$$

rekonstruieren.

6.2.4 Kryptoanalyse

Man nimmt heute an, dass es keine einfachere Möglichkeit gibt, RSA zu brechen, als den Modul n zu faktorisieren. Ein Beweis dieser Annahme ist allerdings noch ausstehend.

Das Faktorisierungsproblem wird in der Zahlentheorie schon seit Jahrhunderten diskutiert. Es gilt als sehr schwierig. Sollte es sich wider Erwarten als einfach herausstellen, so kann man davon ausgehen, dass dies an mehreren Orten gleichzeitig gefunden und daher schnell bekannt würde. Niemand könnte aus der Unsicherheit von RSA dann für längere Zeit einen Vorteil ziehen.

Die wichtigsten Faktorisierungsalgorithmen sind heute wohl die folgenden:

- Elliptic Curve Methode

- Quadratic Sieve
- Number Field Sieve

Die Algorithmen arbeiten mit einer Zeitkomplexität in der Größenordnung von

$$T(n) = e^{\log(n)\log(\log(n))/2}$$

Ende 2001 veröffentlichte der Mathematik-Professor Daniel Bernstein eine Arbeit, in der er einen neuartigen Parallelrechner zur Faktorisierung großer Zahlen beschreibt. Mit dieser Spezialhardware soll es möglich sein, den Rechenaufwand einiger Teilschritte bei der Faktorisierung mit dem *Number Field Sieve* erheblich zu reduzieren. Nach Angaben des Autors kann das neuartige Verfahren bei gleichen Rechenkosten (Produkt aus Rechenzeit und Speicherbedarf) dreimal so lange Zahlen faktorisieren wie bisherige Algorithmen. Allerdings ist der Faktor drei asymptotisch zu sehen - vereinfacht ausgedrückt, ist es keineswegs klar, ob unter realistischen Annahmen mit Bernsteins Ideen für derzeit gängige Schlüssellängen (1024 bis 4096 Bit) eine deutliche Beschleunigung zu erzielen ist. Bernstein hat seine Arbeit bisher nur in Form eines Forschungsantrags veröffentlicht, in dem er die Simulation seiner Maschine und die Ermittlung präziserer Ergebnisse ankündigt.

6.2.5 Wahl von p und q

Um die Faktorisierung des RSA-Moduls möglichst schwer zu machen, werden seine Primfaktoren p und q etwa gleich groß gewählt. Manchmal wird verlangt, dass p und q so gewählt werden, dass bekannte Faktorisierungsalgorithmen kein leichtes Spiel haben. Es ist aber schwer, das vorherzusagen.

Für die beiden Primfaktoren soll noch zusätzlich gelten:

$$e_1 < |\log_2(p) - \log_2(q)| < e_2, e_1 \approx 0.5, e_2 \approx 30$$

Nach heutiger Kenntnis sollte der RSA-Modul n wenigstens 512 Bits lang sein. Aus experimentellen Untersuchungen des Faktorisierungsproblems ergibt sich, dass bei längerfristiger Sicherheit der RSA-Modul n 768, 1024 oder 2048 Bits lang sein sollte.

Bis Ende 2005	Bis Ende 2006	Bis Ende 2007
1024	1024 (Mindestwert) 2048 (Empfehlung)	1536 (Mindestwert) 2048 (Empfehlung)

Tabelle 2

Solche Empfehlungen sind aber mit Vorsicht zu genießen, weil niemand den Fortschritt im Bereich der Algorithmen prognostizieren kann.

7 Hybride Kryptosysteme

Hybride Verschlüsselungsverfahren kombinieren symmetrische und asymmetrische Verschlüsselungsverfahren, um die jeweiligen Vorteile zu nutzen.

Da symmetrische Verfahren bei vergleichbarem Aufwand eine höhere kryptographische Sicherheit bieten, wird das Dokument selbst mit einem symmetrischen Sitzungsschlüssel verschlüsselt.

Dieser Sitzungsschlüssel wird dann mit Hilfe des asymmetrischen Verfahrens verschlüsselt und zusammen mit der Nachricht an den Empfänger übertragen. Der Empfänger kann den Sitzungsschlüssel mit Hilfe seines geheimen Schlüssels bestimmen und mit diesem dann die Nachricht entschlüsseln. Auf diese Weise nutzt man das bequeme Schlüsselmanagement asymmetrischer Verfahren und kann trotzdem große Datenmengen schnell und effektiv mit symmetrischen Verfahren verschlüsseln.

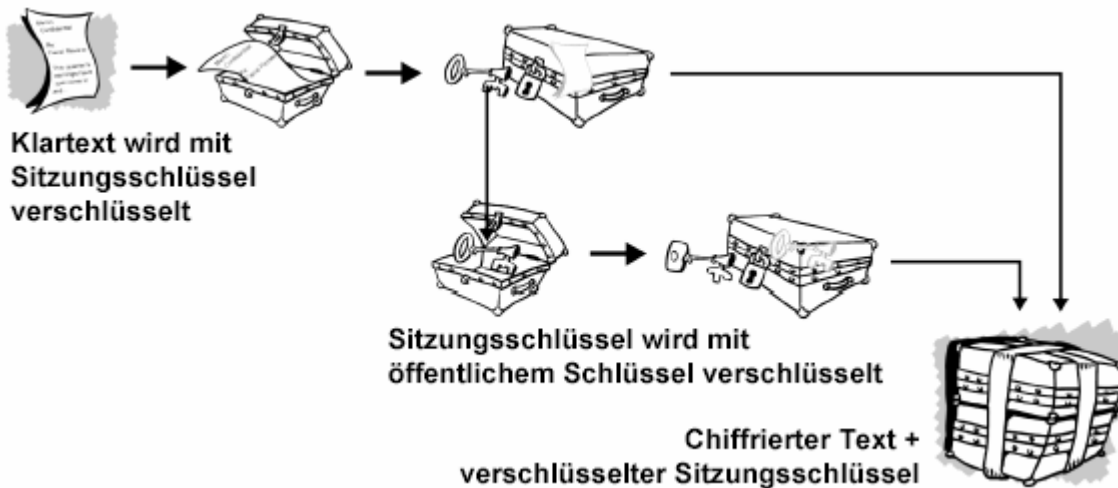


Abbildung 11 (Quelle : PGP, Einführung in die Kryptographie)

8 Hashfunktionen

In diesem Kapitel werden kryptographische Hashfunktionen behandelt.

8.1 Grundbegriffe

Unter einer *Hashfunktion* versteht man eine Abbildung

$$h: \Sigma^* \rightarrow \Sigma^n, n \in \mathbb{N}.$$

Hashfunktionen bilden also beliebig lange Strings aus Strings fester Länge ab. Sie sind nie injektiv.

Hashfunktionen können mit Hilfe von *Kompressionsfunktionen* generiert werden. Eine *Kompressionsfunktion* ist eine Abbildung

$$h: \Sigma^m \rightarrow \Sigma^n, m, n \in \mathbb{N}, m > n.$$

Sie bildet Strings einer festen Länge auf Strings fester kürzerer Länge ab.

Kryptographische Hash- und Kompressionsfunktionen müssen Eigenschaften haben, die ihre sichere Verwendbarkeit garantieren. Diese Eigenschaften werden jetzt beschrieben. Bezeichnen wir den Definitionsbereich von h mit D . Um h in der Kryptographie verwenden zu können, verlangt man, dass Wert $h(x)$ für alle $x \in D$ effizient berechenbar ist.

Die Funktion heißt *Einwegfunktion*, wenn es praktisch unmöglich ist, zu einem $s \in \Sigma^n$ ein $x \in D$ mit $h(x)=s$ zu finden. Es ist nicht bekannt, ob es Einwegfunktionen gibt. Es gibt aber Funktionen, die nach heutiger Kenntnis Einwegfunktionen sind. Ihre Funktionswerte sind leicht zu berechnen, aber es ist kein Algorithmus bekannt, der die Funktion schnell umkehren kann.

Eine *Kollision* von h ist ein Paar $(x, x') \in D^2$ von Strings, für die $x \neq x'$ und $h(x)=h(x')$ gilt. Alle Hashfunktionen und Kompressionsfunktionen besitzen Kollisionen, weil sie nicht injektiv sind.

Die Funktion h heißt *schwach kollisionsresistent*, wenn es praktisch unmöglich ist, für ein vorgegebenes $x \in D$ eine Kollision (x, x') zu finden. Die Funktion h heißt *stark kollisionsresistent*, wenn es praktisch unmöglich ist, irgendeine Kollision (x, x') von h zu finden.

Kollisionsresistente Hashfunktionen erlauben die Überprüfung der Integrität eines Textes, Programms, also der Übereinstimmung mit dem Original. Mit der Hashfunktion kann die Integrität der Originaldaten auf die Integrität eines viel kleineren Hashwertes zurückgeführt werden. In manchen Anwendungen muss man Hashfunktionen benutzen, die stark

kollisionsresistent sind. Ein wichtiges Beispiel sind elektronische Signaturen, die im Kapitel 9.1 beschrieben werden.

8.2 Sicherheitsrisiken - Geburtstags-Attacke

Im Wesentlichen gibt es zwei schwerwiegende Attacken gegen Einweghashfunktionen. Zum einen könnte man ein anderes Dokument herstellen, das denselben Hashwert besitzt wie das Original-Dokument; damit könnte man im Beispiel der Signatur "nachweisen", dass jemand Dokument B unterschrieben hat, obwohl ihm tatsächlich Dokument A zur Unterschrift vorlag.

Diese zweite Attacke wird auch *Geburtstags-Attacke* genannt. Dem liegt das so genannte "Geburtstagsparadoxon" zugrunde: Bei einer Geburtstagsparty müssten in einem Raum 183 Personen sein, damit mit einer Wahrscheinlichkeit größer 50 % eine Person anwesend ist, die den gleichen Geburtstag hat wie der Gastgeber (erste Art von Attacken gegen Einweghashfunktionen). Es müssen aber nur 23 sein, um mit einer Wahrscheinlichkeit größer 50 % zwei Personen mit demselben (aber beliebigen) Geburtstag zu finden.

Der *Geburtstagsangriff* arbeitet folgendermaßen: Um ein gefälschtes Dokument zu signieren, lassen sich zwei Dokumente D und F erzeugen, von denen eines einen harmlosen, das andere einen entsprechend veränderten Text enthält. Beide Dokumente werden durch unwesentliche Änderungen (Leerzeichen, Backspaces, Tabs usw.) variiert, und deren Hashwerte verglichen. Sobald zwei Dokumente mit gleichen Hashwerten gefunden werden, kann man das eine Dokument vorlegen, den Hashwert signieren lassen und später mit Dokument F einsetzen.

Angenommen, man arbeitet mit 64-Bit-Hashwerten, so würde es bei einer Rechenleistung von 1 Mio. Hashwerten pro Sekunde rund 600.000 Jahre dauern, bis ein zweites Dokument mit dem gleichen Hashwert wie ein vorgegebenes Dokument gefunden wäre. Um ein Paar von Dokumenten mit übereinstimmendem Hashwert zu finden, benötigte derselbe Rechner aber nur rund eine Stunde.

Um die Hashfunktion sicher gegenüber der Geburtstagsattacke zu machen, sollte man mit 160-Bit-Hashwerten arbeiten.

8.3 MD5

MD5 ist eine Einweg-Hashfunktion, die von Ron Rivest entworfen wurde. Es ist die verbesserte Version von MD4. MD steht für Message Digest, was soviel wie Zusammenfassung einer Nachricht bedeutet.

Es gibt einen Angriff, der Kollisionen produziert. Obwohl dieser die Sicherheit des Algorithmus im praktischen Einsatz nicht beeinträchtigt, wird allgemein hin von der Nutzung MD5s eher abgeraten - man bevorzugt stattdessen SHA-1, der mit 160 bit Hashwerten arbeitet.

8.3.1 Arbeitsweise

MD5 arbeitet in 4 Runden und mit 64 Konstanten. Das Ergebnis ist ein 128-Bit-Hashwert. Sein Ablauf lässt sich in vier Schritten darstellen:

Schritt 1: Padding der Nachricht

In einem ersten Schritt muss die Nachricht auf ein Vielfaches von 512 Bit ergänzt werden, da die Hauptiterationsfunktion mit 512-Bit-Blöcken arbeitet. Dazu wird sie auf eine Länge kongruent 448 modulo 512 gebracht, indem an die Nachricht zuerst eine einzelne 1 angehängt wird und dann noch so viele Nullen, bis die Nachricht kongruent 448 modulo 512 ist. (Das Anhängen der 1 stellt sicher, dass die Nachricht mindestens eine Eins enthält.) Der schlimmste Fall ist eine 448 Bit lange Nachricht; sie wird hierbei durch das Padden auf 960 Bit erweitert, da sie nach Anhängen der Eins eine Länge von 449 Bit besitzt, weshalb sie dann mit 511 Nullen ergänzt wird. In die letzten 64 freien Bit wird die Länge der ursprünglichen Nachricht geschrieben. Das Resultat, deren Länge ein Vielfaches von 512 ist, wird zu einem Block aus 16 32-Bit Worten. Es sei W_i das i -Wort eines solchen Blocks.

Schritt 2: Initialisierung

Bei der Berechnung werden vier 32-Bit-Puffer verwendet, die den aktuellen Rundenhashwert mit einer Größe von 128 Bit beinhalten. Diese müssen mit folgenden Werten initialisiert werden:

$a := 76543210hex$
 $b := fedcba98hex$
 $c := 98abcdefhex$
 $d := fedcba89hex$

Schritt 3: Berechnung des Blockhashwertes

In diesem Schritt definieren wir:

64 Funktionen $f_i(x,y,z)$:
 $f_i(x,y,z) = (x \cap y) \cup (\neg x \cap z), \quad 0 \leq i \leq 15$
 $f_i(x,y,z) = (x \cap y) \cup (x \cap z) \cup (y \cap z), \quad 16 \leq i \leq 31$
 $f_i(x,y,z) = x \oplus y \oplus z, \quad 32 \leq i \leq 47$
 $f_i(x,y,z) = y \oplus (\neg z \cup x), \quad 48 \leq i \leq 63$

64 Konstanten: $K_i := 4294967296 * \text{abs}(\sin(i))$

16 Rotationskonstanten r_k : geben die Anzahl der Linksshifts an

Die Berechnung des Hashwertes nach MD5 sieht algorithmisch so aus:

```
A := a
B := b
C := c
D := d
FOR i = 0 TO 63 DO
  BEGIN
    T := B + R((A + fi(B,C,D) + Wi + Ki), rk)
    B := A
    A := D
    D := C
    C := T
  END
a := a + A
b := b + B
c := c + C
d := d + D
```

Schritt 4: Bestimmung des Hashwertes

Es wird nun solange Schritt 3 mit dem nächsten Block iteriert, bis alle Nachrichtenblöcke bearbeitet worden sind.

Die Ausgabe des MD5 ist dann die Konkatenation der a, b, c und d: $MD5(W) = a || b || c || d$.

Eigenschaften von MD5

Bei jeder Operation wird eine unterschiedliche nichtlineare Funktion verwendet, deren einzelne Ergebnisbits unabhängig und gleichmäßig verteilt sind. Im Vergleich zu MD4 wurde der Hauptschleife eine vierte Runde hinzugefügt und jeder Schritt enthält eine eindeutige Konstante. Da jeder Schritt vom Ergebnis des vorherigen abhängt, entsteht ein schnellerer Lawineneffekt (kleine Veränderungen der Nachricht machen signifikanten Unterschied im Hashtext aus).

8.3.2 Beispiel einer Kollision

MD5 wurde als weitgehend sicher betrachtet und ist sehr verbreitet, bis 1996 ein deutscher Kryptograph eine Schwäche in der Kompressionsfunktion gefunden hat. Dem Kryptographen Dobbertin gelang es, Kollisionen für die MD5 aufzufinden, das heißt einen Anfangswert k und zwei unterschiedliche 512-Bit-Blöcke B und B' , für die gilt: $f(k, B) = f(k, B')$.

Analyse der MD5-Kompressionsfunktion von Hans Dobbertin:

--

Collision for the compress function of MD5.

Use the initial value $IV = 0x12AC2375\ 0x3B341042\ 0x5F62B97C\ 0x4BA763ED$;
and define the first input $X = (X_i)_{i<16}$ by setting:

$X_0 = 0xAA1DDA5E$	$X_4 = 0x1006363E$	$X_8 = 0x98A1FB19$	$X_{12} = 0x1326ED65$
$X_1 = 0xD97ABFF5$	$X_5 = 0x7218209D$	$X_9 = 0x1FAE44B0$	$X_{13} = 0xD93E0972$
$X_2 = 0x55F0E1C1$	$X_6 = 0xE01C135D$	$X_{10} = 0x236BB992$	$X_{14} = 0xD458C868$
$X_3 = 0x32774244$	$X_7 = 0x9DA64D0E$	$X_{11} = 0x6B7A669B$	$X_{15} = 0x6B72746A$

The second input $X' = (X'_i)_{i<16}$ is defined by setting $X'_i = X_i$ ($i < 16$; $i \neq 14$)
and $X'_{14} = X_{14} + 2^9$.

Then we have a collision, i.e. $MD5\text{-compress}(IV; X) = MD5\text{-compress}(IV; X')$,
and this common compress value is

0xBF90E670 0x752AF92B 0x9CE4E3E1 0xB12CF8DE:

The computation of such a collision takes about 10 hours on a Pentium PC.

--

Damit ist zwar eine wichtige Eigenschaft nicht mehr gegeben, aber MD5 ist trotzdem der weitverbreitetste Hash-Algorithmus.

8.4 SHA-1

Der Secure Hash Algorithm (SHA) wurde 1992 vom National Institute for Standards and Technology in Zusammenarbeit mit der National Security Agency entwickelt. Wegen eines Sicherheitsloches in SHA wurde 1994 SHA-1 entwickelt und vorgestellt.

Im Gegensatz zum MD5, verarbeitet der SHA nur Nachrichten von 2^{64} Bits maximaler Länge. SHA produziert einen 160 bit langen Hashwert, also einen längeren als der von MD5. Ebenfalls unterteilt dieser Algorithmus die Nachricht in 512-Bit-Nachrichtenblöcke zur weiteren Verarbeitung.

8.4.1 Arbeitsweise

Die folgenden Parameter werden im SHA verwendet:

- a, b, c, ... , h: Arbeitsvariablen
- $H^{(i)}$: Der Hashwert für i Nachrichtenblöcke
- $H_j^{(i)}$: Das j-te Wort des $H^{(i)}$
- K_t : Konstante in der t-ten Iteration
- l: Die Länge der Nachricht
- M: Die Nachricht
- $M^{(i)}$: Der i-te Nachrichtenblock

Die verwendeten Funktionen sind:

- \cap : Bitweise AND Operation
- \cup : Bitweise OR Operation
- \oplus : Bitweises XOR
- \neg : Komplement
- +: Addition
- ROTL^n : Rotation nach links um n Stellen

$$f_t(x, y, z) = \begin{cases} (x \wedge y) \vee (\neg x \wedge z) & 0 \leq t \leq 19 \\ x \oplus y \oplus z & 20 \leq t \leq 39 \\ (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) & 40 \leq t \leq 59 \\ x \oplus y \oplus z & 60 \leq t \leq 79. \end{cases}$$

Weiter werden für den Algorithmus noch 4 Konstanten verwendet:

$$K_t = \begin{cases} 5a827999 & 0 \leq t \leq 19 \\ 6ed9eba1 & 20 \leq t \leq 39 \\ 8f1bbcdc & 40 \leq t \leq 59 \\ ca62c1d6 & 60 \leq t \leq 79. \end{cases}$$

Funktionsweise des Algorithmus:

Schritt 1: Padding der Nachricht:

Die Nachricht wird gepadded, so dass sie ein Vielfaches von 512 bytes lang ist. Das Padding funktioniert gleich wie bei MD5:

- eine 1 wird angefügt
- es werden so viele 0 angefügt, dass die Nachricht um 64 bit kürzer ist als ein Vielfaches von 512 bit
- in den verbleibenden 64 bit wird die Länge der Nachricht vor dem Padding hinterlegt.

Schritt 2: Initialisierung

Fünf 32 bit Variablen werden mit folgenden Werten initialisiert:

$$\begin{aligned} H_0^{(0)} &= 67452301 \\ H_1^{(0)} &= efcdab89 \\ H_2^{(0)} &= 98badcfe \\ H_3^{(0)} &= 10325476 \\ H_4^{(0)} &= c3d2e1f0. \end{aligned}$$

Schritt 4: Bestimmung des Hashwertes

Die Nachricht wird in Blöcken zu 512 bit abgearbeitet (in einer Schleife). Diese Schleife hat 4 Runden zu je 20 Operationen. Jede Operation wendet eine nichtlineare Funktion auf 3 der 5 Speichervariablen a - e an und shifted und addiert ähnlich MD5.

Die Berechnung des Hashwertes sieht algorithmisch so aus:

For i=1 to N

{

1. Der Nachrichtenblock wird von 16 32bit Worten in 80 32bit Worte nach folgendem Algorithmus umgewandelt:

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \text{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \leq t \leq 79 \end{cases}$$

2. Dann werden die Variablen kopiert:

$$a = H_0^{(i)}$$

$$b = H_1^{(i)}$$

$$c = H_2^{(i)}$$

$$d = H_3^{(i)}$$

$$e = H_4^{(i)}$$

3. For $t = 0$ to 79:

{

$$T = \text{ROTL}^5(a) + f_t(b, c, d) + e + K_t + W_t$$

$$e = d$$

$$d = c$$

$$c = \text{ROTL}^{30}(b)$$

$$b = a$$

$$a = T$$

}

4. Berechnung des $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)}$$

$$H_1^{(i)} = b + H_1^{(i-1)}$$

$$H_2^{(i)} = c + H_2^{(i-1)}$$

$$H_3^{(i)} = d + H_3^{(i-1)}$$

$$H_4^{(i)} = e + H_4^{(i-1)}$$

}

Die Ausgabe des SHA-1 ist dann die Konkatenation:

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)}.$$

8.4.2 Sicherheit von SHA

Mit seinem 160 Bit langen Hashwert ist SHA widerstandsfähiger gegen Brute-Force Angriffe als MD5. Ein Design-Fehler im 1992 veröffentlichten Algorithmus wurde im heute gebräuchlichen SHA-1 korrigiert, für den bisher keine wirkungsvollen kryptografischen Angriffe bekannt geworden sind. Durch den Einsatz einer fünften Variablen ist SHA-1 auch im Vergleich zu MD5 resistenter gegen Kollisionen geworden.

Angriffe wie die von Dobbertin gegen MD4 und MD5 sind im Moment nicht bekannt.

9 Kryptographische Anwendungen

Kryptographische Verfahren können in einer Vielzahl von Anwendungen zum Einsatz kommen. In diesem Kapitel werden digitale Signaturen und digitales Geld vorgestellt.

9.1 Digitale Signatur

Elektronische Dokumente will man nicht nur verschlüsseln, sondern auch signieren können. Das macht man mit digitalen Unterschriften. Diese Unterschriften haben eine ähnliche Funktion wie gewöhnliche Unterschriften. Digitale Signaturen müssen aber anders realisiert werden als handschriftliche. Man kann nicht einfach eine Unterschrift unter das elektronische Dokument setzen, weil diese Unterschrift leicht kopiert werden kann.

Die Idee, die der Realisierung digitaler Signaturen zugrunde liegt, ist eng mit der Public-Key-Verschlüsselung verwandt. Will Alice elektronisch signieren, braucht sie dazu einen privaten, geheimen Schlüssel d und einen öffentlichen Schlüssel e . Der private Schlüssel ist sicher gespeichert, der öffentliche Schlüssel liegt in einem Verzeichnis, zu dem jeder Zugang hat. Wenn Alice ein Dokument m signieren will, berechnet sie aus dem Dokument und ihrem privaten Schlüssel d digitale Signatur $s(d, m)$ des Dokumentes. Unter Verwendung des öffentlichen Schlüssels kann dann jeder verifizieren, dass die Signatur korrekt ist.

Die folgenden Anforderungen an eine digitale Unterschrift werden meist gestellt:

- Authentizität: Nur eine Person darf in der Lage sein, eine digitale Unterschrift zu erzeugen.
- Verifizierbarkeit: Alle Personen können die Echtheit einer digitalen Signatur zu prüfen.
- Verbindlichkeit: Ein Unterzeichner darf später nicht abstreiten können, eine digitale Unterschrift geleistet und erzeugt zu haben.

Eine digitale Signatur muss sich immer auf eine bestimmte Nachricht beziehen und von der funktional abhängen. Im Prinzip gibt es zwei Arten von digitalen Unterschriften:

- Die digital zu unterschreibende Nachricht wird in Blöcke gleicher Länge aufgeteilt und jeder Block wird einzeln mit dem privaten Schlüssel des Unterzeichners chiffriert.
- Aus der digital zu unterschreibenden Nachricht wird ein Komprimat bestimmter Länge gebildet, und nur dieses Komprimat wird mit Hilfe des privaten Schlüssels des Unterzeichners chiffriert.

Sicherheitsrelevant ist bei der zweiten Art digitaler Unterschriften natürlich auch die Nachrichtenkomprimierung. Wünschenswert wäre hier eine kollisionsresistente Hashfunktion.

9.1.1 RSA-Signaturen

Das RSA-Verfahren kann man auch zur Erzeugung digitaler Signaturen verwenden. Die Idee ist ganz einfach. Alice signiert ein Dokument m , indem sie ihr Entschlüsselungsverfahren auf das Dokument anwendet, also $s = m^d \bmod n$ berechnet, wobei n der RSA-Modul und d der private Schlüssel ist.

Bob verifiziert die Signatur, indem er darauf das Verschlüsselungsverfahren anwendet, also $s^e \bmod n$ berechnet. Ergibt sich daraus das Dokument, so ist die Signatur verifiziert. Es ist nämlich nicht bekannt, wie man ein s berechnen kann, für das $m = s^e \bmod n$ gilt, wenn man den privaten Schlüssel d nicht kennt.

9.1.1.1 Schlüsselerzeugung

Die Schlüsselerzeugung funktioniert genauso wie beim RSA-Verschlüsselungsverfahren. Alice wählt zufällig zwei große Primzahlen p , q und einen Exponenten e mit $1 < e < (p-1)(q-1)$ und $\text{ggT}(e, (p-1)(q-1)) = 1$. Sie berechnet $n = p \cdot q$ und d mit $1 < d < (p-1)(q-1)$ und $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

Ihr öffentlicher Schlüssel ist (n, e) , ihr geheimer Schlüssel ist d .

9.1.1.2 Erzeugung der Signatur

Um die Zahl m zu signieren, berechnet Alice den Wert

$$s = m^d \bmod n.$$

Die Signatur ist s .

9.1.1.3 Verifikation

Bob will die Signatur s verifizieren. Er besorgt sich den öffentlichen Schlüssel (n, e) von Alice und berechnet

$$m = s^e \bmod n.$$

Bob kennt jetzt den Text m . Da er ihn aus der Signatur s gewonnen hat, weiß er, dass s die Signatur von m ist. Er ist sich auch sicher, dass Alice die Signatur s erzeugt hat. Die Signatur s ist nämlich die RSA-Entschlüsselung von m . Solange das RSA-Verschlüsselungsverfahren sicher ist, kann niemand zu der vorgegebenen Nachricht m die Signatur s ohne Kenntnis des privaten Schlüssels d berechnen.

Diese Verifikation kann jeder durchführen, der den öffentlichen Schlüssel von Alice kennt. Will Bob etwa einem Richter gegenüber beweisen, dass Alice den Text m wirklich signiert hat, braucht er nur die Signatur s und den öffentlichen Schlüssel e von Alice vorzulegen. Der Richter kann dann die Verifikation $m = s^e \bmod n$ selber vornehmen.

9.1.2 Angriffe

Zu Beginn der Verifikation besorgt sich Bob den öffentlichen Schlüssel (n, e) von Alice. Wenn es dem Angreifer Oskar gelingt, Bob seinen eigenen öffentlichen Schlüssel als den Schlüssel von Alice unterschieben, kann er danach Signaturen erzeugen, die Bob als Signaturen von Alice anerkennt. Es ist also wichtig, dass Bob den authentischen öffentlichen Schlüssel von Alice hat.

Eine weitere Gefahr beim Signieren mit RSA kommt daher, dass das RSA-Verfahren multiplikativ ist. Sind $m_1, m_2 \in \{0, 1, \dots, n-1\}$ und sind $s_1 = m_1^d \bmod n$ und $s_2 = m_2^d \bmod n$ die Signaturen von m_1 und m_2 , dann ist:

$$s = s_1 s_2 \bmod n = (m_1 m_2)^d \bmod n$$

die Signatur von $m = m_1 m_2$. Aus zwei gültigen Signaturen kann man also leicht eine dritte gültige Signatur erzeugen.

9.2 Blinde Unterschrift

Problemstellung: A möchte, dass B eine Nachricht blind unterschreibt, d.h.

- B soll keine Hinweise über Inhalt der Nachricht erhalten
- B soll lediglich durch seine Signatur beglaubigen, dass er die Nachricht „in der Hand gehabt hat“

Implementierung mit Hilfe von RSA: B besitzt öffentlichen Schlüssel (e, n) , sowie privaten Schlüssel (d, n) . A möchte, dass B die Nachricht m blind unterschreibt:

- A wählt eine Zufallszahl K zwischen 1 und n . A verbirgt die Nachricht m durch: $t = mK^e \bmod n$ ($K =$ Blindungsfaktor) und schickt t an B
- B unterschreibt die unkenntliche Nachricht t :
 $t^d \bmod n = (mK^e)^d \bmod n$
- A rechnet den Blindungsfaktor heraus:
 $s = t^d / K \bmod n$. Dadurch erhält A die von B unterzeichnete Originalnachricht:
 $s = m^d \bmod n$,
denn es ist $t^d = (mK^e)^d = m^d K \bmod n \rightarrow t^d / K \bmod n = m^d K / K \bmod n = m^d \bmod n$

9.3 Digitales Geld

In diesem Unterkapitel wird exemplarisch digitales Geld untersucht.

Ein digitales Geldsystem ist ein elektronisches Zahlungssystem, in dem eine Bank mit mehreren Kunden und Händlern kooperiert. Unter der Voraussetzung, dass die Kunden und Händler bei der Bank ein Konto haben, erlaubt das digitale Geldsystem, dass ein Kunde von seinem Konto digitales Geld abheben kann, um es bei einem Händler zum Bezug von Waren oder zur Beanspruchung von Dienstleistungen auszugeben.

1983 hat David Chaum das Konzept einer blinden Unterschrift (Unterkapitel 9.1), um anonyme digitale Geldsysteme zu ermöglichen.

Im Folgenden wird das Modell nur für eine Münzeinheit beschrieben, für andere Einheiten ist es analog. B (Bank) behält den privaten Schlüssel (d, n) geheim und publiziert den öffentlichen Schlüssel (e, n) .

1. Will sich C eine anonyme digitale Münze ausgeben lassen, erzeugt er eine Zufallsseriennummer m und eine Zufallszahl $K < n$. Aus beiden Werten berechnet er $X = mK^e \bmod n$ und lässt sich X von B blind unterschreiben, d.h. B rechnet $Y = X^d \bmod n$ und gibt Y an C zurück. C kann nun die Zufallszahl K wieder aus Y herausdividieren, indem er $M = Y/K \bmod n$ bildet. C hat damit eine digitale Unterschrift der Nachricht m erhalten, ohne dass B die Nachricht dabei gesehen hätte.
2. C kann nun die Münze M bei einem Händler S ausgeben, und S kann mit Hilfe von (e, n) die Echtheit von M prüfen. Im On-line Modell fragt der Händler bei der Bank nach, ob M schon einmal ausgegeben worden ist. Im Off-line Modell ist die Situation etwas schwieriger, weil B hier zwar nachträglich eine Mehrfacheinlösung von M feststellen kann, dabei aufgrund der Anonymitätseigenschaft aber nicht sagen kann, wer M mehrfach ausgegeben hat.
3. Zuletzt kann S die digitale Münze bei B einlösen und den M entsprechenden Betrag seinem Konto gutschreiben lassen.

Literatur

1. Buchmann, Johannes. *Einführung in die Kryptographie*. Springer Verlag, 1999.
2. Böhmer, Wolfgang. *Virtual Private Networks*. Carl Hanser Verlag, 2002.
3. Oppliger, Rolf. *IT-Sicherheit*. Vieweg Verlag, 1997.
4. Bauer, Friedrich. *Entzifferte Geheimnisse*. Springer Verlag, 1995.
5. Dankmeier, Wilfried. *Codierung*. Vieweg Verlag, 2001.
6. Schneier, Bruce. *Applied cryptography*. Wiley, 1996.
7. Brunnstein, Klaus. *GBI-Skript*. 2002.

Web-Quellen

1. *Rijndael Specification*. <http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf>
2. Reinhard Wobst. *AES unter Beschuss*. <http://www.heise.de/ct/02/21/038/>
3. *AES: Krypto-Standard wahrscheinlich geknackt*.
http://www.chip.de/news_stories/news_stories_8843569.html
4. Nicolas T. Courtois and Josef Pieprzyk. *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*. <http://eprint.iacr.org/2002/044.pdf>
5. *Spezialhardware bedroht möglicherweise RSA-Sicherheit*.
<http://www.heise.de/newsticker/data/wst-05.03.02-002/>
6. Dobbertin, Hans. *Cryptanalysis of MD5 Compress*.
<http://www.cs.ucsd.edu/users/bsy/dobbertin.ps>
7. RSA Laboratories Security Bulletin #4. *On Recent Results for MD2, MD4 and MD5*.
<ftp://ftp.rsa.com/pub/pdfs/bulletn4.pdf>
8. National Institute of Standards and Technology. *Secure Hash Standard*.
<http://csrc.nist.gov/encryption/shs/dfips-180-2.pdf>