Studienarbeit
Von DES zu AES: Anforderungen und Vorgehen beim Standardisierungsprozeß

Universität Hamburg
Fachbereich Informatik
Betreuer: Prof. Dr. K. Brunnstein

Heiko Gerlach

18. April 2000

Zusammenfassung

Der jetzige Standardalgorithmus zur Verschlüsselung (DES) ist nunmehr über 20 Jahre alt. Aufgrund der Fortschritte der Technik ist er nicht mehr als uneingeschränkt sicher zu betrachten. Daher forderte das amerikanische National Institute of Standards and Technology (NIST) am 12. September 1997 dazu auf, Algorithmen einzureichen, die in einem Standardisierungsprozeß evaluiert werden sollen und aus denen ein Nachfolgestandard für den DES-Algorithmus ausgewählt werden soll.

Diese Arbeit soll zunächst einen groben Überblick über die Kryptographie im Allgemeinen geben. Dann sollen die Grundelemente von Blockchiffren kurz dargestellt werden und anschließend eine Übersicht über die Geschichte und Funktionsweise von DES gegeben werden.

Sodann soll der Standardisierungsprozeß, der der Auswahl des AES zu Grunde liegt, beschrieben und die Kandidaten dargestellt werden. Dabei sollen Techniken, die zur Konstruktion dieser Verschlüsselungsalgorithmen benutzt werden und die Elemente aus denen diese bestehen dargestellt werden. Es soll sodann gezeigt werden, wie diese Algorithmen in der Diskussion bewertet werden und inwieweit sie den Kriterien des NIST entsprechen.

Inhaltsverzeichnis

Kapitel 1

Einführung

Am 23. November 1976 wurde der Data Encryption Standard (DES) als amerikanischer Bundesstandard zur Verschlüsselung anerkannt. Der Algorithmus wurde 1983, 1987, 1992 und 1998 als Standard bestätigt.

Durch Standardisierung eines Verschlüsselungsalgorithmuses wurde die Interoperabilität zwischen verschiedenen Anwendungen gesichert und es wurde möglich, die Funktionsweise eines als sicher angesehenen Algorithmuses zu studieren.

DES hat sich über die Jahre als sicher erwiesen und wird in der Zwischenzeit in einer Vielzahl von Anwendungen z. B. im Bankenbereich eingesetzt.

Allerdings besitzt DES auch einige Nachteile:

- Er benutzt einen 56 Bit langen Schlüssel. Dies wurde von Kommentatoren schon während der Standardisierung als zu kurz empfunden.

  1981 wurde geschätzt, daß ein 50 Millionen Dollar teurer Rechner, der auf das Knacken von DES spezialisiert ist, zwei Tage benötigen würde, um einen Schlüssel zu finden.

  Am 15. Juli 1998 wurde die „RSA DES Challenge II-2“ nach 56 Stunden von einer Maschine gelöst, deren gesamte Entwicklungskosten bei etwa 210000 Dollar lagen [17]. Davon entfielen 80000 Dollar auf den Entwurf und den Test der Hardware. Entwickelt wurde diese Maschine von der Electronic Frontier Foundation (EFF), einer Organisation, die sich für Datenschutz und den Erhalt der Bürgerrechte im elektronischen Zeitalter einsetzt, und von Cryptography Research, einer Consulting-Firma, die im Bereich der Kryptographie und Computersicherheit tätig ist.

  Die DES Challenge III wurde inzwischen in unter 23 Stunden gelöst. Beteiligt waren dabei die zuvor erwähnte Maschine sowie distributed.net, eine Organisation, die den Einsatz von verteilten Anwendungen über das Internet fördert.

  Diese Entwicklung ist bedenklich, insbesonders wenn man in Rechnung stellt, daß vertrauliche Informationen auch in einigen Jahren oder gar Jahrzehnten noch geschützt sein müssen.

- DES wurde im Hinblick auf eine Implementation in Hardware entwickelt. Die Struktur von DES eignet sich aber nicht besonders gut für eine Softwareimplementierung, so daß die Leistungsfähigkeit von DES in Software nicht optimal ist.

Die Einführung eines neuen, sichereren Standards erscheint umso dringender, als daß Daten, die heute verschlüsselt werden, unter Umständen noch auf Jahre hinaus gesichert sein müssen.

Auch die zunehmende elektronische Kommunikation erfordert ein erhöhtes Maß an Sicherheit, z. B. bei geschäftlichen Transaktionen über das Internet.

Diese Argumente veranlaßten das National Institute for Standards and Technology (NIST) dazu, mit einer Bekanntmachung im Federal Register die Standardisierung eines neuen Algorithmuses einzuleiten. Das NIST forderte in dieser Mitteilung dazu auf, Kandidaten für einen Algorithmus einzureichen. Diese Algorithmen sollen im Hinblick auf verschiedene Eigenschaften, wie Sicherheit, Performance, Speicherplatzverbrauch etc. evaluiert werden und schließlich soll ein Algorithmus als Standard ausgewählt werden.

Kapitel 2

Grundlagen

Dieses Kapitel soll einen Überblick über die Kryptographie im allgemeinen geben. Insbesonders soll aber auf die mathematischen und informatischen Grundlagen eingegangen werden, die für die nachfolgenden Kapitel benötigt werden.

2.1 Was ist Kryptographie?

Die Kryptologie ist ein Zweig der Mathematik, der sich mit der Absicherung von Nachrichten, bzw. der Umgehung dieser Absicherung beschäftigt.

Das Teilgebiet der Kryptologie, das sich mit der Absicherung von Nachrichten beschäftigt heißt *Kryptographie*. Das Gebiet, das sich mit dem Umgehen von kryptographischen Ansätzen beschäftigt heißt *Kryptanalyse*.

Um eine Nachricht sicher zu übertragen sind mehrere Aspekte zu beachten:

Authentizität Der Empfänger einer Nachricht muß den Absender ermitteln können.

Integrität Der Empfänger einer Nachricht muß feststellen können, ob eine Nachricht verändert wurde.

Verbindlichkeit Der Absender darf nicht abstreiten können, daß er die Nachricht gesendet hat.

Vertraulichkeit Es darf nicht möglich sein, daß Dritte den Inhalt der Nachricht unerlaubt ermitteln.

Um diese Ziele zu erreichen, werden verschiedene kryptographische Techniken eingesetzt. Zu beachten ist auch, daß je nach Zweck der Kommunikation nicht alle dieser Ziele erfüllt sein müssen.

Bei der herkömmlichen Kommunikation mittels Brief wird die Authentizität und Verbindlichkeit durch die Unterschrift des Absenders und die Integrität und Vertraulichkeit durch einen verschlossenen Briefumschlag gewährleistet.

Der *Klartext* ist die ursprüngliche Nachricht, die verschickt werden soll. Sie wird im folgenden in Anlehnung an den englischen Begriff *Plaintext* mit P bezeichnet werden.

Der *Chiffretext (engl. Ciphertext)* ist die verschlüsselte Nachricht, die tatsächlich gesendet wird. Sie soll mit C bezeichnet werden.

Ein *kryptographischer Algorithmus* oder *Chiffrierung* ist die mathematische Funktion, die zum Ver- und Entschlüsseln benutzt wird. Dabei kommt ein *Schlüssel (engl. Key)* zum

Einsatz. Eine mit einem bestimmten Schlüssel verschlüsselte Nachricht läßt sich nur mit dem passenden Schlüssel wieder entschlüsseln. Ein Schlüssel, der zufällig gewählt wird und nur ein einziges mal benutzt wird heißt *Session Key*.

Der *Schlüsselraum (engl. Key Space)* ist die Gesamtheit der zur Verfügung stehenden Schlüssel, die von einem kryptographischen Algorithmus verwendet werden können. Diese Menge soll mit $\mathcal{K}$ bezeichnet werden.

Die *Verschlüsselung (engl. Encryption)* oder Chiffrierung ist der Vorgang des Verschlüsselns. Er soll durch die mathematische Funktion $E_K(P)$ bezeichnet werden. Dies bedeutet, das der Klartext P mit dem Schlüssel K verschlüsselt wird. Die Umkehrung der Verschlüsselung heißt *Entschlüsselung (engl. Decryption)* oder Dechiffrierung. Sie soll durch die mathematische Funktion $D_K(C)$ bezeichnet werden. Der Chiffretext C wird mit dem Schlüssel K entschlüsselt. Stimmt K mit dem Schlüssel überein, der zum Verschlüsseln benutzt wurde, so erhält man den zu C gehörigen Klartext P. Es gilt demnach: $P = D_K(E_K(P))$.

Ein *Kryptosystem* besteht schließlich aus einem Algorithmus einschließlich aller möglichen Klartexte, Chiffretexte und Schlüssel.

2.1.1 Allgemeine Begriffe

Definition 1 (Problem)
Ein Problem P ist eine Menge von Paaren (I, A) von Wörtern über einem Alphabet Σ, so daß jedes Wort $w \in \Sigma^*$ die erste Komponente mindestens eines Paares aus P ist. I heißt Probleminstanz und A heißt Antwort zu I.

Um Probleme zu lösen, werden Algorithmen benutzt, die angeben wie man zu einer Probleminstanz I eine zugehörige Antwort A findet.

Definition 2 (Algorithmus)
Ein *Algorithmus* ist eine präzise Beschreibung eines Verfahrens unter Verwendung von elementaren Verarbeitungsschritten.

Bei einem deterministischen Algorithmus ist an jeder Stelle eindeutig bestimmt, welcher Schritt als nächstes ausgeführt werden soll. Bei nichtdeterministischer Algorithmen ist dies nicht der Fall. An einigen Stellen sind verschiedene Abläufe möglich. Davon wird diejenige Verzweigung gewählt, die zum richtigen Ergebnis führt.

Desweiteren soll hier gefordert werden, daß ein Algorithmus terminiert.

2.1.2 Komplexitätstheorie

Um das Laufzeitverhalten und den Platzbedarf von Algorithmen zu beurteilen, werden die folgenden Definitionen benutzt. Sie geben die Schranken an, in denen eine Funktion in Abhängigkeit von der Größe der Eingabe wächst.

Definition 3 (O-Notation)
Es seien f und g Funktionen sowie c, c_1 und c_2 Konstanten. So ist:

$$\begin{aligned}
\Theta(g(n)) &= \{f(n) | \exists c_1, c_2 > 0 : c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|\} \\
O(g(n)) &= \{f(n) | \exists c : |f(n)| \leq c|g(n)|\} \\
\Omega(g(n)) &= \{f(n) | \exists c > 0 : c|g(n)| \leq |f(n)|\}
\end{aligned}$$

Diese Definitionen werden oft asymptotisch benutzt, d. h. es gibt eine untere Schranke n_0, so daß die Beziehung für alle $n \geq n_0$ gilt. Man schreibt dann:

$$\begin{aligned} f(n) &\in \Theta(g(n)) \quad \text{für} \quad n \to \infty \\ f(n) &\in O(g(n)) \quad \text{für} \quad n \to \infty \\ f(n) &\in \Omega(g(n)) \quad \text{für} \quad n \to \infty \end{aligned}$$

Das folgende Beispiel soll die Anwendung dieser Notation verdeutlichen:

Beispiel 1
$2n + 1 \in O(n^2)$ für $n \to \infty$, da für $c = 5$ und alle $n \geq 1$ gilt: $|2n + 1| \leq c|n^2|$. Das heißt, daß wenn n gegen ∞ geht, der Ausdruck $2n + 1$ nicht schneller wächst als der Ausdruck n^2. Man sagt auch, n^2 ist die obere Schranke.

Um den Aufwand für die Lösung eines Problems einschätzen zu können, werden verschiedene Komplexitätsklassen definiert.

Definition 4 (P)
P ist die Menge der Probleme, die von einem deterministischen Algorithmus in polynomialer Zeit gelöst werden können.

Das heißt, der Algorithmus benötigt $O(p(n))$ Zeit, wobei $p(n)$ ein Polynom ist.

Definition 5 (NP)
NP ist die Menge der Probleme, die von einem nichtdeterministischen Algorithmus in polynomialer Zeit gelöst werden können.

Wird dieser nichtdeterministische Algorithmus durch einen deterministischen simuliert, so benötigt dieser $O(e^n)$ Zeit. Die Berechnungszeit steigt also exponentiell zur Länge der Eingabe.

Definition 6 (polynomialzeit-reduzierbar)
Eine Problem $L_1 \subset \Sigma_1^*$ heißt *polynomialzeit-reduzierbar* auf das Problem $L_2 \subset \Sigma_2^*$ ($L_1 \leq_p L_2$), falls es eine in polynomialer Zeit berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$ gibt, so daß für alle $x \in \Sigma_1^*$ gilt:

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

Definition 7 (NP-Vollständigkeit)
Ein Problem L heißt *schwer* bezüglich **NP**, falls sich jedes Problem $L' \in$ **NP** in polynomialer Zeit auf L reduzieren läßt.

L heißt **NP**-*vollständig*, wenn L schwer für **NP** ist und zusätzlich $L \in$ **NP** gilt.

Dies bedeutet, daß man alle **NP**-vollständigen Probleme in polynomialer Zeit aufeinander abbilden kann. Somit sind diese Probleme äquivalent, da die Transformationszeit im Vergleich zur eigentlichen Zeit, die benötigt wird um das Problem zu lösen, nicht ins Gewicht fällt.

Es gilt offensichtlich **P** $\subseteq$ **NP**. Ein ungelöstes Problem der Informatik ist aber, ob auch **P** $\neq$ **NP** gilt. Wäre dies nicht der Fall, so wäre es möglich, jedes Problem in **NP** in polynomialer Zeit zu lösen. **P** $=$ **NP** gilt genau dann, wenn es ein **NP**-vollständiges Problem L gibt und $L \in P$ gilt.

2.1.3 Mathematische Grundlagen

Hier sollen nun einige nützliche mathematische Grundlagen dargestellt werden, die im folgenden benötigt werden.

Definition 8 (Gruppe)
Das Tupel $(G, \odot)$ bestehend aus einer Menge G und einer Verknüpfung $\odot : G \rightarrow G$, wird als Gruppe bezeichnet, wenn folgende Bedingungen erfüllt sind:

1. das Assoziativgesetz gilt: $\forall a, b, c \in G : a \odot (b \odot c) = (a \odot b) \odot c$.

2. Es existiert genau ein neutrales Element: $\exists e \in G : \forall a \in G : e \odot a = a \odot e = a$.

3. Zu jedem Element aus G gibt es ein inverses Element: $\forall a \in G : \exists a^{-1} : a \odot a^{-1} = a^{-1} \odot a = e$.

Definition 9 (Kommutative (Abelsche) Gruppe)
Eine Gruppe $(G, \odot)$ heißt kommutative oder abelsche Gruppe, wenn zusätzlich das Kommutativgesetz gilt:

4. $\forall a, b \in G : a \odot b = b \odot a$.

Definition 10 (Körper)
Das Tripel $(K, \otimes, \odot)$ bestehend aus einer Menge K und zwei auf dieser Menge definierten Abbildungen $\otimes : K \rightarrow K$ (Addition) und $\odot : K \rightarrow K$ (Multiplikation) wird als Körper bezeichnet, wenn folgende Bedingungen gelten:

1. K ist bezüglich der Addition $\otimes$ eine kommutative Gruppe mit 0 als neutralem Element.

2. $K \backslash \{0\}$ bildet bezüglich der Multiplikation $\odot$ eine kommutative Gruppe.

3. Es gilt das Distributivgesetz: $\forall a, b, c \in K : a \odot (b \otimes c) = (a \odot b) \otimes (a \odot c)$.

2.1.4 Informationstheoretische Grundlagen

Ein Bit ist die kleinste Einheit der Informationsverarbeitung. Es unterscheidet zwei Zustände.

Die *Entropie* einer Nachricht M wird als $H(M)$ bezeichnet. Sie gibt den Informationsgehalt dieser Nachricht an. Wenn eine Nachricht M z. B. n verschiedene Bedeutungen besitzt, die alle gleich wahrscheinlich sind, so beträgt die Entropie dieser Nachricht $H(M) = \log_2 n$ Bit. Diese Konstellation ist aber nach [43] die Unwahrscheinlichste, da in der Realität einige Nachrichten häufiger auftreten als andere.

Die Informationsrate einer Sprache beträgt $r = H(M)/N$, wobei N die Länge der Nachricht ist. Die Informationsrate ist also die Anzahl der Bits, die notwendig ist, um ein Zeichen der Nachricht darzustellen.

Enthält die Sprache L Zeichen, so beträgt die absolute Informationsrate $R = \log_2 L$. Dies ist die maximale Entropie der einzelnen Zeichen, also die Anzahl der Bits, die notwendig ist, um ein Zeichen dieser Sprache darzustellen.

Die *Redundanz* D einer Sprache beträgt $D = R - r$. Die Redundanz einer Nachricht, sind also die Informationen, die zum eindeutigen Dekodieren derselben nicht notwendig sind, da sie in anderer Form schon in der Nachricht enthalten sind.

2.2 Was muß ein gutes Kryptosystem leisten?

Ein sicheres Kryptosystem soll verhindern, daß der Schlüssel entdeckt wird, oder daß der Klartext aus dem Chiffretext ohne Kenntnis des Schlüssels berechnet werden kann. Die folgenden vier Eigenschaften sollten von einem guten Kryptosystem erfüllt werden:

1. Die Sicherheit des Systems darf nur von dem verwendeten Schlüssel abhängen, und D_K sollte ohne Kenntnis des Schlüssels nur „schwer" zu berechnen sein.

2. E_K und D_K sollten effizient zu berechnen sein.

3. Aus dem Chiffretext C dürfen keine Rückschlüsse auf den Klartext P möglich sein.

4. Wenn Informationen über den Klartext bekannt sind (z. B. in welcher Sprache eine Nachricht abgefaßt ist), dürfen diese nicht die Sicherheit des gesamten Kryptosystems beeinflussen.

Diese Forderungen sind teilweise widersprüchlich, da eine effiziente Berechnung der Funktionen im Gegensatz zu den Forderungen nach größtmöglicher Sicherheit steht. Dies ist das größte Problem beim Entwurf von kryptographischen Algorithmen. Sie sollen einen hohen Sicherheitsstandard erfüllen, aber sogleich nur geringe Anforderungen an das Laufzeitverhalten und den Speicherplatzbedarf haben. Insbesonders in modernen Systemen, in denen große Mengen an Daten verschlüsselt übertragen werden müssen, z. B. bei Satellitenübertragungen oder digitalem TV ist dies von Bedeutung.

Punkt 3 fordert, daß jegliche Eigenschaften des Klartextes durch die Verschlüsselungsfunktion so verdeckt werden müssen, daß sie im Chiffretext nicht mehr identifizierbar sind. Dies bedeutet vor allem, daß die Redundanz, die z. B. in der natürlichen Sprache oder in standardisierten Dateiformaten vorhanden ist, nicht mehr nachweisbar sein darf. Dies ist der Fall, wenn der Chiffretext nicht mehr von einer Gleichverteilung zu unterscheiden ist. Um dies zu erreichen, werden zwei Techniken benutzt: *Konfusion* und *Diffusion*. Konfusion verschleiert den Zusammenhang zwischen Klartext und Chiffrat und vereitelt damit die Suche nach Redundanz und statistischen Mustern im Chiffretext. Diffusion verteilt die Redundanz über das Chiffrat und erschwert somit deren Entdeckung.

Forderung 4 soll verhindern, daß partielle Informationen über den Klartext, die dem Angreifer bekannt sind, von diesem dazu benutzt werden können, den Schlüssel oder weitere Teile des Klartextes zu ermitteln.

Einfache Verschlüsselungsverfahren beschränken sich darauf, lediglich Konfusion oder Diffusion in ihren einfachsten Formen zu verwenden:

Substitution Bei einer Substitutionschiffrierung wird jeweils ein Zeichen des Klartextes durch ein anderes Zeichen ersetzt. Dadurch wird Konfusion erreicht.

Ein einfaches Beispiel ist die Caesar-Verschlüsselung, bei der jeder Buchstabe durch das Zeichen ersetzt wird, das im Alphabet drei Stellen später kommt. So wird `DIES IST GEHEIM` durch `GLHV LVW JHKHLP` ersetzt.

Transposition Bei der Transposition, die die Diffusion implementiert, werden die Zeichen des Klartextes permutiert, daß heißt der Chiffretext enthält die gleichen Zeichen wie der Klartext, jedoch in einer anderen Anordnung.

Bei der folgenden Permutation wird der Klartext von links nach rechts und von oben nach unten in das Quadrat geschrieben. Der Chiffretext wird dann von oben nach unten und von links nach rechts gelesen:

1. Klartext: `DIES IST GEHEIM`
2.

D	I	E	S
␣	I	S	T
␣	G	E	H
E	I	M	␣

3. Chiffretext: `D␣␣EIIGIESEMSTH␣`

2.3 Symmetrische Verschlüsselungsverfahren

Symmetrische Verschlüsselungsverfahren haben die Eigenschaft, daß zum Entschlüsseln einer Nachricht der gleiche Schlüssel zum Einsatz kommt wie bei der Verschlüsselung der Nachricht. Dies hat den Nachteil, daß der Schlüssel nur den kommunizierenden Parteien bekannt sein darf, da sonst auch andere die Nachricht lesen und selbst mit diesem Schlüssel verschlüsselte Nachrichten in Umlauf bringen können. Deshalb werden diese Verfahren auch als *Secret Key* Verfahren bezeichnet.

Ein Schlüssel der Dritten bekannt würde, würde die Vertraulichkeit und die Verbindlichkeit von Nachrichten gefährden. Dies wirft das Problem des Schlüsselmanagements auf. Wie werden die Schlüssel, die zum Chiffrieren der Daten benutzt werden verwaltet? Es muß sichergestellt sein, daß die kommunizierende Parteien den gleichen Schlüssel benutzen. Dieser muß aber seinerseits auf einem sicheren Übertragungsweg verteilt werden.

Ein großer Vorteil von symmetrischen Verschlüsselungen ist, daß sie im Vergleich zu asymmetrischen Verfahren eine wesentlich größere Geschwindigkeit haben.

2.3.1 Blockchiffren

Blockchiffren verarbeiten gleichbleibend große Blöcke von mehreren Klartextzeichen. Dazu wird der Klartext P entsprechend in gleichgroße Teile $B_1, B_2, \ldots$ aufgespalten. Diese Blöcke werden dann einzeln verschlüsselt, so daß $C = E_K(P) = E_K(B_1) + E_K(B_2) + \cdots$, wobei $+$ die Verkettung der Chiffretextblöcke darstellen soll. Wie dieses Zusammenfügen der verschlüsselten Blöcke erfolgt, wird genauer in Kapitel 2.9 erläutert.

Die Verschlüsselung eines einzelnen Klartextblockes ist eine umkehrbare Funktion mit $E_K^{-1} = D_K$ und $E_K(B_i) = E_K(B_j) \Leftrightarrow B_i = B_j$.

Blockchiffren setzen eine Kombination aus Diffusion und Konfusion zur Verschlüsselung ein.

2.3.2 Stromchiffren

Stromchiffren verarbeiten im Gegensatz zu Blockchiffren immer ein Zeichen des Klartextes, dies kann ein Bit aber auch ein Byte sein, Dabei ist die Verschlüsselung eines einzelnen Zeichens aber keine Funktion, d. h. daß das gleiche Klartextzeichen in verschiedene Chiffretextzeichen überführt wird. Dies ist jeweils abhängig von der Position des Zeichens im Klartext.

Stromchiffren benutzen Konfusion und wenn sie mit Rückkoppelung arbeiten auch Diffusion. Desweiteren sei hier noch angemerkt, daß auch asymmetrische Stromchiffren existieren.

2.4 Asymmetrische Verschlüsselungsverfahren

Asymmetrische Verschlüsselungsverfahren oder *Public-Key-Verfahren* benutzen im Gegensatz zu symmetrischen Verfahren ein Schlüsselpaar, also zwei zueinander gehörende Schlüssel. Davon wird einer als *öffentlicher Schlüssel (engl. Public Key)* und der andere als *privater Schlüssel (engl. Private Key)* bezeichnet. Der private Schlüssel wird dabei geheim gehalten und der öffentliche Schlüssel publiziert, so daß eine Zuordnung zwischen diesem Schlüssel und dem Besitzer des Schlüssel nachvollziehbar ist.

Die Verwendung zweier zueinander gehörender Schlüssel ermöglicht eine Vielzahl von Anwendungen, die mit symmetrischer Verschlüsselung nur schwer zu verwirklichen sind.

2.4.1 Mathematische Grundlagen

Public-Key-Systeme basieren auf sogenannten Trapdoor-Funktionen, also Funktionen, die in der einen Richtung einfach zu berechnen sind, aber in der Gegenrichtung nur mit Hilfe von zusätzlichem Wissen einfach berechnet werden können. $f(x)$ ist einfach zu berechnen, während es schwer ist, x zu einem vorgegebenen Wert $f(x)$ zu ermitteln. f liegt in **P** und f^{-1} liegt in **NP**. Probleme dieser Art sind die Faktorisierung großer Zahlen oder die Berechnung diskreter Logarithmen in endlichen Körpern.

Als Beispiel soll hier das RSA-Verfahren (nach den Erfindern Rivest, Shamir und Adleman) kurz dargestellt werden. Dieses Verfahren beruht auf dem Problem der Faktorisierung großer Zahlen. Der öffentliche Schlüssel sei K_p (public Key) und der private Schlüssel sei K_s (secret Key).

Zuerst wählt man zwei große Primzahlen p und q und berechnet dann deren Produkt

$$n = pq$$

Dann wird e so gewählt, daß e und $(p-1)(q-1)$ relativ prim zueinander sind, also außer 1 keine gemeinsamen Teiler haben.

Nun wird d berechnet, so daß gilt:

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$

Dies geschieht mit Hilfe des erweiterten Euklidischen Algorithmus. d ist also das Inverse zu e bezüglich $\pmod{(p-1)(q-1)}$.

Der öffentliche Schlüssel ist dann

$$K_p = (e, n)$$

und der private Schlüssel ist

$$K_s = (d, n)$$

Zur Verschlüsselung einer Nachricht wird diese in Zahlen kodiert, die kleiner als n sind. Sodann wird jeder dieser Blöcke $P_1, P_2, \ldots$ einzeln verschlüsselt:

$$C_i = E_{K_p}(P_i) = P_i^e \bmod n$$

Die Entschlüsselung wird dann folgendermaßen ausgeführt:

$$P_i = D_{K_s}(C_i) = C_i^d \bmod n$$

Zur Verschlüsselung kann auch der private Schlüssel eingesetzt werden (natürlich nur von dem Besitzer des Schlüssels) und zur Entschlüsselung dann der öffentliche Schlüssel. Allerdings ist dann keine Vertraulichkeit mehr gewährleistet, da jeder der den öffentlichen Schlüssel kennt die Nachricht dechiffrieren und lesen kann. Für diese Art der Benutzung gibt es aber andere Anwendungen wie z. B. digitale Signaturen (Abschnitt 2.7).

Für den praktischen Einsatz von Public-Key-Verfahren sollten einige Punkte beachtet werden:

Die Primzahlen p und q dürfen auf keinen Fall öffentlich gemacht werden. Da sie für die Benutzung des Verfahrens nicht notwendig sind, können sie also gelöscht werden.

Für die Praxis sollten die Zahlen p und q ausreichend groß gewählt werden, so daß n in der Binärdarstellung mindestens 512 Bit besitzt und die Dezimalzahl also etwa 155 Stellen lang ist. Allerdings wurde im Februar 1999 eine 140 Stellen lange Zahl faktorisiert, so daß für sicherheitsrelevante Anwendungen deutlich größere Zahlen mit 1024 oder 2048 Bit gewählt werden sollten.

2.4.2 Anwendungen

Der große Vorteil von asymmetrischen Verschlüsselungsverfahren besteht darin, daß jeder Benutzer ein Schlüsselpaar (K_s, K_p) aus privatem und öffentlichem Schlüssel besitzt. Durch eine geeignete Infrastruktur kann dann der öffentliche Schlüssel anderen Benutzern des gleichen Systems zugänglich gemacht werden. Wenn diese Infrastruktur einen Mechanismus vorsieht, der die Zuordnung von öffentlichem Schlüssel zu einem Benutzer authentisiert, kann eine sichere Kommunikation zwischen zwei Beteiligten stattfinden, ohne daß diese erst geheime Schlüssel austauschen müssen, oder sich gar kennen. Jede Nachricht wird einfach mit dem öffentlichen Schlüssel des Kommunikationspartners verschlüsselt und dieser benutzt seinen geheimen Schlüssel um die Nachricht zu dechiffrieren.

2.4.3 Nachteile

Asymmetrische Verschlüsselungsalgorithmen benötigen eine wesentlich größere Schlüssellänge als Blockchiffren, wie oben für das RSA-Verfahren erwähnt wurde. Andere Public-Key-Verfahren besitzen diese Eigenschaft auch, wenn die Größe des Schlüssels auch von dem speziellen Verfahren abhängt.

Der Schlüssel und der Nachrichten- bzw. Chiffretext werden mit mathematischen Funktionen transformiert. Mikroprozessoren können mit Operanden dieser Größenordnung nicht umgehen, so daß diese Operationen also in Software durchgeführt werden müssen und deshalb eine wesentlich niedrigere Geschwindigkeit beim Ver- und Entschlüsseln als bei symmetrische Verfahren bedingen. Dieser Nachteil wird durch hybride Systeme ausgeglichen.

2.4.4 Hybride Systeme

Hybride Systeme Verbinden die Vorteile von symmetrischen und asymmetrischen Systemen. Mit einem symmetrischen Algorithmus und einem zufällig gewählten Schlüssel wird die eigentliche Nachricht verschlüsselt. Der Schlüssel, der lediglich den Bruchteil der Größe der eigentlichen Nachricht besitzt, wird sodann mit einem asymmetrischen Verfahren chiffriert und mit der Nachricht versandt.

Nur der Empfänger kann mit seinem privaten Schlüssel den Session Key dechiffrieren und diesen dann dazu benutzen, um die eigentliche Nachricht lesbar zu machen.

2.5 Einweg-Hash-Funktionen

Einweg-Hash-Funktionen (Message-Digest) sind Funktionen, die zu ihren Eingabedaten einen Hashwert berechnet, der eine feste Länge besitzt. Durch den Vergleich zweier Hashwerte kann man ohne die Daten zu kennen, diese auf Gleichheit überprüfen. Umgekehrt ist es aber schwer, zu einem vorgegebenen Hashwert Daten zu finden, die zu diesem passen. Dies schließt auch ein, daß es schwer sein muß, zu vorgegebenen Daten einen zweiten Datensatz zu finden, der den gleichen Hashwert besitzt.

2.6 Message Authentication Code

Message Authentication Codes (MAC) dienen dazu, die Integrität von Nachrichten zu testen und diese zu authentifizieren, d. h. es wird die Authentizität des Absenders sichergestellt. Ein MAC arbeitet wie eine Hash-Funktion, benutzt aber zusätzlich einen geheimen Schlüssel. Nur mit diesem Schlüssel ist es möglich die Integrität der Daten zu überprüfen.

Message Authentication Codes können durch Einweg-Hash-Funktionen in Kombination mit symmetrischen Verschlüsselungsverfahren oder durch die Verwendung einer Blockchiffre, die im CBC-Modus (siehe Abschnitt 2.9) betrieben wird, aufgebaut werden.

2.7 Digitale Signaturen

Digitale Signaturen dienen dazu, den Absender einer Nachricht zu authentisieren und die Integrität der Nachricht sicherzustellen. Hierzu werden asymmetrische Verfahren benutzt. Mit dem in Abschnitt 2.4.1 beschriebenen RSA-Verfahren würde dazu die Nachricht mit dem geheimen Schlüssel des Absenders verschlüsselt, bzw. signiert werden:

$$C = E_{K_s}(P)$$

und die Überprüfung der Signatur erfolgt durch das Entschlüsseln des Chiffretextes mit Hilfe des öffentlichen Schlüssels des Absenders:

$$P = D_{K_p}(C)$$

Aufgrund der Performance-Nachteile von asymmetrischen Verfahren wird aber oft nicht die eigentliche Nachricht signiert, sondern es wird erst ein Hash-Wert über diese gebildet und sodann wird dieser Wert, der wesentlich kürzer als die eigentliche Nachricht ist, signiert.

2.8 Pseudozufallszahlengeneratoren

Pseudozufallszahlengeneratoren (engl. Pseudo Random Number Generators – PRNG) dienen dazu, für vielfältige Zwecke zufällige Zahlen zur Verfügung zu stellen. Die Wahrscheinlichkeit, eine bestimmte Zahl zu erhalten soll für alle möglichen Zahlen gleich groß sein. Ein PRNG wird mit einem Startwert *(engl. Seed)* initialisiert und liefert für unterschiedliche Startwerte jeweils verschiedene Folgen von Zufallszahlen.

Da Computer und die auf ihnen laufende Software inhärent deterministisch sind, ist es faktisch nicht möglich wirklich zufällige Zahlen zu erzeugen. Für einen gegebenen Startwert wird stets die gleiche Folge von Zahlen erzeugt. Hinzu kommt, daß sich die Folge der Werte nach einer Periode wiederholt.

Die Anwendungsgebiete von PRNGs sind vielfältig. Sie dienen dazu, Session Keys, die nur temporär genutzt werden, zu bestimmen und liefern Zufallszahlen, die in vielen kryptographischen Protokollen benötigt werden. Desweiteren benötigen viele Stromchiffren einen PRNG als internen Schlüsselgenerator.

2.9 Betriebsmodi für Blockverschlüsselungsalgorithmen

Blockchiffren verarbeiten immer eine feste Anzahl von einigen wenigen Zeichen (Bytes). So arbeitet DES z. B. mit 64 Bit-Blöcken, also 8 Byte. Da eine zu versendende Nachricht oder abzuspeichernde Daten in der Regel um ein Vielfaches größer sind, sind Mechanismen notwendig, mit denen diese Daten verarbeitet werden können. In [33] sind 4 Modi spezifiziert, mit denen DES betrieben werden soll.

Der einfachste Betriebsmodus ist der *Electronic Code Book (ECB)* Modus. Bei diesem Modus wird der Klartext P in Teilblöcke $P_0, P_1, P_2, \ldots$ aufgeteilt, die eine Größe haben, die dem zugrundeliegenden Algorithmus entspricht. Unter Umständen muß der letzte Teilblock noch mit Füllbytes auf die notwendige Größe gebracht werden. Danach wird jeder Teilblock P_i einzeln verschlüsselt. Die Entschlüsselung erfolgt analog. Damit gilt: $C_i = E_K(P_i)$ und $P_i = D_K(C_i)$.

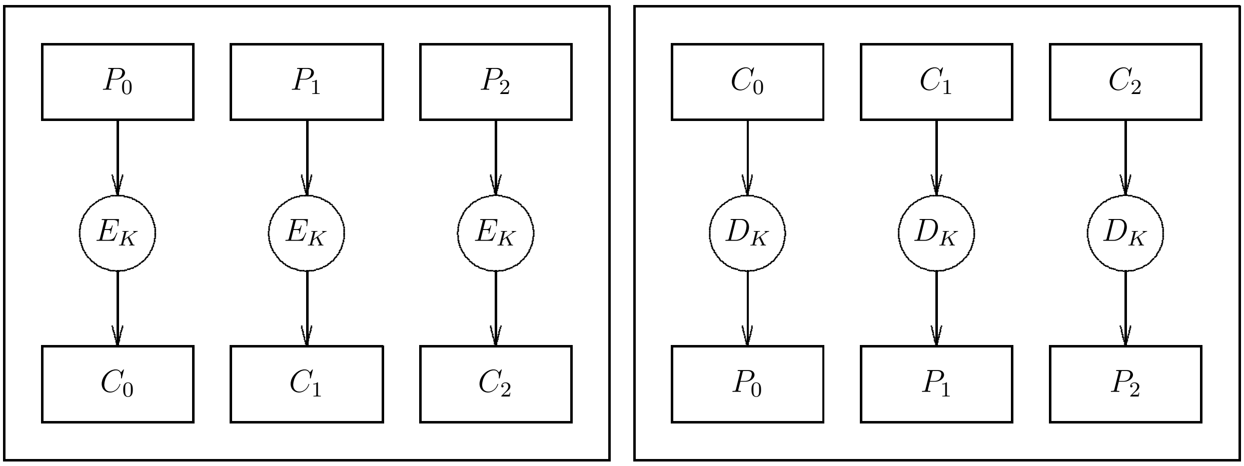

Abbildung 2.1: Ver- und Entschlüsselung im ECB-Modus

Der zweite definierte Modus ist der *Cipher Block Chaining (CBC)* Modus. Hierbei werden die einzelnen Teilblöcke miteinander verkettet, indem vor der Verschlüsselung eines Teilblockes dieser mit dem vorherigen Chiffretextblock XOR-verknüpft wird. Der erste Klartextblock P_0 wird mit einem Initialisierungsvektor IV gefüllt, der eindeutig sein sollte. Dieser kann z. B. aus zufällig gewählten Bytes bestehen. Dieser IV muß nicht geheim gehalten werden. Er dient lediglich dazu, mit dem ersten tatsächlichen Klartextblock P_1 verknüpft zu werden. Dadurch, daß für jede verschlüsselte Nachricht ein eindeutiger IV gewählt wird, ist sichergestellt, daß selbst die gleiche Nachricht P in einen anderen Chiffretext C überführt wird. Dies verhindert, daß einzelne Teilblöcke der Nachricht ausgetauscht werden können, da jeder Teilblock von dem vorherigen abhängig ist. Mathematisch ergibt sich damit für die Verschlüsselung:

$$P_0 = IV$$

$$
\begin{aligned}
C_0 &= E_K(P_0) \\
C_i &= E_K(C_{i-1} \oplus P_i)
\end{aligned}
$$

Für die Entschlüsselung gilt:

$$
P_i = C_{i-1} \oplus D_K(C_i), i > 0
$$

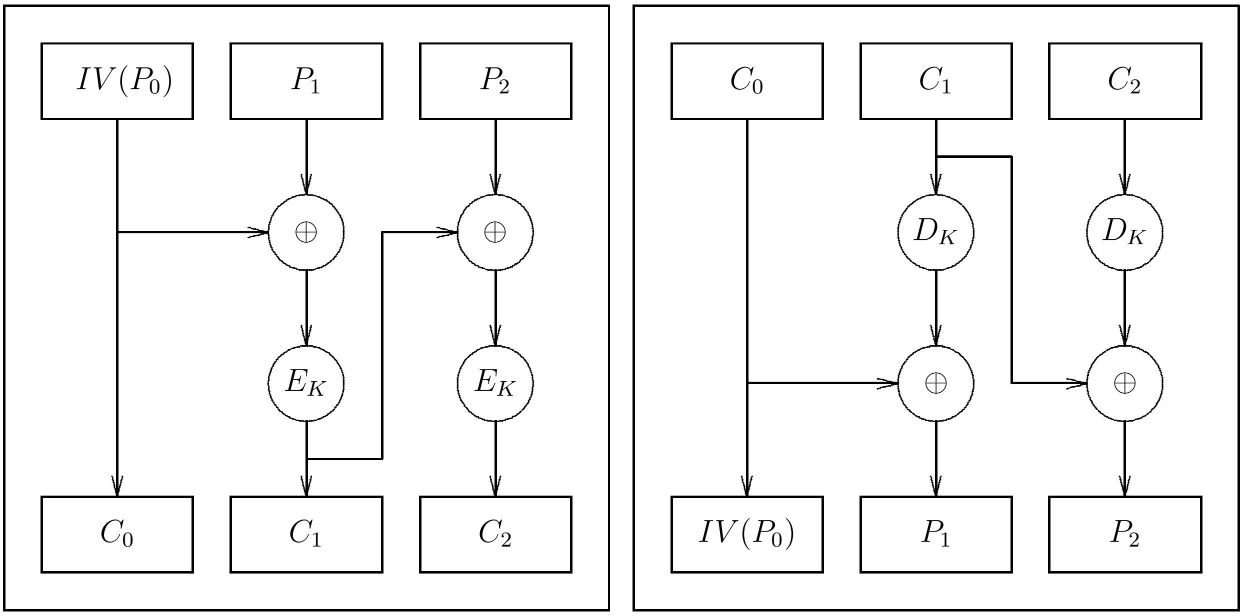

Abbildung 2.2: Ver- und Entschlüsselung im CBC-Modus

Zwei weitere Verfahren beschreiben wie man eine Blockchiffrierung als Stromchiffre betreiben kann. Dies sind der *Cipher Feedback (CFB)* und der *Output Feedback (OFB)* Modus.

In der Literatur werden noch andere Betriebsmodi vorgeschlagen, die verschiedene sicherheitsrelevante Eigenschaften besitzen wie z. B. Selbstsynchronisation oder geringe Fehlerfortpflanzung. Doch in der Praxis bieten diese gegenüber den vier standardisierten Verfahren nur geringe Vorteile.

2.10 Kryptographische Protokolle

Kryptographische Protokolle benutzen die zuvor vorgestellten Bausteine wie symmetrische und asymmetrische Verschlüsselung, Hash-Funktionen, MACs, digitale Signaturen usw., um vielfältige Aufgaben zu lösen. Dies reicht vom sicheren Austausch von Daten, über Verfahren zum Schlüsselaustausch und Authentifizierung, bis hin zu der Möglichkeit geheime Wahlen über ein Computernetzwerk abzuhalten und dabei Betrug zu verhindern.

2.11 Angriffe auf Verschlüsselungsalgorithmen

Will ein Angreifer eine Anwendung kompromittieren, so kann er auf verschiedenen Ebenen ansetzen. Er kann direkt beim Benutzer ansetzen und z. B. durch Bestechung in den Besitz eines Schlüssels gelangen. Er kann aber auch fehlerhafte Protokolle ausnutzen oder wenn diese korrekt sind, eine fehlerhafte Implementationen derselben angreifen. Schließlich bieten auch die verwendeten kryptographischen Elemente, wie Verschlüsselungsalgorithmen, Hash-Funktionen und dergleichen einen Angriffspunkt. Im folgenden sollen nun einige Möglichkeiten aufgezeigt werden, Verschlüsselungsalgorithmen direkt anzugreifen.

Hierzu kann man sich verschiedener Methoden bedienen. Diese sind auf verschiedene Ziele gerichtet wie z. B. das Aufdecken des Klartextes oder eines Schlüssels, der benutzt wurde. Nicht alle dieser Methoden lassen sich gegen jeden Algorithmus einsetzen und die kryptographische Stärke einer Chiffrierung bemißt sich nach der Resistenz gegen solche Angriffe. Die Schwäche eines Algorithmuses liegt oft in der falschen Wahl der verwendeten Elemente. In DES wurden die S-Boxen z. B. so entworfen, daß sie einen guten Schutz gegen die differentielle Kryptanalyse bieten, allerdings war zu diesem Zeitpunkt die Lineare Kryptanalyse noch nicht bekannt, so daß dieser Angriff nicht berücksichtigt werden konnte.

2.11.1 Ciphertext Only

Bei dieser Klasse von Angriffen steht dem Kryptanalytiker nur der Chiffretext zur Verfügung, um den Klartext zu gewinnen. Dies kann geschehen, indem der Schlüssel gefunden wird oder indem er einen Weg findet, um den Klartext auch ohne Schlüssel wiederherzustellen.

2.11.2 Known Plaintext

Bei *Known Plaintext Angriffen* steht dem Angreifer Wissen über den Klartext zur Verfügung. Dies kann z. B. daraus resultieren, daß Nachrichten immer mit einer festgelegten Einleitung beginnen oder daß der konkrete Inhalt einer Nachricht auf andere Weise dem Angreifer zur Kenntnis kommt. Dies kann ausgenutzt werden, um Informationen über den verwendeten Algorithmus und den Schlüssel zu gewinnen.

2.11.3 Chosen Plaintext

Bei *Chosen Plaintext Angriffen* hat der Kryptanalytiker zusätzlich die Möglichkeit den Klartext, der verschlüsselt werden soll, selbst zu wählen. Dies kann geschehen, wenn der Angreifer direkten oder indirekten Zugriff auf das Verschlüsselungsgerät hat. Auf diese Weise läßt sich der Schlüssel, der verwendet wird ermitteln.

2.11.4 Chosen Ciphertext

Ein *Chosen Ciphertext Angriff* ist das Gegenstück zum Chosen Plaintext Angriff. Hierbei hat der Angreifer die Möglichkeit den Chiffretext vorzugeben und aus dem resultierenden Klartext Rückschlüsse auf den Schlüssel zu ziehen.

2.11.5 Brute Force

Die einfachste Möglichkeit um einen Schlüssel zu finden und damit dann den Klartext zu entschlüsseln ist ein *Brute Force Angriff.* Dabei wird der gesamte Schlüsselraum durchsucht und jeder Schlüssel $K \in \mathcal{K}$ wird daraufhin untersucht, ob die Entschlüsselung $D_K(C)$ einen sinnvollen Klartext P ergibt. Dies muß nicht zwangsläufig für die gesamte Nachricht getestet werden – es genügt einige wenige Chiffretextblöcke zu entschlüsseln. Wenn der richtige Schlüssel gefunden wird, kann damit der Rest der Nachricht entschlüsselt werden. Da dem Angreifer nur der Chiffretext zur Verfügung steht, liegt hier ein Ciphertext Only Angriff vor.

Diesen Angriff kann man gegen jeden Verschlüsselungalgorithmus einsetzen. Der Aufwand hierfür ist abhängig von der Anzahl der möglichen Schlüssel, von denen im Mittel die Hälfte getestet werden muß. Die Zeitkomplexität des Angriffs liegt also in $O(|\mathcal{K}|)$, während die Platzkomplexität in $O(1)$ liegt, da man lediglich Speicherplatz für die entschlüsselte Nachricht benötigt und dieser für jeden zu testenden Schlüssel wiederverwendet werden kann und somit konstant ist. Um diesen Angriff zu erschweren, sollte der Schlüsselraum also so groß sein, daß ein bloßes durchprobieren aller Schlüssel so viel Zeit benötigen würde, daß es nicht praktikabel ist.

2.11.6 Lineare Kryptanalyse

Die *Lineare Kryptanalyse* ist ein Known Plaintext Angriff mit dem der Chiffrierschlüssel aufgedeckt werden kann. Hierbei werden die nichtlinearen Elemente des Algorithmuses durch lineare Funktionen approximiert. Mittels eines Klartext-Chiffretext-Paares lassen sich dann einige Bit des Schlüssels rekonstruieren und durch wiederholte Anwendung läßt sich schließlich der ganze Schlüssel für die Approximation des Algorithmuses finden. Ein Bit des so gefundenen Schlüssels ist auch mit einer bestimmten Wahrscheinlichkeit ein Bit des Originalschlüssels.

Allerdings sind hierfür unter Umständen sehr große Mengen an bekanntem Klartext notwendig, so daß dieser Angriff in der Praxis nur schwer anwendbar ist.

2.11.7 Differentielle Kryptanalyse

Die *differentielle Kryptanalyse* ist eine Chosen Plaintext Attacke. Hierbei werden zwei Klartexte P und P' gewählt, die eine bestimmte Differenz ΔP haben. Wie diese Differenz erklärt wird ist von dem Chiffrierungsalgorithmus abhängig. So ist bei DES diese Differenz über die XOR-Verknüpfung definiert.

Sodann werden die beiden gewählten Klartexte verschlüsselt. Aus der Differenz ΔC der Chiffretexte lassen sich Rückschlüsse auf auf einige Schlüsselbits ziehen. Wiederholt man dies für verschiedene Klartextpaare (P, P'), so ergibt sich ein Schlüssel, der am wahrscheinlichsten ist. Werden aber mehr Plaintexte benötigt, als überhaupt möglich sind, so ist dieser Angriff unmöglich. Z. B. ist ein Algorithmus mit einer Blockgröße von 64 Bit gegen diesen Angriff immun, wenn mehr als 2^{64} Klartexte benötigt werden. Aber auch wenn dieser Angriff theoretisch möglich ist, ist dafür unter Umständen soviel Speicherplatz erforderlich, daß er praktisch undurchführbar bleibt.

Kapitel 3

Elemente und Strukturen von Blockchiffren

Chiffrierungsalgorithmen werden aus elementareren Funktionen in einer bestimmten Art und Weise zusammengesetzt. Hier sollen nun einige dieser Funktionen und möglichen Strukturen dargestellt werden.

3.1 Elemente

Im folgenden sollen einige elementare Operationen beschrieben werden, die häufig in kryptographischen Algorithmen benutzt werden. Für sich genommen bieten diese keine große Sicherheit, doch in Kombination mit anderen Operationen und durch wiederholte Anwendung wird ein angemessenes Sicherheitsniveau erreicht.

3.1.1 Addition, Subtraktion und Multiplikation

Diese Operationen sind einfach und effizient zu benutzen. Da dies elementare mathematische Operationen sind, können sie auch leicht beim Einsatz in kryptographischen Algorithmen analysiert werden und so ihre Auswirkungen auf die Sicherheit des gesamten Algorithmuses festgestellt werde.

Häufig werden Operationen aus verschiedenen Gruppen gemischt. So bei IDEA, das die Addition modulo 2^{16} und die Multiplikation modulo 2^{16+1} benutzt.

Diese Operationen können von Mikroprozessoren direkt ausgeführt werden. Dazu sollten die Operanden aber jeweils in ein Maschinenwort des Prozessors passen, da sonst Geschwindigkeitseinbußen auftreten. Dies ist unter anderem bei Berechnungen mit sehr großen Zahlen der Fall, wie sie bei der asymmetrischen Verschlüsselung benutzt werden. Aber auch wenn diese Operationen modulo einem Wert ausgeführt werden, sind zusätzliche Maschinenoperationen notwendig, die die Geschwindigkeit beeinflussen.

3.1.2 Datenabhängige Rotation

Die datenabhängige Rotation wird in dem RC5-Algorithmus benutzt, der von Ron Rivest stammt. Sie ist ebenso einfach und effizient zu benutzen, wie die vorhergehenden Operationen, da entsprechende Befehle von den meisten Mikroprozessoren direkt ausgeführt werden können. Die Rotation nach Links wird mit $\lll$ und die Rotation nach Rechts mit $\ggg$ bezeichnet. Diese

Operationen beziehen sich immer auf eine bestimmte Länge der Bitstrings, die rotiert werden. Diese Länge sollte für eine effektive Ausführung in Maschinensprache jeweils der Wortgröße des Prozessors entsprechen. Ein Beispiel für eine Linksrotation einer Binärzahl um 3 Stellen ist somit $000111_2 \lll 3 = 111000_2$.

3.1.3 Exklusiv-Oder-Verknüpfung

Auch die Exklusiv-Oder-Verknüpfung (XOR) zweier Werte ist eine Standardoperation, die direkt von einem Prozessor ausgeführt werden kann. Sie wird durch $\oplus$ dargestellt und verknüpft die beiden Operanden nach folgender Tabelle:

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

Die XOR-Verknüpfung zweier Bitstrings erfolgt bitweise: $011010_2 \oplus 111000_2 = 100010_2$.

3.1.4 S-Boxen

Substitutions-Boxen oder kürzer *S-Boxen* implementieren die Substitution eines Wertes durch einen anderen Wert, der aus einer Tabelle entnommen wird. Diese Substitution stellt eine nichtlineare Funktion dar, die Konfusion bewirkt. Der Eingabewert legt fest, welche Zeile und welche Spalte der Tabelle ausgewählt werden. Der Wert, der an dieser Stelle steht, wird dann weiterverwendet. Der Ausgabewert muß dabei nicht die gleiche Länge wie die Ausgabe haben.

S-Boxen können fest vorgegeben oder aus dem jeweils verwendeten Schlüssel abgeleitet werden. Aber es ist auch möglich S-Boxen als algebraische Funktion zu definieren, z. B. durch Multiplikation modulo einem konstanten Wert, wie z. B. bei IDEA.

3.1.5 P-Boxen

Permutations-Boxen oder *P-Boxen* permutieren die Bits eines Eingabewertes. Somit besitzt der Ausgabewert die gleiche Länge wie der Eingabewert und Eingabe sowie Ausgabe besitzen die gleiche Anzahl von gesetzten und nicht gesetzten Bits. P-Boxen sind lineare Funktionen und bewirken Diffusion.

Eine P-Box mit einer 5-Bit Ein- und Ausgabe kann z. B. so dargestellt werden:

$$[5, 3, 4, 2, 1]$$

Dies heißt, daß das erste Bit der Ausgabe gleich dem fünften Bit der Eingabe ist. Das zweite Bit der Ausgabe ist das dritte Bit der Eingabe usw. Für eine n-Bit Ein- und Ausgabe gibt es $n!$ mögliche Permutationen.

3.1.6 MDS Matrizen

Eine Maximum Distance Separable Matrix (MDS Matrix) stellt eine Abbildung dar, die einen Vektor $(x_1, x_2, \ldots, x_a)$ mit a Elementen auf einen Vektor $(y_1, y_2, \ldots, y_b)$ mit b Elementen abbildet. Eine solche Abbildung von einem Vektor x nach einem Vektor y kann man als Tupel (x, y) darstellen. Für eine MDS-Abbildung gilt nun: wenn $(x, y) \neq (x', y')$, dann ist die Anzahl der sich unterscheidenden Elemente von x und x' plus die Anzahl der sich unterscheidenden Elemente von y und y' mindestens $b + 1$. Die für die Kryptographie wichtige Eigenschaft ist hierbei, daß die Änderung eines einzigen Elementes des Eingangsvektors sich auf alle Elemente des Ausgangsvektors auswirkt.

3.1.7 Pseudo-Hadamard-Transformation

Die *Pseudo-Hadamard-Transformation (PHT)* dient dazu, zwei Werte a und b in zwei neue Werte zu transformieren. Sie kann besonders effizient implementiert werden. Die beiden notwendigen Transformationen sehen folgendermaßen aus:

$$\begin{aligned} a' &= a + b \bmod 2^{32} \\ b' &= a + 2b \bmod 2^{32} \end{aligned}$$

3.1.8 Whitening

Beim *Whitening* werden Teile des Schlüssels mit der Eingabe der ersten Runde und der Ausgabe der letzten Runde eines Algorithmuses XOR-verknüpft. Dies erschwert Angreifern den Zugriff auf diese Daten und macht Angriffe komplizierter, ohne jedoch die Komplexität der Verschlüsselung zu vergrößern.

3.1.9 Schlüsselexpansion

Die *Schlüsselexpansion (engl. Key Expansion oder Key Schedule)* verwandelt den Benutzerschlüssel K in verschiedenen Teilschlüssel $K_1, K_2, \ldots$, die von dem Algorithmus intern genutzt werden. So kann in jeder Runde des Algorithmuses ein anderer Teilschlüssel angewendet werden (siehe DES: Abschnitt 4.2.1).

Die Schlüsselexpansion ist ein wichtiger Bestandteil eines Verschlüsselungsalgorithmuses. Die Teilschlüssel müssen so erzeugt werden, daß die Sicherheit der Verschlüsselung nicht geschwächt wird und die Berechnung der Teilschlüssel darf die Performance nicht zu stark beeinträchtigen. Desweiteren muß der Speicherbedarf für die Teilschlüssel berücksichtigt werden.

3.1.9.1 Sicherheit

Von der Schlüsselexpansion hängt in nicht unwesentlichen Maße die Sicherheit eines Algorithmuses ab, da sie festlegt, wie die einzelnen Teilschlüssel aussehen. Die folgenden Punkte sollten deshalb vermieden werden:

3.1.9.1.1 Schwache Schlüssel bedingen Teilschlüssel, die alle gleich sind:

$$K_1 = K_2 = \ldots$$

Dies hat zur Folge, daß in jeder Runde des Algorithmuses der gleiche Teilschlüssel zum Einsatz kommt und somit die Sicherheit der Verschlüsselung erheblich geschwächt wird. Bei DES hat dies zur Folge, daß gilt: $E_K(E_K(P)) = P$.

3.1.9.1.2 Halbschwache Schlüssel
sind Schlüssel, die äquivalent sind. Für zwei verschiedene Schlüssel $K \neq K'$ gilt dann:

$$E_K(P) = E_{K'}(P)$$

Meist ist nur ein kleiner Teil des gesamten Schlüsselraumes betroffen. Aber dennoch sollten halbschwache Schlüssel vermieden werden.

3.1.9.1.3 Ein nichtlinearer Schlüsselraum
ist dadurch gekennzeichnet, daß nicht alle Schlüssel gleich stark sind. Der überwiegende Teil der Schlüssel ist schwach und nur wenige Schlüssel besitzen eine bestimmte Form, die sicher ist. Deshalb sollte darauf geachtet werden, daß der Schlüsselraum *flach* ist, also alle Schlüssel gleich stark sind.

3.1.9.2 Performance

Die Performance der Schlüsselexpansion ist ein wichtiger Faktor eines Verschlüsselungsalgorithmuses, da sie einen großen Einfluß auf die Anwendbarkeit der Chiffrierung in einigen Gebieten hat. Dies ist der Fall, wenn der Schlüssel häufig gewechselt wird, wie z. B. bei der Verschlüsselung von digitalem TV, wo ein Sitzungsschlüssel nur einige Sekunden in Gebrauch ist.

3.1.9.3 Speicherplatzbedarf

Der Speicherplatzbedarf für die Teilschlüssel ist insofern von Bedeutung, daß Chiffrierungsalgorithmen auch auf Plattformen laufen müssen, die wenig RAM zur Verfügung stellen. Insbesonders auf Smartcards und in Embedded Systems. Günstig ist es, wenn zu Beginn nicht alle Teilschlüssel berechnet werden müssen, sondern wenn jeder Teilschlüssel erst dann bestimmt werden kann, wenn er auch benötigt wird. Dies ist unter Umständen aber ineffizienter. Wenn nur ein einziger Schlüssel benötigt wird und dieser im Gerät gespeichert werden kann, so ist es auch möglich, die Teilschlüssel im voraus zu berechnen und dann im ROM abzulegen.

3.1.10 Anzahl der Runden

Die Anzahl der Runden, die intern von der Verschlüsselung verwendet werden, ist für die Sicherheit derselben ausschlaggebend. Selbst ein Algorithmus, der nur mäßige Sicherheit bietet, kann durch eine Erhöhung der Rundenzahl sicherer gemacht werden. Aber durch eine größere Rundenzahl wird auch die Arbeitsgeschwindigkeit und somit die Nutzbarkeit eines Algorithmuses eingeschränkt, so daß hier ein Kompromiß zwischen Sicherheit und Benutzbarkeit gefunden werden muß.

3.2 Strukturen

Die im vorigen Abschnitt beschriebenen Elemente können auf verschiedene Weise zu einem Verschlüsselungsalgorithmus kombiniert werden. Dabei spielt die Art und Weise, wie diese Elemente verbunden werden eine entscheidende Rolle für die Sicherheit. Auch muß gewährleistet werden, daß die resultierende Funktion auch wieder umkehrbar ist, da sonst eine Entschlüsselung unmöglich wäre.

Grundsätzlich kann man Verschlüsselungsalgorithmen aber noch auf viele andere Arten aufbauen. So lassen sich Algorithmen z. B. mittels algebraischer Verfahren erzeugen, die mathematisch nachweisbare Eigenschaften, wie die Resistenz gegen Differentielle und Lineare Kryptanalyse besitzen. Allerdings muß man beachten, daß solche Konstruktionen wieder Angriffsmöglichkeiten bieten, die sich aus ihrer besonderen Struktur ergeben.

3.2.1 Produktchiffrierungen

Produktchiffren bauen eine Verschlüsselungsfunktion aus einfacheren Einzelfunktionen auf, die miteinander kombiniert werden und erreichen so eine Kombination aus Diffusion und Konfusion. Die einzelnen einfachen Bausteine sind für sich genommen nicht sicher, aber in ihrer Kombination ergeben sie eine komplexe Struktur, die die Sicherheit des gesamten System sicherstellen.

Als Einzelbausteine kommen dabei Transpositionen (Permutationen), Substitutionen, arithmetische Operationen, XOR-Verknüpfungen und andere Elemente zum Einsatz.

3.2.2 Iterierte Blockchiffrierungen

Bei einer *iterierten Blockchiffrierung* wird eine Rundenfunktion mehrfach angewendet. Diese Rundenfunktion muß für jeden Schlüssel K eine Bijektion sein, damit auch die Chiffrierung umkehrbar ist.

3.2.3 SP-Netzwerke

Substitutions-Permutations-Netzwerke (SP-Netzwerke) bestehen aus mehreren Schichten von Substitution und Permutation, sind also Produktchiffren, die aus Substitution und Transposition (Permutation) aufgebaut sind.

3.2.4 Square

Square ist eine iterierte Blockchiffrierung. Jede Runde ist aus vier verschiedenen invertierbaren Transformationen aufgebaut, die auf einer Matrix von Bytes arbeiten. Diese vier Transformationen sind:

- eine lineare Transformation,
- eine nichtlineare Transformation,
- eine Byte-Permutation und
- die Addition des Rundenschlüssels.

3.2.5 Feistel-Netzwerke

Die meisten Blockchiffren sind Feistel-Netzwerke. Diese sind nach Horst Feistel benannt, der mit anderen ein Team von Kryptographen bei IBM bildete. Dort wurde unter anderem DES entwickelt, das auch ein Feistel-Netzwerk als grundlegende Struktur besitzt (siehe Kapitel 4).

Bei einem Feistel-Netzwerk wird der Eingabeblock in k Teilblöcke aufgeteilt. Auf einige von diesen Blöcken wird dann in aufeinanderfolgenden Runden eine Rundenfunktion f angewendet.

Das Ergebnis dieser Funktion wird mit den unbehandelten Teilblöcken der vorhergehenden Runde Exklusiv-Oder-verknüpft. Danach wird die Reihenfolge der Teilblöcke für die nächste Runde geändert.

Feistel-Netzwerke kann man in verschiedene Klassen einteilen: Typ-1, Typ-2, Typ-3 und verallgemeinerte Feistel-Netzwerke [46]. Wenn der Eingabeblock in $k = 2$ Hälften aufgeteilt wird, so hat das daraus resultierende Typ-1-Netzwerk die Eigenschaft, daß der gleiche Algorithmus zum Ver-, sowie zum Entschlüsseln verwendet werden kann, unabhängig davon, wie die Rundenfunktion f aussieht. Dabei müssen lediglich die Teilschlüssel, die in der Funktion f benutzt werden, in der umgekehrten Reihenfolge angewendet werden. Abbildung 3.1 stellt eine Runde eines solchen Netzwerkes dar.

Der Klartextblock wird in einen rechten und einen linken Teil aufgespalten. Dann wird unter Verwendung des entsprechenden Teilschlüssels die Funktion f auf den rechten Block angewendet und das Ergebnis mit dem linken Teilblock XOR-verknüpft. Danach werden die beiden Teilblöcke vertauscht. Die mathematischen Abhängigkeiten sehen folgendermaßen aus: $L_i = R_{i-1}$ und $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$. Die Entschlüsselung ergibt sich damit als: $R_i = L_{i+1}$ und $L_i = R_{i+1} \oplus f(R_i, K_i)$

Die Funktion $g_{t,i}$ erzeugt das Ergebnis der i-ten Runde. f_i und $f_{i,j}$ sind Rundenfunktionen. Dabei wird weitgehend die Notation aus [46] benutzt.

Definition 11
Sei I_n eine von 2^n Zeichenkette über dem Alphabet $\{0, 1\}$ mit der Länge n, und H_n sei die Menge aller 2^{n2^n} Funktionen $f : I_n \to I_n$. Es gelte $B_j \in I_n$ und $f_i, f_{i,j} \in H_n, g_{t,i} \in H_{kn}$.

Desweiteren seien:

$$\begin{aligned} L_{rot}^{(\rho)}(B_1, B_2, \ldots, B_k) &= (B_{\rho+1}, \ldots, B_k, B_1, B_2, \ldots, B_\rho) \\ R_{rot}^{(\rho)}(B_1, B_2, \ldots, B_k) &= (B_{k-\rho+1}, \ldots, B_k, B_1, B_2, \ldots, B_{k-\rho}) \end{aligned}$$

definiert. $L_{rot}^{(\rho)}$ rotiert $(B_1, \ldots, B_k)$ ρ-mal nach links. $R_{rot}^{(\rho)}$ ist hierzu die inverse Funktion.

Definition 12 (Typ-1-Feistel-Netzwerke)
Das Ergebnis einer Runde ist:

$$g_{1,i}(B_1, B_2, \ldots, B_k) = (B_2 \oplus f_i(B_1), B_3, \ldots, B_k, B_1).$$

Desweiteren sei:

$$\pi_{1,i}(B_1, B_2, \ldots, B_k) = (B_1, B_2 \oplus f_i(B_1), \ldots, B_k).$$

Die Funktionen $\pi_{1,i}$ sind zu sich selbst invers, da

$$\begin{aligned} \pi_{1,i}(B_1, B_2, \ldots, B_k) \circ \pi_{1,i}(B_1, B_2, \ldots, B_k) &= (B_1, B_2 \oplus f_i(B_1) \oplus f_i(B_1), \ldots, B_k) \\ &= (B_1, B_2, \ldots, B_k). \end{aligned}$$

Damit läßt sich schreiben:

$$g_{1,i} = L_{rot}^{(\rho)} \circ \pi_{1,i}$$

und für $f_1, f_2, \ldots, f_s \in H_n$ sei

$$\varphi_1 = g_{1,s} \circ g_{1,s-1} \circ \cdots \circ g_{1,2} \circ g_{1,1}$$

ein Typ-1-Feistel-Netzwerk.

Die zugehörige inverse Funktion ist:

$$\varphi_1^{-1} = \pi_{1,1} \circ R_{rot}^{(1)} \circ \cdots \circ \pi_{1,s-1} \circ R_{rot}^{(1)} \circ \pi_{1,s} \circ R_{rot}^{(1)}.$$

Definition 13 (Typ-2-Feistel-Netzwerke)
Es sei $k = 2l$, $l \in \mathbf{P}$ und $g_{2,i} \in H_{kn}$:

$$g_{2.i}(B_1, B_2, \ldots, B_k) = (B_2 \oplus f_{i,1}(B_1), B_3, B_4 \oplus f_{i,3}(B_3), \ldots, B_{k-1}, B_k \oplus f_{i,k-1}(B_{k-1}), B_1).$$

Somit ist

$$g_{2,i} = L_{rot}^{(1)} \circ \pi_{2,i}$$

mit

$$\pi_{2,i}(B_1, B_2, \ldots, B_k) = (B_1, B_2 \oplus f_{i,1}(B_1), B_3, \ldots, B_{k-1}, B_k \oplus f_{i,k-1}(B_{k-1})).$$

$\pi_{2,i}$ ist auch hier selbstinvers.

Für jede von s Runden sei ein Tupel von Funktionen definiert:

$$h_i = (f_{i,1}, \ldots, f_{i,k-1}).$$

Ein Typ-2-Feistel-Netzwerk wird dann definiert durch:

$$\varphi_2(h_s, h_{s-1}, \ldots, h_2, h_1) = g_{2,s} \circ \cdots \circ g_{2,2} \circ g_{2,1}$$

und die inverse Funktion hierzu ist

$$\varphi_2^{-1}(h_s, h_{s-1}, \ldots, h_2, h_1) = g_{2,1}^{-1} \circ \cdots \circ g_{2,s-1}^{-1} \circ g_{2,s}^{-1}$$

mit $g_{2,i} = \pi_{2,i} \circ R_{rot}^{(1)}$.

Definition 14 (Typ-3-Feistel-Netzwerke)
Es sei $g_{3,i} \in H_{kn}$ und

$$g_{3,i}(B_1, B_2, \ldots, B_k) = (B_2 \oplus f_{i,1}(B_1), B_3 \oplus f_{i,2}(B_2), \ldots, B_k \oplus f_{i,k-1}(B_{k-1}), B_1).$$

Mit

$$\pi_{3,i}(B_1, B_2, \ldots, B_k) = (B_1, B_2 \oplus f_{i,1}(B_1), B_3 \oplus f_{i,2}(B_2), \ldots, B_k \oplus f_{i,k-1}(B_{k-1}))$$

gilt hier:

$$g_{3,i} = L_{rot}^{(1)} \circ \pi_{3,i}$$

$\pi_{3,i}$ ist hier allerdings keine selbstinverse Funktion wie in den Definitionen 12 und 13. Die Umkehrfunktion ist

$$\pi_{3,i}^{-1}(C_1, C_2, \ldots, C_k) = (B_1, B_2, \ldots, B_k)$$

mit $B_1 = C_1$ und $B_j = C_j \oplus f_{i,j-1}(B_{j-1})$.

Damit läßt sich ein Typ-3-Feistel-Netzwerk definieren als

$$\varphi_3(h_s, \ldots, h_2, h_1) = g_{3,s} \circ \cdots \circ g_{3,2} \circ g_{3,1}$$

mit $h_i = (f_{i,1}, f_{i,2}, \ldots, f_{i,k-1})$. Die Umkehrfunktion zu φ_3 ist somit

$$\varphi_3^{-1}(h_s, \ldots, h_2, h_1) = \pi_{3,1}^{-1} \circ R_{rot}^{(1)} \circ \cdots \circ \pi_{3,s-1}^{-1} \circ R_{rot}^{(1)} \circ \pi_{3,s}^{-1} \circ R_{rot}^{(1)}.$$

Definition 15 (Verallgemeinerte Feistel-Netzwerke)
Typ-1- und Typ-2-Feistel-Netzwerke lassen sich aus der Definition des Typ-3-Feistel-Netzwerkes ableiten, indem bestimmte Funktionen aus den Funktionstupeln h_i entfernt werden.

Es sei $\pi_{r,i}$ aus $\pi_{3,i}$ abgeleitet indem einige Funktionen $f_{i,j}$ aus h_i entfernt werden. Es sei $g_{r,i}^{(\rho)} = L_{rot}^{(\rho)} \circ \pi_{r,i}$ mit $1 \leq \rho \leq k-1$. Dann ist für $h_1, h_2, \ldots, h_s$

$$\varphi_r^{(\rho)}(h_s, \ldots, h_2, h_1) = g_{r,s}^{(\rho)} \circ g_{r,2}^{(\rho)} \circ \cdots \circ g_{r,1}^{(\rho)}.$$

Die inverse Funktion $(\varphi_r^{(\rho)})^{-1}$ wird analog zu Definition 14 angegeben.

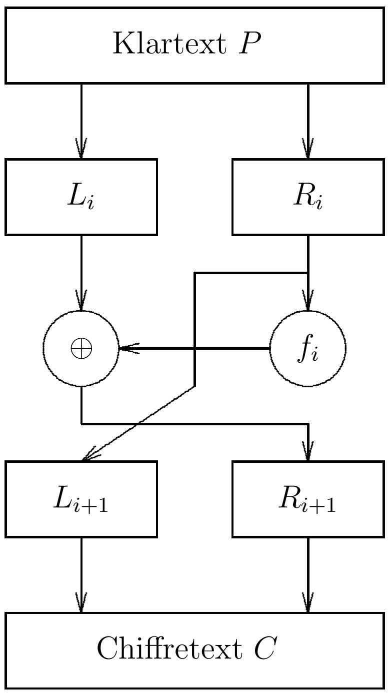

Abbildung 3.1: Eine Runde eines Typ-1-Feistel-Netzwerkes mit $k = 2$

Kapitel 4

Der aktuelle Standard: DES

DES ist ein Verschlüsselungsalgorithmus, der Anfang der 70'er Jahre bei IBM entwickelt wurde. Er basierte auf Lucifer, einem Algorithmus, der von einem IBM-Vorschungsteam gegen Ende der sechziger Jahre entwickelt wurde. Viele Techniken, die in Lucifer benutzt werden, sind auch in DES vorhanden, wie z. B. die Substitution mittels S-Boxen, die sich mit Permutationen abwechselt.

In diesem Kapitel soll die Geschichte und die Funktionsweise von DES, der einen großen Einfluß auf die weitere Entwicklung der Kryptographie hatte, dargestellt werden.

4.1 Standardisierung

1972 gab das National Bureau of Standards (NBS) den Anstoß zu einem Programm zur sicheren Verschlüsselung. Als ein Teil davon war ein standardisierter Verschlüsselungsalgorithmus vorgesehen. Durch Tests und Zertifizierungen sollte die Interoperabilität von Geräten gewährleistet werden.

Am 15.5.1973 wurde dann im Federal Register eine Ausschreibung für einen standardisierten kryptographischen Algorithmus veröffentlicht. Dieser Algorithmus sollte folgende Entwicklungsziele haben:

- hoher Grad an Sicherheit,
- vollständige Spezifikation und Nachvollziehbarkeit,
- kein geheimer Algorithmus, die Sicherheit darf nur vom Schlüssel abhängen,
- der Algorithmus muß allen Anwendern zur Verfügung stehen,
- vielseitige Anwendung,
- kostengünstige Implementation in elektronischen Komponenten,
- effizient in der Benutzung,
- Validierbarkeit und
- Exportierbarkeit.

Von den Algorithmen, die eingesandt wurden, erfüllte keiner diese Kriterien, so daß am 27.8.1974 eine zweite Ausschreibung erfolgte. Hieraufhin wurde von IBM ein Algorithmus eingereicht, der auf Lucifer basierte, der zu Beginn der 70'er Jahre ebenfalls bei IBM entwickelt wurde. Dieser Algorithmus konnte effizient in Hardware implementiert werden und IBM erklärte sich auch bereit dem NBS eine unentgeltliche Lizenz zur Verfügung zu stellen.

Schließlich wurde der Algorithmus und die Lizenz am 17.3.1975 veröffentlicht und zu Stellungnahmen aufgefordert. Diese Aufforderung wurde am 1.8.1975 nochmals wiederholt.

Es wurden dann zwei Workshops zur Evaluierung des Algorithmus abgehalten. Diese beschäftigten sich mit den mathematischen Grundlagen und der Schlüssellänge von DES. Am 23.11.1976 wurde er schließlich als Bundesstandard zur Verwendung für alle nichtgeheimen Regierungsvorgänge freigegeben und am 15.1.1977 wurde die offizielle Beschreibung des Standards in [32] veröffentlicht. Sechs Monate später trat er dann in Kraft.

1980 wurden in [33] vier Betriebsmodi für DES veröffentlicht. Dies waren die in Abschnitt 2.9 dargestellten Verfahren: ECB, CBC, CFB und OFB.

ANSI, als industrielles Standardisierungsgremium, erkannte DES 1981 als Standard für den privaten Sektor an. Der Algorithmus wird hier als Data Encryption Algorithm (DEA) bezeichnet (ANSI X3.92). Ebenso wurden die vier Betriebsmodi festgelegt (ANSI X3.106). Später folgten weitere Standards, z. B. für Finanztransaktionen von der Financial Institution Retail Security Group des ANSI, die einen Standard zur Verwaltung und Geheimhaltung von PINs (ANSI X9.8) auf Basis von DES entwickelte.

Die ISO standardisiert keine Verschlüsselungsalgorithmen. Dennoch benutzt sie DES in einem internationalen Authentifizierungsstandard und einem Standard zur Schlüsselverwaltung (International Wholesale Financial Standards Group der ISO).

Vom NBS wurde zu Beginn festgelegt, daß DES alle fünf Jahre auf seine Sicherheit hin überprüft werden soll. In den Jahren 1983, 1988 und 1993 wurde DES jeweils bestätigt. Zuletzt geschah dies 1998. Doch schon 1988 gab es Bedenken den Standard fortzuschreiben, da es absehbar war, daß die Sicherheit, die DES bietet, auf absehbare Zeit nicht mehr ausreichen würde. DES wurde aber dennoch als Standard fortgeschrieben, da kein Ersatz vorhanden war und dieser Algorithmus schon eine weite Verbreitung gefunden hatte.

4.2 Funktionsweise

DES ist ein Typ-1-Feistel-Netzwerk (siehe Definition 12) mit $k = 2$ Teilblöcken und 16 Runden. Die Rundenfunktion ist $f_i(P_i) = f(K_i, P_i)$.

DES arbeitet auf 64 Bit-Blöcken und benutzt einen 64 Bit-Schlüssel. Von diesen 64 Bit werden jedoch 8 Bit zur Paritätsprüfung benutzt, so daß effektiv nur 56 Bit als Schlüssel zur Verfügung stehen. Deshalb werden die 8 Paritätsbit häufig auch weggelassen, so daß der Schlüssel dann nur als 56 Bit Wert angegeben wird. Wird dies getan, so entfällt der erste Teil der Schlüsselexpansion, der die Paritätsbit entfernt.

Die Struktur von DES ist in Abbildung 4.1 dargestellt. Der Klartext wird zuerst einer Eingangspermutation IP unterworfen und dann in zwei 32 Bit große Blöcke B_0 und B_1 aufgeteilt. Danach folgen 16 Runden eines Feistelnetzwerkes, wobei für die i-te Runde gilt: $B_{i+1} = B_{i-1} \oplus f(K_i, B_i)$. Dabei ist K_i ein Rundenschlüssel, der durch die Schlüsselexpansion aus dem Benutzerschlüssel K abgeleitet wird.

Nach den 16 Runden erfolgt eine Vertauschung der letzten beiden Blöcke B_{16} und B_{17} und daraufhin wird die zu IP inverse Permutation IP^{-1} angewandt.

Die Entschlüsselung kann mit demselben Algorithmus erfolgen. Dabei müssen lediglich die Teilschlüssel K_i in der umgekehrten Reihenfolge verwendet werden.

4.2.1 Schlüsselexpansion

Die Schlüsselexpansion verwandelt den Benutzerschlüssel in 16 Rundenschlüssel, die von der Rundenfunktion f verwendet werden. Diese Expansion kann im Voraus erfolgen, wobei die Rundeschlüssel dann in einem Vektor gespeichert werden müssen, oder der Rundenschlüssel kann dann berechnet werden, wenn er benötigt wird.

Zuerst werden die Paritätsbit entfernt und der Schlüssel so auf eine Länge von 56 Bit verkürzt und permutiert. Danach werden aus diesen 56 Bit in jeder Runde i 48 Bit für den Rundenschlüssel K_i ausgewählt.

Dazu wird der 56 Bit-Schlüssel in zwei 28 Bit große Hälften geteilt. Diese werden jeweils in Abhängigkeit von der aktuellen Runde i um ein oder zwei Bit rotiert. Danach werden durch eine Kompressionspermutation 48 Bit ausgewählt. Hierdurch wird nicht nur festgelegt welche Bits von K in K_i übernommen werden, sondern es wird auch die Reihenfolge des Auftretens in K_i gegenüber derjenigen in K verändert.

4.2.2 Rundenfunktion

Die Funktion, die in jeder Runde auf einen Block angewandt wird, ist die Rundenfunktion $f_i(B_i) = f(K_i, B_i)$. Sie setzt sich ihrerseits wieder aus elementareren Funktionen zusammen.

Es werden zuerst jeweils 8-mal 6 Bit aus B_i ausgewählt. Da B_i aber nur aus 32 Bit lang ist, werden einige Bits mehrfach gewählt. Diese Operation heißt deshalb auch Expansionspermutation. Ebenso werden 8-mal 6 Bit aus K_i gewählt. Da der Rundenschlüssel aber schon eine Länge von 48 Bit besitzt wird auch jedes Bit nur einmal auftreten. Diese beiden 48 Bit langen Werte werden nun bitweise XOR-verknüpft und ergeben so wieder 8 Werte mit je 6 Bit.

Diese 8 Werte dienen nun als Eingabe für 8 S-Boxen, die als Ergebnis jeweils einen 4 Bit langen Wert liefern. Also insgesamt 32 Bit, auf die nun eine P-Box angewandt wird.

Mit E als Expansionpermutation, S als Anwendung der S-Boxen und P als Anwendung der P-Box gilt:

$$f_i = P \circ S \circ \oplus_{K_i} \circ E$$

Dabei ist $\oplus_{K_i}$ die oben beschriebene XOR-Verknüpfung mit den Bit des Rundenschlüssels.

4.3 Sicherheit und Schwachstellen

DES wurde Anfang der 70'er-Jahre entwickelt und 1977 als Standard anerkannt. Man kann also auf über 2 Jahrzehnte der Analyse dieses Algorithmuses zurückblicken.

DES hat sich gegen viele Angriffe resistent gezeigt. Aber auch einige Probleme wurden aufgedeckt. Diese Erkenntnisse haben die Theorie der Kryptographie vorangebracht. Hier sollen nun einige theoretische, aber inzwischen auch praktisch durchführbare Angriffe kurz dargestellt werden.

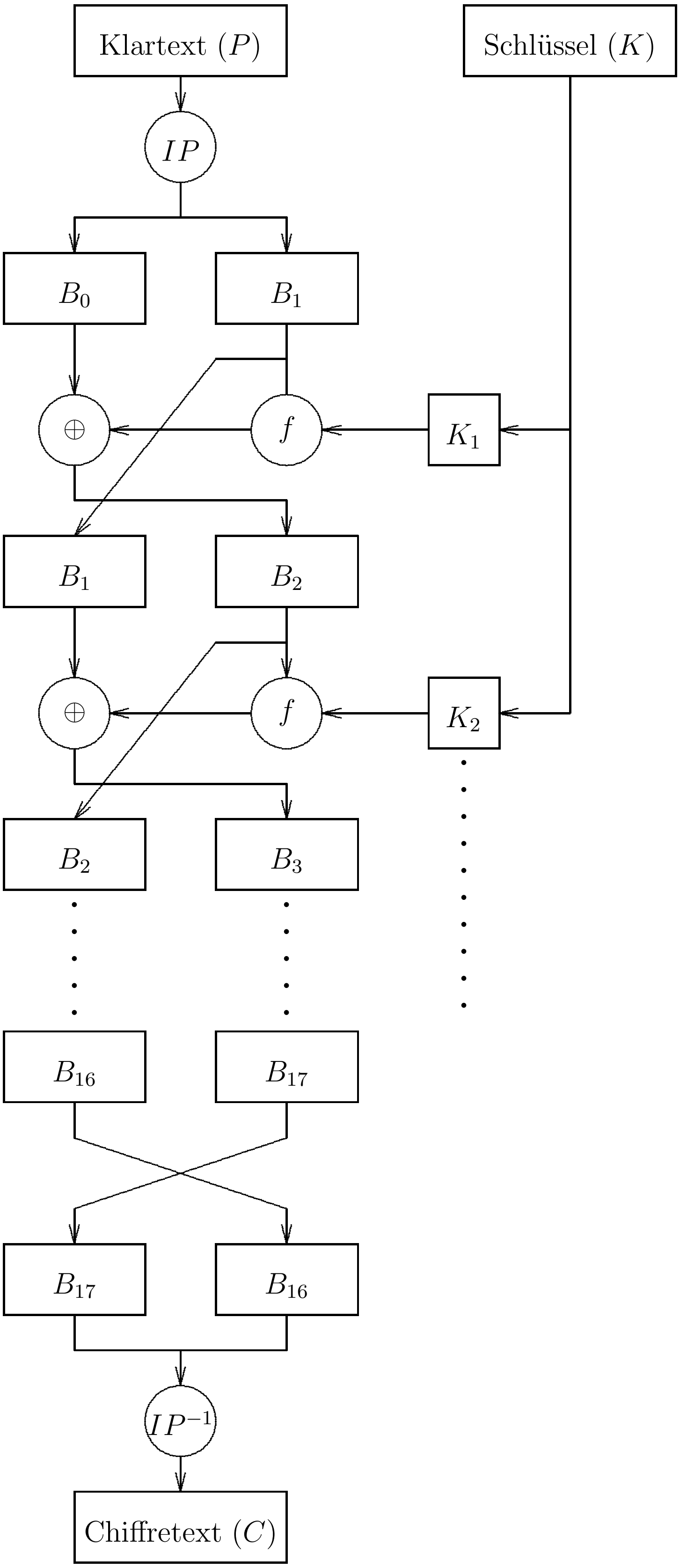

Abbildung 4.1: Die Struktur von DES

4.3.1 Brute Force

Der Data Encryption Standard benutzt einen 56 Bit langen Schlüssel und somit sind insgesamt 2^{56} verschiedene Schlüssel möglich. Diese alle der Reihe nach durchzutesten bis der richtige Schlüssel gefunden wird ist inzwischen realisierbar. Dies ist vor allem darauf zurückzuführen, daß Computer und Computernetzwerke, insbesonders das Internet, eine weite Verbreitung gefunden haben und eine koordinierte Schlüsselsuche zu geringen Kosten möglich ist, indem viele Rechner über das Internet kooperieren und jeweils einen kleinen Teil des Schlüsselraumes durchsuchen. Desweiteren ist es inzwischen auch möglich spezielle Hardware zu relativ geringen Kosten herzustellen, die nach DES-Schlüsseln sucht.

Die Firma RSA Security forderte dazu auf, geheime DES-Schlüssel zu suchen, um die Anfälligkeit dieses Algorithmuses gegen Brute Force Angriffe zu zeigen. Tabelle 4.1 zeigt die verschiedenen DES-Challenges und die Zeiten, die benötigt wurden, um diese Aufgaben zu lösen. (Angaben von `http://www.rsasecurity.com` und `http://www.eff.org`)

Challenge	durchgeführt von	gelöst am	benötigte Zeit
DES-I	Rocke Verser et. al.		96 Tage
DES-II-1	distributed.net	24.2.1998	41 Tage
DES-II-2	EFF	15.7.1998	56 Stunden
DES-III	distributed.net und EFF	13.1.1999	22.25 Stunden

Tabelle 4.1: Benötigte Zeit für Brute Force Angriffe gegen DES

Der Descracker ist eine spezielle Maschine, die zum Brute-Force-Testen von DES-Schlüsseln entwickelt wurde. Die Maschine besteht aus Chips, die nur zu dem Zweck entworfen wurden, DES-Schlüssel zu testen. Jeder dieser Chips enthält 24 „Search Units“, die unabhängig voneinander einen Bereich des Schlüsselraumes durchsuchen. 64 Dieser Chips sind auf einem Board zusammengefaßt und 12 Boards sind wiederum in einem Chassis vereinigt. Der DES Cracker besteht aus 2 Chassis. Diese Maschine benötigt etwa 9 Tage, um den gesamten Schlüsselraum von DES zu durchsuchen [17]. Dieser Zeitraum läßt sich allerdings durch eine größere Anzahl von Chips verkleinern, wobei allerdings auch höhere Kosten anfallen.

4.3.2 Differentielle Kryptanalyse

Die differentielle Kryptanalyse von DES wurde Anfang der 90'er Jahre von Eli Biham und Adi Shamir durchgeführt. Eine ausführliche Beschreibung des Angriffs findet sich in [9].

Für diesen Angriff werden etwa $2^{47} \approx 10^{14}$ gewählte Klartext und $2^{37} \approx 10^{11}$ DES-Verschlüsselungen benötigt. Auf den ersten Blick scheint es so, als wäre dieser Angriff erfolgreicher, als ein Brute Force Angriff, der 2^{56} Verschlüsselungen benötigt. Doch da die differentielle Kryptanalyse ein Chosen Ciphertext Angriff ist, müssen die gewählten Klartexte mit dem angegriffenen Schlüssel chiffriert werden. Dies in der Praxis zu arrangieren ist so gut wie unmöglich. In [12] führt Don Coppersmith aus, daß dieser Angriff den Entwicklern von DES bekannt war und der Algorithmus entsprechend ausgelegt wurde.

4.3.3 Lineare Kryptanalyse

Die lineare Kryptanalyse von DES wurde von Mitsuru Matsui in [30] vorgestellt. Sie basiert auf statistischen linearen Beziehungen zwischen Plaintext, Chiffretext und dem benutzten

Schlüssel. Für diesen Angriff gegen DES werden $2^{47} \approx 10^{14}$ bekannte Klartext benötigt. Dies ist einfacher zu bewerkstelligen als ein Chosen Plaintext Angriff, da die Klartexte nur bekannt sein müssen, aber nicht so gewählt werden müssen, daß sie bestimmte Eigenschaften besitzen und dann noch mit dem entsprechenden Schlüssel chiffriert werden müssen.

Kapitel 5

Anforderungen an den Advanced Encryption Standard

Am 2. Januar 1997 gab das NIST bekannt, daß es beabsichtigt einen neuen Verschlüsselungsstandard zu verabschieden und stellte die Anforderungen, die an diesen Standard zu stellen seien zur öffentlichen Diskussion. Diese Anforderungen wurden dann auf einem öffentlichen Workshop am 15. April 1997 diskutiert. In [34] forderte das NIST am 12. September 1997 dazu auf, Verschlüsselungsalgorithmen einzusenden und gab die Kriterien bekannt, die diese Algorithmen erfüllen sollten. Sodann wurde auch das weitere Vorgehen zur Standardisierung in diesem Dokument beschrieben.

Der Standard soll nur einen einzigen Algorithmus berücksichtigen, der mindestens so sicher wie Triple-DES ist, aber wesentlich effizienter arbeitet. Der Übergang von DES zu dem neuen Standard soll in mehreren Jahren vollzogen werden.

5.1 Das Verfahren der Standardisierung

Das NIST forderte dazu auf Kandidatenalgorithmen bis zum 15. Juni 1998 einzusenden. Wenn diese die Minimalanforderungen und formalen Bedingungen erfüllten, so sollten sie in einem Evaluierungsverfahren untersucht werden. Als Ergebnis würde dann ein Algorithmus als zukünftiger Standard gewählt.

5.1.1 Zeitlicher Ablauf

Auf einer Konferenz vom 20. bis 22. August 1998 in Ventura (Kalifornien, USA) wurden 15 Kandidaten vorgestellt, die für den Evaluierungsprozeß akzeptiert wurden.

Danach folgte die erste Runde der Auswertung. Der Schwerpunkt lag hier auf der Evaluation der Geschwindigkeit bei einer Schlüssellänge von 128 Bit. Das NIST machte während dieser Phase aber keine eigenen Kryptanalyse, sondern griff auf die Auswertungen zurück, die von Dritten veröffentlicht wurden.

Auf der zweiten Konferenz am 22. und 23. März 1999 in Rom folgte eine öffentliche Diskusion der AES-Kandidaten basierend auf den Erkenntnissen aus der ersten öffentlichen Runde der Auswertung.

Danach wurde aufgrund der in der ersten Evaluationsrunde gewonnenen Erkenntnisse eine Auswahl von fünf Kandidaten getroffen, die in einer zweiten Evaluationsrunde genauer betrachtet werden sollen. In dieser Runde werden weitere technische Auswertungen gemacht.

Aber auch rechtliche Aspekte wie Patente werden geprüft.

Auf der dritten Konferenz im April 2000 in New York sollen die Ergebnisse dieser zweiten Runde diskutiert werden und anschließend ein Algorithmus gewählt werden.

5.1.2 Minimalanforderungen

Als Minimalanforderungen an die Kandidatenalgorithmen wurden die folgenden Bedingungen gestellt:

- der Algorithmus muß eine symmetrische Chiffre sein,
- der Algorithmus muß ein Blockchiffre sein,
- er muß mit einer Blöckegröße von 128 Bit arbeiten und
- er muß Schlüssellängen von 128, 192 und 256 Bit verarbeiten.

Ein Algorithmus kann aber auch über diese Forderungen hinausgehen und z. B. weitere Blockgrößen oder Schlüssellängen anbieten.

5.1.3 Formalien

Weiterhin gibt es formale Bedingungen für die Einsendung von Algorithmen. Es wird gefordert, daß bestimmte Materialien der Bewerbung beigefügt werden.

Gefordert wird ein *Cover Sheet*, das Namen und Anschrift des Einsenders sowie Angaben zu den Erfindern und Entwicklern enthält. Der Einsendung soll die *Dokumentation* des Algorithmuses beiliegen und auf *magnetische Medien* sollen verschiedene Implementationen bereitgestellt werden. Zuletzt sollen *Angaben über anwendbare U.S. oder andere Patente* vorhanden sein.

5.1.4 Dokumentation

Die Dokumentation der Chiffren besteht aus verschiedenen Teilen. Einige dieser Anforderungen sind optional. Aber auch über diese Anforderungen hinausgehende Erläuterungen sind erwünscht, da sie helfen einen Algorithmus genauer zu bewerten.

5.1.4.1 Spezifikation des Algorithmuses

Die Dokumentation soll eine vollständige Spezifikation des Algorithmuses enthalten. Diese Spezifikation soll alle Angaben enthalten, die notwendig sind, um den Algorithmus zu implementieren. Dies schließt mathematische Gleichungen, Tabellen, Diagramme und Parameter ein, die die Funktionsweise der Chiffre beeinflussen.

Desweiteren sollten auch die Design-Kriterien erläutert werden, die der Auswahl von Elementen des Algorithmuses zugrunde liegen, um den öffentlichen Auswertungsprozeß zu erleichtern.

5.1.4.2 Performance Schätzungen

Die Dokumentation soll weiterhin eine Einschätzung der Effizienz des Algorithmuses für Soft- und Hartwareimplementierungen enthalten. Diese sollten mindestens für die vom NIST spezifizierte „AES Analysis Plattform“ enthalten sein. Diese besteht aus einem PC mit Intel Pentium Pro Prozessor, der mit 200 MHz getaktet ist, 64 MB Hauptspeicher und Windows 95 als Betriebssystem. Als Compiler kommt der ANSI C Compiler der Borland C++ Development Suite 5.0 zum Einsatz und für Java Implementierungen das Java Development Kit 1.1 von Javasoft.

Performance Einschätzungen für weitere Plattformen liegen im Ermessen der Einsender. Jedoch sollten dann jeweils genaue Angaben über die eingesetzte Hard- und Software gemacht werden, so daß eine unabhängige Überprüfung von dritter Seite möglich ist.

Für folgende Operationen soll die Anzahl der benötigten Takte für die jeweilige Plattform angegeben werden:

- die Verschlüsselung eines Blockes,
- die Entschlüsselung eines Blockes,
- die Initialisierung des Algorithmuses,
- das Schlüsselsetup und
- die Änderung eines Schlüssels.

Diese Angaben sollen für alle Kombinationen von Schlüssel- und Blockgröße gemacht werden, die in Abschnitt 5.1.2 angegeben sind. Desweiteren sollen Angaben zu Tradeoffs von Geschwindigkeits- und Speicherplatzanforderungen enthalten sein.

5.1.4.3 Statistische Tests

Es sollen auch statistische Tests in der Dokumentation enthalten sein. Dies sind *Known Answer Test (KAT)* und *Monte Carlo Tests (MCT)*.

5.1.4.3.1 Known Answer Tests Known Answer Tests sollen die Eigenschaften eines Algorithmuses dokumentieren. Sie sollen wiederum für alle Kombinationen aus Block- und Schlüsselgröße angegeben werden. Diese Tests bestehen aus mehreren einzelnen Test, die alle im ECB-Modus durchgeführt werden.

Variable Key Known Answer Test Bei diesem Test ist jedes Bit des Plaintextes auf 0 gesetzt. Im Schlüssel wird genau ein Bit auf 1 gesetzt und der Plaintext dann verschlüsselt. Dies wird für alle Bitpositionen des Schlüssels wiederholt. Somit ergibt sich ein Tripel bestehend aus dem Schlüssel, dem Null-Plaintext und dem daraus resultierenden Chiffretext.

Variable Plaintext Known Answer Test Dieser Test ähnelt dem vorherigen. Jedoch werden alle Bit des Schlüssels auf 0 gesetzt und jeweils genau ein Bit des Plaintextes auf 1 gesetzt.

Interne Zwischenergebnisse Wenn der Algorithmus interne Zwischenergebnisse produziert, etwa wenn er aus mehreren Runden besteht, so sollen jeweils für eine Ver- und eine Entschlüsselungsoperation diese Zwischenergebnisse angegeben werden.

Interne Tabellen Wenn der Algorithmus interne Tabellen benutzt, so sollen Testergebnisse vorhanden sein, die jeden Tabelleneintrag berücksichtigen.

5.1.4.3.2 Monte Carlo Tests Für alle Kombinationen aus Schlüssel- und Blockgrößen sollen diese Tests durchgeführt werden. Sie bestehen aus vier Einzeltests, von denen zwei im ECB- und zwei im CBC-Modus durchgeführt werden. Dabei werden sowohl die Ver- als auch die Entschlüsselung getestet. Das Verfahren für die Monte Carlo Tests wird in [2] beschrieben.

Für diesen Test wird ein erster Klartext und Schlüssel gewählt und aufgezeichnet. Danach folgen 10000 Verschlüsselungen, wobei als Klartext der Chiffretext der vorhergehenden Verschlüsselung benutzt wird. Nach diesen 10000 Iterationen wird der daraus resultierende Chiffretext gespeichert und ein neuer Schlüssel gebildet, indem der alte Schlüssel mit bestimmten Bits des Chiffretextes XOR-verknüpft wird. Dieser Vorgang wird vierhundertmal wiederholt. Die Entschlüsselung erfolgt analog. Für die beiden Tests im CBC-Modus muß zu Anfang noch ein Initialisierungsvektor gewählt werden.

5.1.4.4 Sicherheitseinschätzungen

Eine Einschätzung der Stärke der Chiffre soll in der Dokumentation enthalten sein. Sowohl für jede Kombination von Schlüssel- und Blockgrößen, die in Abschnitt 5.1.2 aufgezählt werden, soll eine solche Einschätzung vorgenommen werden, als auch für alle anderen Kombinationen, die der Algorithmus unterstützt.

Desweiteren soll eine Analyse in Bezug auf die Anfälligkeit gegenüber bekannten Angriffen wie Known oder Chosen Plaintext Attacken gemacht werden. Es sollen eventuell vorhandene schwache Schlüssel oder andere sicherheitsrelevante Eigenschaften des Algorithmuses angegeben werden.

Eine weitere Forderung ist ein mathematischer Beweis für die Nicht-Existenz von Hintertüren (engl. Trapdoors), soweit dies möglich ist.

Weiterhin ist schließlich die Angabe von Literaturhinweisen auf sicherheitsrelevante Untersuchungen zu diesem Algorithmus erwünscht und wenn möglichen sollen diese Artikel der Einsendung des Algorithmuses beigefügt werden.

5.1.4.5 Vor- und Nachteile des Algorithmuses

In diesem Teil der Dokumentation sollen Vor- und Nachteile in Bezug auf die Anwendbarkeit des Algorithmuses in verschiedenen Bereichen dargestellt werden. Inwieweit ist der Algorithmus als Baustein für andere kryptographische Elemente wie Stromchiffren, MACs, PRNGs oder Hash-Funktionen geeignet? Eignet er sich für die Implementation auf 8-Bit Prozessoren? Gibt es Vor- oder Nachteile für spezielle Anwendungen, wie z. B. in Geldautomaten, bei ISDN oder digitalem TV? Und inwieweit läßt sich der Algorithmus in Hardware verwirklichen?

Desweiteren soll angegeben werden welche Schlüssel- und Blockgrößen der Algorithmus über die Minimalanforderungen hinausgehend unterstützt.

5.1.5 Magnetische Medien

Auf magnetischen Medien, soll der Source-Code für verschiedene Implementierungen eines Algorithmuses sowie einige andere Programme und Daten mitgeliefert werden.

5.1.5.1 Referenzimplementation

Enthalten sein soll eine Referenzimplementation in ANSI C, die die Funktionsweise eines Algorithmuses verdeutlichen soll. Diese Implementation soll alle Funktionen des Algorithmuses beinhalten und diese Funktionen für alle Kombinationen aus Block- und Schlüsselgrößen, die in Abschnitt 5.1.2 gefordert werden sowie alle anderen Kombinationen, die der Algorithmus unterstützt, implementieren.

Das NIST hat ein API spezifiziert, das von allen Kandidaten benutzt werden soll.

5.1.5.2 Mathematisch optimierte Implementationen

Mathematisch optimierte Versionen sollen in ANSI C und in Java vorhanden sein, um die Performance der Algorithmen mit verschiedenen Programmiersprachen zu testen.

Die ANSI C Implementation soll ebenfalls das NIST API benutzen und die Java Implementation soll auf der „Java Cryptography Architecture“ aufbauen, so daß für alle Algorithmen die gleichen Voraussetzungen gelten.

Implementiert werden sollen hierbei nur die Kombinationen aus Block- und Schlüsselgröße, die in den Minimalanforderungen spezifiziert sind. Die Programme sollen aber im ECB-, CBC- und 1-Bit-CFB-Modus laufen.

5.1.5.3 Implementation der statistischen Tests

Desweiteren soll der Code mit dem die Ergebnisse der statistischen Test erzeugt wurden mitgeliefert werden. Dieser soll sowohl für die Referenzimplementation, als auch für die optimierten Versionen vorhanden sein. Die Ergebnisse dieser Test sollen ebenfalls auf Disketten zugänglich sein.

5.1.6 Weitere Dokumentation

Desweiteren soll die Dokumentation auch in elektronischer Form einer Einsendung beigefügt sein, um sie einer breiten Öffentlichkeit zur Verfügung zu stellen. Einschränkungen gibt es hierbei allerdings für den Source-Code der Programme, der den U.S.-Exportbeschränkungen für Kryptographie unterliegt.

5.1.7 Patente

Das NIST verlangt desweiteren eine Erklärung über das Vorhandensein von anwendbaren Patenten. Es wird in [34] ausgeführt, daß viele der potentiellen Anwender darauf gedrungen haben, daß der Algorithmus weltweit zur Verfügung steht, ohne das Lizenzgebühren gezahlt werden müssen. Das NIST geht davon aus, daß genügend Algorithmen eingesendet werden, die diese Voraussetzung erfüllen.

5.1.8 Evaluation

Die Einsendungen, die die Voraussetzungen aus den Abschnitten 5.1.2 bis 5.1.7 erfüllen werden danach einem Evaluationprozeß unterzogen. Das NIST selbst wertet die Algorithmen aus, es ist aber auch jeder andere aufgerufen Auswertungen zu machen. Diese werden ebenfalls in die Beurteilung eines Algorithmuses mit einbezogen.

Die Evaluation der Kandidaten auf deren Tauglichkeit als zukünftiger Verschlüsselungsstandard beruht auf den vom NIST in [34] bekanntgemachten Kriterien.

5.1.8.1 Sicherheit

Das wohl wichtigste Kriterium, um einen Verschlüsselungsalgorithmus zu evaluieren und zu bewerten ist seine Sicherheit, also der Aufwand, der benötigt wird, um ihn zu kryptanalysieren. Hierunter fällt die Sicherheit eines Algorithmuses im Vergleich zu den anderen eingereichten Algorithmen, die Zufälligkeit des Chiffretextes, die Korrektheit der mathematische Grundlagen sowie weitere Gesichtspunkte, die erst während der Evaluation auftreten. Angriffe gegen einen Algorithmus werden vom NIST auf dessen Praktikabilität hin untersucht.

5.1.8.2 Kosten

Die Kosten eines Algorithmuses werden vom NIST unter verschiedenen Gesichtspunkten untersucht.

5.1.8.2.1 Lizenzbedingungen Das NIST beabsichtigt den zukünftigen Standard weltweit ohne Abgaben zugänglich zu machen.

5.1.8.2.2 Effizienz Wie Effizient ein Algorithmus arbeitet wird sowohl für Soft- als auch für Hardwareimplementationen evaluiert. Insbesonders wird hierbei die Geschwindigkeit eines Algorithmuses in Betracht gezogen. Hierfür werden die mathematisch optimierten Implementation herangezogen. Von Dritten eingereichte Ergebnisse für Plattformen, die sich von der vom NIST spezifizierten unterscheiden, werden mit berücksichtigt.

Dies gilt natürlich auch für den folgenden Punkt:

5.1.8.2.3 Speicheranforderungen Auch die Speicheranforderungen der Algorithmen wird in der Evaluation berücksichtigt. Dies gilt wiederum sowohl für Soft- als auch Hardwareimplementationen. Bei Softwareimplementationen ist die Programmgröße sowie der benötigte Speicher von Interesse. Auch hierfür kommen die optimierten Versionen der Algorithmen zum Einsatz.

5.1.8.3 Eigenschaften

Ein weiteres Entscheidungskriterium sind die spezifischen Eigenschaften eines Algorithmuses, die seinen Einsatz in bestimmten Gebieten beeinflussen.

5.1.8.3.1 Flexibilität Algorithmen mit einer größeren Flexibilität werden bevorzugt. Darunter fällt, ob ein Algorithmus auch andere Blockgrößen oder Schlüssellängen verarbeiten kann, als in Abschnitt 5.1.2 gefordert. Aber auch, inwieweit der Algorithmus auf 8-Bit-Prozessoren implementierbar ist, sich für Geldautomaten, digitales Fernsehen oder Satellitenkommunikation eignet. Unter diesen Punkt fällt auch, inwieweit sich der Algorithmus als Baustein für weitere kryptographische Elemente wie z. B. MACs oder Stromchiffren eignet.

5.1.8.3.2 Implementierbarkeit in Hard- und Software Der Verschlüsselungsalgorithmus soll sowohl in Soft- als auch in Hardware implementierbar sein.

5.1.8.3.3 Einfachheit Unter diesem Gesichtspunkt wird das Design eines Algorithmuses bewertet. Je einfacher dieser ist, desto besser.

5.2 Eingereichte Algorithmen

Bis zum 15. Juni 1998 wurden insgesamt 21 Algorithmen als Kandidaten für den AES eingesandt. Davon wurden jedoch nur 15 für den weiteren Auswahlprozeß zugelassen. Nach [1] wurden folgende Algorithmen abgelehnt, da die Bewerbungsunterlagen nicht vollständig waren:

GEM von Lance Gharat,

RAINBOW von Samsung Advanced Institute of Technology,

Simple von Richard Frank,

TMD von MobileSafe LLC,

Vobach Technology von Design Automations Systems, Inc. und

WICKER'98 von LAN Crypto, Inc.

Die folgende Tabelle 5.1 nach [6] soll schon einen kurzen Überblick über die eingereichten Chiffren und ihr Design geben.

Chiffre	Typ	Rundenzahl
CAST-256	Feistel	48
Crypton	Square	12
Deal	Feistel	6, 8, 8
DFC	Feistel	8
E2	Feistel	12
Frog		8
HPC		8
LOKI97	Feistel	16
Magenta	Feistel	6, 6, 8
Mars	Feistel	32
RC6	Feistel	20
Rijndael	Square	10, 12, 14
SAFER+	SP-Netzwerk	8, 12, 16
Serpent	SP-Netzwerk	32
Twofish	Feistel	16

Tabelle 5.1: Die Kandidatenalgorithmen

Im folgenden sollen die 15 Kandidaten vorgestellt werden, die als Kandidaten für den Advanced Encryption Standard in Frage kamen und in der ersten Evaluationrunde ausgewertet wurden. Insbesonders sollen hier ein Überblick über die Strukturen und die Funktionsweisen der wichtigsten Elemente der Algorithmen gegeben werden. Für eine detailierte Beschreibung der Chiffren sei hier auf die Originalliteratur verwiesen.

5.2.1 CAST-256

CAST-256 [18] wurde von Entrust Technologies, Inc. (Kanada) entwickelt. Dieser Algorithmus ist ein modifiziertes Feistel-Netzwerk.

5.2.1.1 Schlüsselexpansion

Die Schlüsselexpansion von CAST-256 hat die Aufgabe für jede von 48 Runden zwei Schlüssel k_{i,r_j} und k_{i,m_j} (mit $i = 0, \dots, 11$ und $j = 0, \dots, 3$) bereitzustellen. k_{i,r_j} ist 5 Bit groß und k_{i,m_j} ist ein 32-Bit Wert. Hierzu wird der Benutzerschlüssel in acht 32-Bit Worte aufgeteilt, von denen einige mit 0 initialisiert werden, wenn der Schlüssel kleiner als 256 Bit groß ist. Auf diese Datenwörter werden dann 12 Runden, die dann eine Feistel-ähnliche Struktur, ähnlich der der Verschlüsselung besitzen. Auch hier kommen die Rundenfunktionen f_1, f_2 und f_3 zum Einsatz. Aus den Zwischenergebnissen der 12 Runden werden die Rundenschlüssel abgeleitet.

5.2.1.2 Verschlüsselung

Der Datenblock wird mit dem 128 Bit großen Plaintext initialisiert: $B_0 = P$. Dann erfolgt die Verschlüsselung in zwei Schritten. Für die ersten 6 Runden mit $i = 0, \dots, 5$ gilt:

$$B_{i+1} = Q_i(B_i)$$

und für die darauffolgenden 6 Runden mit $i = 6, \dots, 11$ gilt:

$$B_{i+1} = \overline{Q}_i(B_i).$$

Der Chiffretext ist dann $C = B_{12}$.

Die Funktionen Q und $\overline{Q}$ bestehen aus vier Operationen, die eine Feistel-ähnliche Struktur besitzen. Hierzu wird der 128-Bit Eingabeblock in vier 32-Bit Blöcke aufgeteilt:

$$B_i = B_i^0 || B_i^1 || B_i^2 || B_i^3.$$

In Q wird nun jedes 32-Bit-Wort mit dem Ergebnis einer Rundenfunktion von dem darauffolgenden Datenwort verknüpft:

$$\begin{aligned}
B_{i+1}^2 &= B_i^2 \oplus f_1(B_i^3, k_{i,r_0}, k_{i,m_0}) \\
B_{i+1}^1 &= B_i^1 \oplus f_2(B_{i+1}^2, k_{i,r_1}, k_{i,m_1}) \\
B_{i+1}^0 &= B_i^0 \oplus f_3(B_{i+1}^1, k_{i,r_2}, k_{i,m_2}) \\
B_{i+1}^3 &= B_i^3 \oplus f_1(B_{i+1}^0, k_{i,r_3}, k_{i,m_3}).
\end{aligned}$$

In $\overline{Q}$ erfolgt dies in der umgekehrten Reihenfolge und ist somit die Inversion von Q:

$$\begin{aligned}
B_{i+1}^3 &= B_i^3 \oplus f_1(B_i^0, k_{i,r_3}, k_{i,m_3}). \\
B_{i+1}^0 &= B_i^0 \oplus f_3(B_i^1, k_{i,r_2}, k_{i,m_2}) \\
B_{i+1}^1 &= B_i^1 \oplus f_2(B_i^2, k_{i,r_1}, k_{i,m_1}) \\
B_{i+1}^2 &= B_i^2 \oplus f_1(B_{i+1}^3, k_{i,r_0}, k_{i,m_0}).
\end{aligned}$$

Das 128-bittige Ergebnis von Q, bzw. $\overline{Q}$ ist dann

$$B_{i+1} = B_i^0 || B_i^1 || B_i^2 || B_i^3.$$

Insgesamt besitzt CAST-256 also $4 \cdot 12 = 48$ Runden und benutzt drei Rundenfunktionen f_1, f_2 und f_3. Diese nehmen als Eingabe zwei Rundenschlüssel und einen 32-Bit Eingabewert I. Dieser kann alternativ auch als Folge von vier Bytes $I_a, \ldots, I_d$ betrachtet werden, wobei I_a das *least significant Byte* und I_d das *most significant Byte* ist.

j bezeichnet im folgenden die aktuelle Runde innerhalb der Funktion Q, bzw. $\overline{Q}$ mit $j \in \{0, 1, 2, 3\}$. Einer der Rundenschlüssel k_{i,m_j} wird als *masking Key* bezeichnet und besteht aus 32 Bit. Der andere wird als *Rotation Key* k_{i,r_j} bezeichnet und ist 5 Bit groß.

5.2.1.2.1 f_1 Zuerst wird der Eingabewert mittels zweier Rundenschlüssel modifiziert:

$$I' = (k_{i,m_j} + I) \lll k_{i,r_j}.$$

Danach folgt die Anwendung von S-Boxen auf einzelne Bytes:

$$I'' = (S_1(I'_a) \oplus S_2(I'_b)) - S_3(I'_c) + S_4(I'_d).$$

I'' ist die Ausgabe der Funktion.

5.2.1.2.2 f_2 Der Aufbau dieser Rundenfunktion gleicht dem von F_1. Jedoch werden die Ergebnisse der S-Box-Anwendung auf eine andere Art und Weise verknüpft:

$$\begin{aligned} I' &= (k_{i,m_j} \oplus I) \lll k_{i,r_j} \\ I'' &= (S_1(I'_a) - S_2(I'_b) + S_3(I'_c)) \oplus S_4(I'_d). \end{aligned}$$

5.2.1.2.3 f_3 Auch hier gleicht der Aufbau dem von f_1. Aber in dieser Funktion werden nicht nur die Ausgaben der S-Boxen unterschiedlich verknüpft, sondern auch die Verknüpfung mit den Rundenschlüsseln geschieht auf eine leicht geänderte Weise:

$$\begin{aligned} I' &= (k_{i,m_j} - I) \lll k_{i,r_j} \\ I'' &= ((S_1(I'_a) + S_2(I'_b)) \oplus S_3(I'_c)) - S_4(I'_d). \end{aligned}$$

5.2.1.3 Entschlüsselung

Dadurch daß in der Verschlüsselung zwei zueinander inverse Teile verwendet werden, ist es möglich den gleichen Algorithmus zur Entschlüsselung zu benutzen. Dazu müssen aber die Schlüssel k_{i,r_j}, k_{i,m_j} in der umgekehrten Reihenfolge angewendet werden.

5.2.1.4 Verwendete Operationen

Dieser Algorithmus verwendet verschiedene Operationen. Dies sind die bitweise XOR-Verknüpfung, die Linksrotation von 32-Bit Datenwörtern sowie die Addition und Subtraktion modulo 2^{32}. Desweiteren werden vier S-Boxen mit 8 Bit als Eingabe und 32 Bit als Ausgabe verwendet. Diese S-Boxen sind konstant.

5.2.2 CRYPTON

Crypton [27] ist ein Algorithmus, der von Chae Hoon Lim bei Future Systems, Inc. (Korea) entwickelt wurde und auf dem Prinzip von Square beruht. Crypton verarbeitet 128-Bit-Blöcke und besitzt 12 Runden. Die Schlüssellänge ist variabel von 64 bis 256 Bit in Schritten von je 32 Bit. Der Algorithmus ist selbstinvers, d. h. es wird der gleiche Algorithmus zum Ver- als auch zum Entschlüsseln benutzt. Lediglich die Rundenschlüssel sind dabei verschieden.

5.2.2.1 Schlüsselexpansion

Um die benötigten Rundenschlüssel zu ermitteln werden zuerst mittels der Funktionen $\tau, \gamma, \sigma, \pi$ sowie der XOR-Verknüpfung einige interne Werte berechnet. Aus diesen werden dann durch Linksrotationen und die XOR-Verknüpfung die eigentlichen Rundenschlüssel abgeleitet.

5.2.2.2 Verschlüsselung

Crypton arbeitet mit zwölf Runden. In jeder dieser Runden wird eine Rundentransformation angewendet. Diese ist allerdings für gerade und ungerade Runden unterschiedlich:

$$\begin{aligned} \rho_{oK}(B) &= (\sigma_K \circ \tau \circ \pi_o \circ \gamma_o)(B) \\ \rho_{eK}(B) &= (\sigma_K \circ \tau \circ \pi_e \circ \gamma_e)(B) \end{aligned}$$

ρ_o wird in ungeraden Runden und ρ_e in geraden Runden auf B angewendet.

Die im folgenden beschriebenen Funktionen arbeiten auf einem 128-Bit Block, der als Vektor von vier 32-Bit Blöcken angesehen wird:

$$B = B_3||B_2||B_1||B_0.$$

5.2.2.2.1 Nichtlineare Substitution γ Diese Funktion wird durch zwei konstante S-Boxen implementiert, die jeweils acht Bit als Eingabe und acht Bit als Ausgabe besitzen. Es wird also immer ein Byte ersetzt. Um einen 128-Bit Block zu ersetzen sind also 16 S-Box-Substitutionen notwendig. Diese sind so angeordnet, daß gilt:

$$\gamma_o(\gamma_e(B)) = \gamma_e(\gamma_o(B))$$

5.2.2.2.2 Lineare Transformation π π implementiert eine bitweise Permutation. Hierzu wird die Maskierung einzelner Bit mit einer AND-Verknüpfung und die XOR-Verknüpfung eingesetzt. Hierbei gilt für π_o und π_e:

$$\pi_e(B_3||B_2||B_1||B_0) = \pi_o(B_2||B_1||B_0||B_3).$$

5.2.2.2.3 Lineare Transformation τ Diese Funktion ist eine Permutation über den Bytes des Blockes B, die neu angeordnet werden.

5.2.2.2.4 Schlüsseladdition σ In diesem Schritt werden vier Rundenschlüssel zu je 32 Bit mittels der XOR-Funktion mit dem Datenblock verknüpft.

5.2.2.3 Entschlüsselung

Für die Entschlüsselung kann die gleiche Funktion wie zur Verschlüsselung verwendet werden. Allerdings müssen die Rundenschlüssel in der umgekehrten Reihenfolge angewendet werden.

5.2.2.4 Verwendete Operationen

In Crypton werden verschiedene Operationen verwendet, die alle auf Byte-Ebene arbeiten. Dies sind die S-Boxen, die ein Byte als Ein- und Ausgabe besitzen, die Permutation von Bytes sowie die bitweise XOR- und AND-Verknüpfung, die auf Bytes oder noch größeren Datenwörtern arbeiten. Die Linksrotation, die während der Schlüsselexpansion benutzt wird, rotiert ein Datenwort um acht oder sechzehn Bit, also ein, bzw. zwei Byte.

5.2.3 DEAL

DEAL [25] wurde von Lars Knudsen (Norwegen) entwickelt und von Richard Outerbridge (Kanada) als AES-Vorschlag eingesandt. Der Algorithmus ist ein Typ-1-Feistel-Netzwerk mit $k = 2$ Blöcken, das intern DES als Rundenfunktion benutzt.

5.2.3.1 Schlüsselexpansion

Es gibt für jede der drei Schlüssellängen einen eigenen Algorithmus für die Schlüsselexpansion.

Für Schlüsselgrößen von 128 und 192 Bit wird eine Rundenzahl von $r = 6$ und für 256 Bit Schlüssellänge werden $r = 8$ Runden vorgeschlagen. Ein Rundenschlüssel RK_i ist $64r$ Bit lang.

Beispielhaft soll nachfolgend für 128-Bit Schlüssel gezeigt werden, wie aus dem Benutzerschlüssel K, der hierbei aus zwei 64 Bit Worten K_1 und K_2 besteht, die 6 Rundenschlüssel RK_i mit $i = 1, \ldots, 6$ berechnet werden. Dabei ist E die Verschlüsselungsfunktion von DES und KI ist ein interner Schlüssel, der fest vorgegeben ist.

$$\begin{aligned}
RK_1 &= E_{KI}(K_1) \\
RK_2 &= E_{KI}(K_2 \oplus RK_1) \\
RK_3 &= E_{KI}(K_1 \oplus 2^{63} \oplus RK_2) \\
RK_4 &= E_{KI}(K_2 \oplus 2^{62} \oplus RK_3) \\
RK_5 &= E_{KI}(K_1 \oplus 2^{60} \oplus RK_4) \\
RK_6 &= E_{KI}(K_2 \oplus 2^{56} \oplus RK_5)
\end{aligned}$$

Für die Schlüssellängen von 192 und 256 Bit erfolgt dies nach dem gleichen Schema.

5.2.3.2 Verschlüsselung

Sei P ein 128 Bit großer Plaintextblock. B_0^L und B_0^R seien die linke, bzw. rechte Hälfte von P. Der Chiffretextblock, also daß Ergebnis der Verschlüsselung sei $C = B_r^L || B_r^R$, also die Konkatenation der Ergebnisblöcke nach r Runden.

Für eine Runde von DEAL gilt:

$$B_j^L = B_{j-1}^R \oplus E_{RK_j}(B_{j-1}^L)$$

$$B_j^R = B_{j-1}^L$$

Hierbei ist E wiederum die Verschlüsselungsfunktion von DES.

5.2.3.3 Entschlüsselung

Die Entschlüsselung erfolgt mit dem gleichen Algorithmus. Dabei werden lediglich die Rundenschlüssel in der umgekehrten Reihenfolge verwendet und zu Beginn und am Ende der Entschlüsselung die Blöcke vertauscht, also:

$$\begin{aligned} B_0^L &= C_0^R \\ B_0^R &= C_0^L \\ P_r^L &= B_r^R \\ P_r^R &= B_r^L \end{aligned}$$

Dabei ist der Chiffretextblock $C = C_0^L || C_0^R$ und der entschlüsselte Plaintext ist $P = P_r^L || P_r^R$.

5.2.4 DFC

DFC (Decorrelated Fast Cipher) [19] wurde von Serge Vaudenay am Centre National pour la Recherche Scientifique (Frankreich) entwickelt.

5.2.4.1 Schlüsselexpansion

Die Schlüsselexpansion von DFC verwandelt den Benutzerschlüssel K in acht Rundenschlüssel $K_1, \ldots, K_8$ mit einer Länge von jeweils 128 Bit. Hierzu wird die Verschlüsselungsfunktion verwendet.

5.2.4.2 Verschlüsselung

DFC ist ein Typ-1-Feistel-Netzwerk, das auf zwei 64-Bit Blöcken arbeitet. Der 128 Bit große Eingabeblock P wird in zwei Teilblöcke aufgespalten:

$$B_0 || B_1.$$

Für die acht Runden des Feistel-Netzwerkes gilt:

$$B_{i+1} = F_{K_i}(B_i) \oplus B_{i-1}.$$

Der Chiffretext ist dann

$$C = B_9 || B_8.$$

F ist hierbei die Rundenfunktion. In ihr kommen die Addition und Multiplikation sowie die XOR-Verknüpfung zum Einsatz. Desweiteren werden die Modulo-Operationen $\bmod (2^{64} + 13)$ und $\bmod 2^{64}$ und eine S-Box mit 6 Bit als Eingabe und 32 Bit als Ausgabe verwendet. Die S-Box ist fest vorgegeben.

5.2.4.3 Entschlüsselung

Für die Entschlüsselung kann die gleiche Funktion benutzt werden wie zur Verschlüsselung. lediglich die acht Rundenschlüssel müssen in der umgekehrten Reihenfolge angewendet werden.

5.2.4.4 Verwendete Operationen

Dieser Algorithmus verwendet nur wenige Elemente. Addition, Multiplikation, XOR-Verknüpfung und eine S-Box werden in der Rundenfunktion eingesetzt. Diese wird wiederum in der Ver-, als auch in der Entschlüsselungsfunktion eingesetzt. Desweiteren wird die Rundenfunktion für die Erzeugung der Rundenschlüssel benutzt.

5.2.5 E2

E2 [36] stammt von der Nippon Telegraph and Telephone Corporation (Japan). E2 ist ein Typ-1-Feistel-Netzwerk mit 12 Runden.

5.2.5.1 Schlüsselexpansion

Die Schlüsselexpansion von E2 erzeugt aus dem Benutzerschlüssel 16 Rundenschlüssel mit je 128 Bit. Dazu wird die XOR-Verknüpfung, die Anwendung einer S-Box und die Multiplikation eines Vektors, dessen Elemente aus acht Bytes besteht, mit einer konstanten Matrix P, deren Elemente entweder 0 oder 1 sind. Die Addition wird hierbei durch die XOR-Operation ersetzt.

5.2.5.2 Verschlüsselung

Zuerst wird eine Eingangspermutation IP angewandt, die unter anderem zwei Rundenschlüssel benutzt, um die beiden Datenblöcke $L_0||R_0$ zu modifizieren. Danach folgen 12 Runden eines Feistelnetzwerkes mit $i = 1, \ldots, 12$ gilt:

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

Woraufhin eine Ausgangspermutation FP auf $L_{12}||R_{12}$ angewandt wird. Diese ist invers zu IP, wird aber mit anderen Rundenschlüsseln benutzt.

Die Rundenfunktion F benutzt die XOR-Verknüpfung, um den Rundenschlüssel mit dem Datenwort zu verknüpfen sowie eine konstante S-Box, die ein Byte als Eingabe hat und ein Byte als Ausgabe. Diese S-Box kann aber auch durch eine Funktion beschrieben werden. Desweiteren wird die Matrixmultiplikation mit P verwendet, die auch während der Schlüsselexpansion zum Einsatz kommt. Anschließend wird das Datenwort byteweise nach links rotiert.

5.2.5.3 Entschlüsselung

Die Entschlüsselung folgt dem Schema von Definition 12. Lediglich die Eingangs- und Ausgangspermutationen müssen mit den richtigen Rundenschlüsseln benutzt werden.

5.2.5.4 Verwendete Operationen

E2 verwendet vorwiegend Operationen, die auf Byte-Ebene arbeiten: die bitweise XOR-Verknüpfung, eine S-Box, die ein Byte als Ein-, bzw. Ausgabe besitzt, das byteweises Rotieren sowie die Matrixmultiplikation mit einem Vektor, dessen Elemente aus Bytes bestehen.

5.2.6 Frog

Frog [44] ist ein Algorithmus, der von TecApro (Costa Rica) entwickelt wurde. Das Design von Frog unterscheidet sich fundamental von dem herkömmlicher Verschlüsselungsalgorithmen. Die eigentliche Verschlüsselung ist relativ einfach gehalten. Die Komplexität ergibt sich aus dem internen Schlüssel, der mit der Schlüsselexpansion erzeugt wird.

5.2.6.1 Schlüsselexpansion

Frog erzeugt aus dem Benutzerschlüssel, der 5 bis 125 Bytes groß sein kann, acht Rundenschlüssel, die wiederum aus je drei verschiedenen Teilschlüsseln bestehen:

xorBu besteht aus 16 Bytes, die zur XOR-Verknüpfung benutzt werden,

substPermu besteht aus 256 Bytes, die eine S-Box beschreiben und

bombPermu besteht wiederum aus 16 Bytes, die ein anderes Byte in dem zu verschlüsselnden Block referenzieren und daher nur Werte von 0 bis 15 annehmen dürfen.

Damit benötigen die acht Rundenschlüssel zusammen 2304 Bytes an Speicherplatz. Die Rundenschlüssel werden in vier Schritten berechnet.

5.2.6.1.1 Schritt 1 Zuerst wird ein 2304 Byte großes Array mit Kopien des Benutzerschlüssels aufgefüllt und dann mit 251 zufälligen Bytes XOR-verknüpft. Diese Bytes sind aus dem „A Million Random Digits“ der RAND corporation abgeleitet.

5.2.6.1.2 Schritt 2 In diesem Schritt wird sichergestellt, daß das 2304 Byte große Array angemessene Werte enthält, die für die in der Verschlüsselung verwendeten Operationen benutzbar sind.

5.2.6.1.3 Schritt 3 Danach wird mit dem Benutzerschlüssel ein 16 Byte großer Initialisierungsvektor IV initialisiert und sodann ein Array von 2304 Bytes, die jeweils einen Wert von 0 haben, mit dem ebenfalls 2304 Byte großen internen Schlüssel und dem IV im CBC-Modus durch die Frog-Verschlüsselungsfunktion chiffriert. Das Ergebnis ist wiederum 2304 Byte lang.

5.2.6.1.4 Schritt 4 Schließlich muß noch sichergestellt werden, daß das Ergebnis wiederum ein gültiger interner Schlüssel ist. Dies erfolgt auf die gleiche Weise wie in Schritt 2. Das Resultat ist der für die Verschlüsselung benutzte interne Schlüssel, der aus den acht Rundenschlüsseln besteht.

5.2.6.2 Verschlüsselung

Die Verschlüsselung eines 16 Byte großen Datenblockes erfolgt byteweise in vier Schritten. Diese werden im folgenden dargestellt.

5.2.6.2.1 Schritt 1

Das aktuelle Byte des Datenblockes wird mit dem aktuellen Byte von $xorBu$ XOR-verknüpft.

5.2.6.2.2 Schritt 2

Es erfolgt jetzt eine Ersetzung mittels $substPermu$. Das Ergebnis, ist der Wert, der durch das aktuelle Byte referenziert wird. Dies entspricht dem Prinzip einer S-Box.

5.2.6.2.3 Schritt 3

Nun wird das darauffolgende Byte des Datenblockes mit dem bisher berechneten Ergebnis XOR-Verknüpft. Es wird also nicht das aktuelle Byte verändert, sondern das nächste, bzw. das erste Byte des Datenblockes wenn das aktuelle Byte auch das letzte im Datenblock ist.

5.2.6.2.4 Schritt 4

Daraufhin wird mittels des Arrays $bombPermu$ der Index eines weiteren Bytes des Datenblockes ermittelt und dieses Byte mit dem aktuellen Byte XOR-verknüpft.

5.2.6.3 Entschlüsselung

Die Entschlüsselung unterscheidet sich in folgenden Punkten von der Verschlüsselung:

- Die acht Rundenschlüssel müssen in der umgekehrten Reihenfolge verwendet werden.
- Die vier Schritte der Verschlüsselung werden in der umgekehrten Reihenfolge ausgeführt.
- $substPermu$ muß invertiert werden.

5.2.6.4 Verwendete Operationen

Frog verwendet als Hauptelement die XOR-Verknüpfung, die fast überall angewendet wird. Desweiteren wird in jeder Runde des Algorithmuses eine eigene S-Box mit 256 Einträgen benutzt.

5.2.7 HPC

HPC (Hasty Pudding Cipher) [42] von Rich Schroeppel (USA) ist eine Familie von Blockchiffren, die Blöcke und Schlüssel beliebiger Größe verarbeiten kann. Zusätzlich bietet die Chiffre ein weiteres Sicherheitsfeature, das bewirkt, daß die gleichen Klartextblöcke zu unterschiedlichen Chiffretextblöcken verschlüsselt werden, den sogenannten *Spice*.

Insgesamt besteht diese Chiffre aus fünf Subchiffren, die für verschiedene Blockgrößen ausgelegt sind. Hier soll jedoch nur die Subchiffre betrachtet werden, die die vom NIST geforderte Blockgröße von 128 Bit implementiert. HPC arbeitet mit 64 Bit großen Datenwörtern und die arithmetischen Operationen sind deshalb modulo 2^{64} auszuführen.

5.2.7.1 Schlüsselexpansion

Im folgenden bezeichnet $\ll$ die Linksverschiebung und $\gg$ die Rechtsverschiebung eines Datenwortes.

Chiffre	Blockgröße
tiny	≤ 35 Bit
short	36 – 64 Bit
medium	65 – 128 Bit
long	129 – 512 Bit
extended	≥ 513 Bit

Tabelle 5.2: HPC Subchiffren

Zuerst wird eine Schlüsselexpansionstabelle K mit 256 64-Bit Wörtern erstellt:

$$\begin{aligned} K_0 &= PI19 + \text{Subchiffren-Nummer} \\ K_1 &= E19 * \text{Schlüssellänge} \\ K_2 &= R220 \lll \text{Anzahl der Bit der Subchiffre} \\ K_i &= K_{i-1} + (K_{i-2} \oplus K_{i-3} \gg 23 \oplus K_{i-3} \ll 41) \end{aligned}$$

$PI19$, $E19$ und $R220$ sind Konstanten, die aus π, e und $\sqrt{2}$ abgeleitet sind.

Als nächstes werden die ersten 128 Wörter (zu je 64 Bit) des Benutzerschlüssels mit den ersten 128 Wörtern von K XOR-verknüpft.

Danach wird eine Funktion zum Mischen von K benutzt und danach eventuell weitere Wörter des Benutzerschlüssels wieder mit den ersten 128 Wörtern von K XOR-verknüpft. Die Mischfunktion stellt sicher, daß jedes Bit der Schlüsselexpansionstabelle von jedem anderen Bit beeinflußt wird. Hierzu werden Addition, Subtraktion, XOR-, OR- und AND-Verknüpfung sowie Links- und Rechtsverschiebung von 64-Bit Datenwörtern benutzt.

Zusätzlich kann noch ein sogenannter Spice verwendet werden. Zwei gleiche Datenblöcke, die mit dem gleichen Benutzerschlüssel, aber mit unterschiedlichem Spice verschlüsselt werden resultieren in unterschiedlichen Chiffretexten. Der Spice selbst braucht dabei nicht geheim zu bleiben.

5.2.7.2 Verschlüsselung

Zu Beginn der Verschlüsselung wird der Klartext P auf zwei 64 Bit große Datenwörter, S_0 und S_1, aufgeteilt. Danach werden zwei Wörter, die in Abhängigkeit von der Blockgröße gewählt werden, zu S_0 und S_1 addiert. Jetzt folgen 8 Runden mit einer Mischfunktion. In dieser Funktion werden Addition, Subtraktion, XOR- und AND-Verknüpfung sowie Links- und Rechtsverschiebung benutzt. Hierbei werden sowohl Elemente aus K, als auch Elemente des Spice mittels Addition, Subtraktion und XOR-Verknüpfung zu S_0 und S_1 gemischt.

5.2.7.3 Entschlüsselung

Bei der Entschlüsselung werden die acht Runden in der umgekehrten Reihenfolge durchlaufen. Die Schritte in der Mischfunktion sind ebenfalls invertierbar.

5.2.7.4 Verwendete Operationen

HPC verwendet eine Vielzahl von verschiedenen Operationen. Die arithmetischen Funktionen arbeiten jeweils auf 64-Bit Wörtern, also modulo 2^{64}. Dies sind die Addition, Subtraktion und Multiplikation. Eine weitere Gruppe von Operationen arbeitet bitweise. Dies sind XOR-, AND- und OR-Verknüpfung sowie die das Links- und Rechtsverschiebung von Datenwörtern.

5.2.8 Loki97

Dr. Lawrie Brown, Prof. Josef Pieprzyk und Prof. Jennifer Seberry (Australien) entwickelten Loki97 [10] als Nachfolger von Loki, der die Minimalanforderungen des NIST erfüllt.

5.2.8.1 Schlüsselexpansion

Die Schlüsselexpansion errechnet 48 Rundenschlüssel K_i aus dem Benutzerschlüssel. Dazu wird die Rundenfunktion f von Loki97 in Verbindung mit der Addition modulo 2^{64} und die Multiplikation mit einem konstantem Wert modulo 2^{64} benutzt.

5.2.8.2 Verschlüsselung

Loki97 ist ein Typ-1-Feistel-Netzwerk mit 16 Runden. Der Klartext wird in zwei 64 Bit große Blöcke aufgeteilt:

$$P = L_0 || R_0.$$

Danach folgen 16 Runden eines Feistel-Netzwerkes. Die Addition wird modulo 2^{64} ausgeführt.

$$\begin{aligned} R_i &= L_{i-1} \oplus f(R_{i-1} + K_{3i-2}, K_{3i-1}) \\ L_i &= R_{i-1} + K_{3i-2} + K_{3i} \end{aligned}$$

Danach wird der Chiffretext zusammengesetzt:

$$C = R_{16} || L_{16}.$$

Der Aufbau der Rundenfunktion f wird im folgenden beschrieben. Definiert ist diese Funktion folgendermaßen:

$$f(B, K) = Sb(P(Sa(KP(B, K))), K).$$

KP eine Funktion, die in Abhängigkeit von dem aktuellen Rundenschlüssel K entscheidet, ob und welche Bit in B permutiert werden. E ist eine Expansionsfunktion, die 64 Bit als Eingabe und 96 Bit als Ausgabe besitzt. Hierzu werden einige Bit verdoppelt. Sa und Sb bestehen aus der Anwendung zweier S-Boxen und P implementiert eine Bitpermutation mittels einer P-Box.

5.2.8.3 Entschlüsselung

Für die Entschlüsselung kann der gleiche Algorithmus wie zur Verschlüsselung benutzt werden. Die Rundenschlüssel K_i mit $i = 1, \ldots, 48$ müssen in der umgekehrten Reihenfolge benutzt werden und die Rundenschlüssel K_{3i} und K_{3i-2}, die in der Addition benutzt werden, müssen invertiert werden.

5.2.8.4 Verwendete Operationen

Die von Loki97 verwendeten Operationen sind die XOR-Verknüpfung, die Addition und die Multiplikation modulo 2^{64}. In der Permutationsfunktion KP werden die bitweise AND-, OR- und die bitweise Negation verwendet. In den Funktionen Sa und Sb werden zwei konstante S-Boxen benutzt. Die erste S-Box nimmt 13 Bit als Eingabe und die zweite S-Box 11 Bit. Diese beiden S-Boxen lassen sich durch Funktionen beschreiben, die die XOR- und AND-Verknüpfung sowie die Multiplikation, bzw. Exponentiation modulo einem festen Wert als Elemente benutzen. Die Funktion P wird durch eine P-Box implementiert.

5.2.9 Magenta

Magenta [29] ist ein Algorithmus, der von der Deutschen Telekom AG (Deutschland) entwickelt wurde und dort zum Schutz sensitiver Managementdaten eingesetzt wird.

5.2.9.1 Schlüsselexpansion

Der Benutzerschlüssel wird in Teilschlüssel aufgespalten. Die Anzahl dieser ist von der Größe des Benutzerschlüssel abhängig:

128 Bit: $K = K_1||K_2$

192 Bit: $K = K_1||K_2||K_3$

256 Bit: $K = K_1||K_2||K_3||K_4$

Jeder Teilschlüssel ist damit 64 Bit lang.

5.2.9.2 Verschlüsselung

Magenta ist ein Typ-1-Feistel-Netzwerk mit $k = 2$, das mit unterschiedlichen Rundenzahlen in Abhängigkeit von der Schlüsselgröße. Der Eingabeblock wird in zwei Teilblöcke aufgeteilt: $P = L_0||R_0$. Für eine Runde des Feistel-Netzwerkes gilt dann:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus E^{(3)}(R_{i-1}, K_x). \end{aligned}$$

K_x ist dabei ein 64 Bit großer Teil des Benutzerschlüssels. Die folgende Tabelle gibt den zu verwendenden Teilschlüssel für eine Runde in Abhängigkeit von der Größe des Benutzerschlüssels an.

Die Rundenfunktion $E^{(r)}$ hat als Eingabe einen Datenblock mit 128 Bit, also 16 Bytes:

$$E^{(r)}(b_0, \ldots, b_{15}) = C_e^{(r)}$$

$$\begin{aligned} C^{(j+1)}(b_0, \ldots, b_{15}) &= T((b_0, \ldots, b_7) \oplus C_e^{(j)}, (b_8, \ldots, b_{15}) \oplus C_o^{(j)}) \\ C^{(1)} &= T(b_0, \ldots, b_{15}) \end{aligned}$$

Runde	128 Bit	192 Bit	256 Bit
1	K_1	K_1	K_1
2	K_1	K_2	K_2
3	K_2	K_3	K_3
4	K_2	K_3	K_4
5	K_1	K_2	K_4
6	K_1	K_1	K_3
7			K_2
8			K_1

Tabelle 5.3: Verwendete Teilschlüssel von Magenta

$C_e^{(j)}$ enthält die Bytes mit geradem Index, während $C_o^{(j)}$ diejenigen Bytes mit ungeradem Index enthält. Dies sind in jedem Fall acht Bytes, bzw. 64 Bit. Diese Funktion hat die Struktur einer Fast-Hadamard-Transformation [26], die ähnlich wie die Fast-Fourier-Transformation aufgebaut ist.

$$\begin{aligned}
T(b_0,\ldots,b_{15}) &= \Pi(\Pi(\Pi(\Pi(b_0,\ldots,b_{15})))) \\
\Pi(b_0,\ldots,b_{15}) &= (PE(x_0,x_8), PE(x_1,x_9),\ldots,PE(x_7,x_{15})) \\
PE(x,y) &= (A(x,y), A(y,x)) \\
A(x,y) &= f(x \oplus f(x))
\end{aligned}$$

f ist eine Funktion, die ein Byte auf einen anderen Byte-Wert abbildet. Sie kann leicht als Tabelle implementiert werten. Ein Byte wird hierbei als Element von GF(256) betrachtet, d. h. das i-te Bit wird als Koeffizient eines Polynoms mit dem Grad 7 betrachtet. Ein Byte mit 8 Bit wird also beschrieben durch folgendes Polynom:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0.$$

Um sicherzustellen, daß jedes Ergebnisse nur acht Bit benötigt, werden alle Rechnungen modulo dem Polynom

$$p(x) = x^8 + x^6 + x^5 + x^2 + 1$$

getätigt.

Die Rundenfunktion f ist schließlich als

$$f(x) = \begin{cases} 2^x & \text{für } x \neq 255 \\ 0 & \text{für } x = 0 \end{cases}$$

definiert.

5.2.9.3 Entschlüsselung

Für die Entschlüsselungsfunktion D kann die Verschlüsselungsfunktion E eingesetzt werden, wenn die Bytes vor und nach dem Entschlüsseln permutiert werden:

$$D_K(C) = V(E_K(C))$$

mit

$$V(b_0||b_1||\dots||b_{15}) = b_8||b_9||\dots||b_{15}||b_0||\dots||b_7$$

Die b_x sind die 16 Bytes eines 128 Bit großen Datenblockes.

5.2.9.4 Verwendete Operationen

Die grundlegende Operation, die von Magenta verwendet wird ist die XOR-Verknüpfung. Die Funktion f hat als Eingabe nur ein Byte und man kann sie daher effizient als Tabelle implementieren, bzw. im voraus berechnen.

5.2.10 MARS

Mars [11] wurde von einem Team von Kryptographen bei IBM (USA) entwickelt. Dieser Algorithmus hat eine Blockgröße von 128 Bit und kann Schlüsselgrößen von 128 bis 1248 Bit verarbeiten. Bei Mars handelt es sich um ein Typ-3-Feistel-Netzwerk, das auf vier 32-Bit Wörtern arbeitet. Insgesamt besteht dieser Algorithmus aus 3 Phasen. Die erste und letzte Phase dienen dazu, die lineare und differentielle Kryptanalyse sowie Chosen-Ciphertext-Angriffe zu erschweren. In der zweiten Phase erfolgt die eigentliche Verschlüsselung.

5.2.10.1 Schlüsselexpansion

Der Benutzerschlüssel K besteht aus mindestens vier und bis zu 39 32-Bit Wörtern. Diese werden durch die Schlüsselexpansion in 40 Rundenschlüssel $RK_1, RK_2, \dots, RK_{40}$ überführt. Hierzu werden die Rotation von Datenwörtern, die XOR-Verknüpfung, die Addition modulo 2^{32} und die S-Box eingesetzt. Ein Teil der Schlüsselexpansion überprüft die Rundenschlüssel daraufhin, ob sie schwach sind und modifiziert sie in diesem Fall.

5.2.10.2 Verschlüsselung

Hier soll nun die Funktionsweise der 16 Runden des Typ-3-Feistel-Netzwerkes dargestellt werden, das in der zweiten Phase benutzt wird. Die Addition und Multiplikation werden im folgenden immer modulo 2^{32} durchgeführt.

Zuerst werden mit der Funktion $E(B_0, K_{2i+4}, K_{2i+5})$ drei Werte A_1, A_2, A_3 ermittelt, die benutzt werden, um die drei anderen Blöcke zu verändern. i entspricht dabei dem Rundenindex. B_0 wird dann um einen festen Betrag nach links rotiert und B_2 wird um A_2 erhöht.

Danach wird werden die beiden anderen Blöcke in Abhängigkeit von dem aktuellen Rundenindex i verändert. In den ersten acht Runden wird A_1 zu B_1 addiert. B_3 wird mit A_3 XOR-verknüpft. In den letzten acht Runden wird B_3 mit A_1 addiert und B_1 mit A_3 XOR-verknüpft.

Zuletzt werden die Wörter B_0 bis B_3 nach rechts rotiert.

Diese Struktur unterscheidet sich leicht von Definition 14. Ein Datenwort beeinflußt die drei anderen Datenwörter direkt. Dies geschieht über die XOR-Verknüpfung und anders als in der ursprünglichen Definition auch über die Addition. Dazu wird noch zwischen den ersten acht und den letzten acht Runden unterschieden.

5.2.10.3 Entschlüsselung

Nach der Inversion der letzten Phase erfolgt die Entschlüsselung analog zu Definition 14. Daran schließt sich die inverse erste Phase an. Desweiteren werden die verwendeten Operationen durch die entsprechenden inversen Operationen vertauscht, also die Addition durch Subtraktion und die Links- durch die Rechtsrotation ersetzt.

5.2.10.4 Verwendete Operationen

In den 16 Runden der zweiten Phase der Verschlüsselung werden die Addition modulo 2^{32} sowie die XOR-Verknüpfung und die Rotation eines Datenwortes um einen festen Betrag verwendet. Dazu werden drei Werte benutzt, die mittels der E-Funktion berechnet werden. Diese drei Werte hängen von einem Datenblock sowie zwei Rundenschlüsseln ab. In dieser Funktion werden die Addition und Multiplikation modulo 2^{32}, die Rotation um einen festen Betrag sowie die datenabhängige Rotation verwendet. Desweiteren kommt hier die XOR-Verknüpfung und eine S-Box mit 512 Einträgen zum Einsatz.

In der ersten und letzten Phase der Verschlüsselung werden die gleichen Elemente benutzt, jedoch sind die Operationen hierbei nicht von dem benutzten Schlüssel beeinflußt.

5.2.11 RC6

RC6 [38] wurde von Ron Rivest, RSA Data Security (USA), als Nachfolger von RC5 entworfen, der die Bedingungen des NIST erfüllt.

Der Algorithmus ist vollständig parametrisiert, d. h. man kann die Größe des Schlüssels, die Wortgröße des Prozessors und die Anzahl der Runden beliebig angeben. Dies wird folgendermaßen notiert: RC6-$w/r/b$. w ist die Wortbreite des Prozessors in Bit, r ist die Anzahl der Runden und b ist die Länge des Schlüssels in Byte.

5.2.11.1 Schlüsselexpansion

Bei der Schlüsselexpansion wird der b Byte lange Benutzerschlüssel K in $2r+3$ Rundenschlüssel RK_i überführt. Dabei kommen als Operationen die Linksrotation $\lll$, die Addition modulo w und die Multiplikation modulo w zum Einsatz.

5.2.11.2 Verschlüsselung

RC6 arbeitet auf vier w Bit breiten Wörtern. Für 128 Bit große Eingabeblöcke also 32 Bit große Wörter B_1, B_2, B_3, B_4. Für eine Runde der Verschlüsselung gilt dann:

$$\begin{aligned}
B_1^j &= B_2^{j-1} \\
B_2^j &= f(B_3^{j-1}, B_2^{j-1}, B_4^{j-1}, RK_{2j}) \\
B_3^j &= B_4^{j-1} \\
B_4^j &= g(B_1^{j-1}, B_2^{j-1}, B_4^{j-1}, RK_{2j+1})
\end{aligned}$$

In den Rundenfunktionen f und g werden die Linksrotation $\lll$ und die Addition modulo w sowie die XOR-Verknüpfung $\oplus$ benutzt.

RC6 ist also kein klassisches Feistel-Netzwerk, da zwei Blöcke jeweils von den beiden unverändert bleibenden Blöcken und dem Rundenschlüssel beeinflußt werden. Auch werden die zu verändernden Blöcke nicht mit dem Ergebnis der Rundenfunktionen XOR-verknüpft, sondern diese werden direkt in der Rundenfunktion verarbeitet.

5.2.11.3 Entschlüsselung

Um die Verschlüsselung rückgängig zu machen wird in den Rundenfunktionen f^{-1} und g^{-1} die Rechtsrotation benutzt. Desweiteren werden die Rundenschlüssel RK_i in der umgekehrten Reihenfolge verwandt, bzw. der Rundenindex j wird von r bis 1 durchlaufen.

$$\begin{aligned}
B_1^j &= g^{-1}(B_4^{j-1}, B_1^{j-1}, B_3^{j-1}, RK_{2j+1}) \\
B_2^j &= B_1^{j-1} \\
B_3^j &= f^{-1}(B_2^{j-1}, B_1^{j-1}, B_3^{j-1}, RK_{2j}) \\
B_4^j &= B_3^{j-1}
\end{aligned}$$

5.2.11.4 Verwendete Operationen

RC6 verwendet Operationen, die auf Wörtern der Größe von w Bit arbeiten. Bei Eingabeblöcken von 128 Bit entspricht dies $w = 32$. Dabei kommt die Links- und Rechtsrotation von w-Bit-Wörtern und die XOR-Verknüpfung zum Einsatz. Desweiteren wird die Addition, Subtraktion und Multiplikation modulo 2^w verwendet.

5.2.12 Rijndael

Rijndael [15] ist ein Algorithmus, der von Joan Daemen und Vincent Rijmen (Belgien) stammt. Es werden viele Techniken eingesetzt, die auch in Square von J. Daemen, L. R. Knudsen und V. Rijmen stammt [14].

Dieser Algorithmus arbeitet auf einer Matrix von Bytes, dem sogenannten State. Diese Matrix besitzt 4 Zeilen und $b/32$ Spalten, wobei b die Blockgröße ist:

$$\begin{pmatrix}
a_{0,0} & a_{0,1} & \dots \\
a_{1,0} & \ddots & \\
a_{2,0} & & \\
a_{3,0} & & a_{3,b/32}
\end{pmatrix}$$

k sei die Schlüsselgröße und die Anzahl r der Runden ist abhängig von b und k. Dieser Wert liegt zwischen 10 und 14.

5.2.12.1 Schlüsselexpansion

Die Anzahl der Rundenschlüssel ist abhängig von der Anzahl der Runde. Jeder Rundenschlüssel besitzt genausoviele Bit wie der Eingabeblock. Es gibt aber zwei verschiedene Algorithmen, mit denen der Nutzerschlüssel expandiert werden kann. Welcher dieser Algorithmen zum Einsatz kommt, ist abhängig von der Schlüssellänge.

5.2.12.2 Verschlüsselung

Jede Runde der Verschlüsselung besteht aus der Anwendung verschiedener Operationen. Zuerst wird eine S-Box angewandt, die ein Byte des State durch einen anderen Byte-Wert ersetzt. Danach werden in Abhängigkeit von der Blockgröße die drei letzten Zeilen der Matrix a um verschiedene Werte nach rechts rotiert. Daraufhin erfolgt die Transformation aller Spalten, woraufhin der Rundenschlüssel mittels einer XOR-Verknüpfung angewandt wird.

Die letzte Runde des Verschlüsselungsalgorithmuses unterscheidet sich leicht von den vorhergehenden. Bei dieser Runde wird die Spaltentransformation weggelassen, was keinen Einfluß auf die Sicherheit des Algorithmuses hat, aber dafür sorgt, daß Ver- und Entschlüsselung ähnlich aufgebaut sind.

5.2.12.3 Entschlüsselung

Bei der Entschlüsselung werden lediglich die inversen Operationen zu der Verschlüsselung angewandt. Zuerst wird der State mit dem Rundenschlüssel XOR-verknüpft. Sodann die inverse Spaltentransformation und dann die inverse Zeilenrotation ausgeführt. Als letztes wird die inverse S-Box angewandt.

Die erste Runde der Entschlüsselung besitzt keine Inversion der Spaltentransformation, da diese ja in der letzten Runde der Verschlüsselung fehlt.

5.2.12.4 Verwendete Operationen

In diesem Verschlüsselungsalgorithmus werden vor allem Operationen verwandt, die auf einzelnen Bytes arbeiten. Dazu gehören die S-Boxen, die Rotation der Zeilen sowie die XOR-Verknüpfung.

Bytewerte werden als Polynome im Galois-Feld GF(2^8) betrachtet. D. h. daß jedes Bit eines Bytes ein Koeffizient eines Polynoms ist. Z. B. $00101101_2 = x^5 + x^3 + x^2 + 1$. Die Addition zweier Bytes entspricht der Addition der Koeffizienten modulo 2. Oder anders ausgedrückt: die Addition zweier Bytes entspricht der XOR-Verknüpfung. Die Multiplikation zweier Bytes ist als die Multiplikation zweier Polynome modulo einem irreduziblen Polynom 8. Grades definiert. Dies läßt sich durch ein Linksshift und eine XOR-Verknüpfung implementieren.

Die Spaltentransformation ist als Polynommultiplikation mit dem Polynom $3x^3+1x^2+1x+2$ modulo $x^4 + 1$ definiert. Die Koeffizienten sind dabei die 4 Byte einer Spalte der Matrix a, also aus GF(2^8) und somit selbst Polynome. Die Spaltentransformation läßt sich dann als Matrixmultiplikation folgendermaßen schreiben:

$$\begin{pmatrix} a'_0 \\ a'_1 \\ a'_2 \\ a'_3 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

5.2.13 SAFER+

SAFER+ [13] wurde von Prof. James L. Massey, Prof. Gurgen H. Khachatrian und Dr. Melsik K. Kuregian entwickelt und von der Cylink Corporation (USA) als AES-Kandidat vorgeschlagen. Er basiert auf anderen Algorithmen der SAFER-Familie und wurde an die Anforderungen des NIST angepaßt.

5.2.13.1 Schlüsselexpansion

Die Schlüsselexpansion von SAFER+ erzeugt aus dem Benutzerschlüssel K $2r + 1$ Rundenschlüssel, wobei r die Anzahl der Runden angibt. Jeder Rundenschlüssel ist ebenfalls 128 Bit groß.

Die Operationen, die in der Schlüsselexpansion benutzt werden, arbeiten alle auf einzelnen Bytes. Es werden hierbei die XOR-Verknüpfung, die Linksrotation um 3 Bit sowie die Addition modulo 256 benutzt.

5.2.13.2 Verschlüsselung

SAFER+ ist ein SP-Netzwerk, bei dem sich die Anzahl der Runden nach der Schlüsselgröße richtet:

Schlüssellänge	Anzahl der Runden r
128	8
192	12
256	16

In jeder Runde i wird eine Rundenfunktion f angewendet. Diese erhält als Eingabe den 128 Bit großen Datenblock (in Form von 16 Bytes) sowie zwei Rundenschlüssel, die ebenfalls in 16 Byte aufgeteilt werden:

$$B_i = f(B_{i-1}, K_{2i-1}, K_{2i}).$$

SAFER+ arbeitet auf einzelnen Bytes, deshalb soll auch die Funktion f auf Byteebene betrachtet werden:

$$f(B, K, K') = f(b_1, \ldots, b_{16}, k_1, \ldots, k_{16}, k'_1, \ldots, k'_{16})$$

Die Bytes werden in zwei Gruppen unterteilt. Die erste Bytegruppe besteht aus den Bytes mit den Indizes 1, 4, 5, 8, 9, 12, 13 und 16. Die zweite Gruppe enthält die restlichen Bytes mit den Indizes 2, 3, 6, 7, 10, 11, 14 und 15.

Im ersten Schritt werden die Datenbytes der Gruppe 1 mit den korrespondierenden Bytes des Schlüssels K XOR-verknüpft. Die der Gruppe 2 werden mittels der Addition modulo 256 mit den entsprechenden Bytes von K verknüpft.

Danach werden die Bytes der ersten Gruppe mittels einer Exponentiation verändert: $45^{b_j} \bmod 257$, wobei $45^{128} = 0$ gelten soll. Die Bytes der zweiten Gruppe werden logarithmiert: $\log_{45} b_j$. Ist $b_j = 0$, so ist das Ergebnis 128.

Daraufhin werden die Bytes der Gruppe 2 mit den entsprechenden Bytes des Schlüssels K' XOR-verknüpft. Die Bytes der ersten Gruppe werden mit den entsprechenden Bytes von K' mittels der Addition modulo 256 verknüpft.

Die Bytes $b_1, b_2, \ldots, b_{16}$ werden als Vektor von Bytes mit 16 Elementen betrachtet, der dann mit einer konstanten 16×16-Matrix M multipliziert wird. Die verwendete Addition und Multiplikation wird modulo 256 ausgeführt.

Der resultierende Vektor aus 16 Bytes ist das Ergebnis der Funktion f.

Nachdem alle Runden des SP-Netzwerkes durchlaufen wurden wird noch eine Ausgangstransformation auf B_r angewendet. Dabei werden die Bytes der Gruppe 1 mit den korrespondierenden Bytes von K_{2r+1} XOR-verknüpft und die der Gruppe 2 mit den entsprechenden Bytes von K_{2r+1} byteweise addiert.

5.2.13.3 Entschlüsselung

Die Entschlüsselung muß die Verschlüsselung rückgängig machen. Dazu wird erst eine Eingabetransformation ausgeführt, die die Inversion der Eingabetransformation ist. Danach wird in jeder Runde die Inversion f^{-1} der Rundenfunktion f angewendet. Die Rundenschlüssel werden dabei in der umgekehrten Reihenfolge benutzt.

In f^{-1} müssen die inversen Operationen wie in der Funktionen f in der umgekehrten Reihenfolge angewendet werden. Dabei muß die inverse Matrix zu M modulo 256 für die Matrixmultiplikation benutzt werden. Die Logarithmierung und die Exponentiation werden vertauscht.

5.2.13.4 Verwendete Operationen

Dieser Algorithmus verwendet nur wenige Operationen, die fast alle auf Byteebene arbeiten. Dies sind die bitweise XOR-Verknüpfung, Addition und Multiplikation modulo 256 sowie Exponentiation und Logarithmierung. In der Schlüsselexpansion wird noch die Rotation eines Bytes um 3 Bit angewendet.

5.2.14 Serpent

Serpent [3] stammt von Ross Anderson (UK), Eli Biham (Israel) und Lars Knudsen (Norwegen). Dieser Algorithmus ist ein SP-Netzwerk und erlaubt eine sogenannte Bitslice-Implementation, d. h. man kann ihn so implementieren, daß er Gebrauch von Parallelismus bei der Ver- oder Entschlüsselung macht.

5.2.14.1 Schlüsselexpansion

Aus dem Benutzerschlüssel K, der eine Länge von bis zu 256 Bit besitzen kann, werden 33 Rundenschlüssel $K_i, i = 0, \ldots, 32$ abgeleitet, die je 128 Bit groß sind. Diese werden mit Hilfe der S-Boxen und der Permutation IP aus dem Benutzerschlüssel abgeleitet.

5.2.14.2 Verschlüsselung

Zuerst wird eine Permutation IP angewendet. Danach folgen 32 Runden eines SP-Netzwerkes und anschließend eine weiter Permutation FP, die invers zu IP ist. Die letzte Runde des SP-Netzwerkes unterscheidet sich dabei von den anderen Runden. Kurz läßt sich das folgendermaßen darstellen:

$$\begin{aligned} B_0 &= IP(P) \\ B_{i+1} &= R_i(B_i) \\ C &= FP(B_{32}) \end{aligned}$$

Hierbei ist R_i wiefolgt definiert:

$$\begin{aligned} R_i(B) &= L(S_{i \bmod 8}(B \oplus K_i)) && i = 0, \ldots, 30 \\ R_i(B) &= S_{i \bmod 8}(B \oplus K_i) \oplus K_{32} && i = 31 \end{aligned}$$

S_i mit $i = 0, \ldots, 7$ bezeichnet die Anwendung der i-ten S-Box und L ist eine lineare Transformation. Diese arbeitet auf einem 128-Bit Block und transformiert diesen unter Benutzung des entsprechenden Rundenschlüssels.

5.2.14.3 Entschlüsselung

Da Serpent kein Feistel-Netzwerk, sondern ein SP-Netzwerk ist, müssen die Rundenfunktionen R_i umkehrbar sein und in der inversen Reihenfolge benutzt werden. Dies ist durch die verwendeten Operationen gewährleistet. Desweiteren müssen die inversen S-Boxen verwendet werden.

5.2.14.4 Verwendete Operationen

Serpent besitzt 8 S-Boxen mit jeweils 4 Bit als Eingabe und 4 Bit als Ausgabe. Weitere benutzte Operationen sind die XOR-Verknüpfung sowie in der linearen Transformation die Rotation und die Verschiebung von Datenwörtern. Diese Operationen arbeiten jeweils auf Wörtern der Größe von 32 Bit.

5.2.15 Twofish

Twofish [40] von Counterpane Systems (USA) kann Schlüssellängen bis 256 Bit verarbeiten und kann an viele Plattformen angepaßt werden, auch wenn nur eingeschränkte Resourcen zur Verfügung stehen.

5.2.15.1 Schlüsselexpansion

Twofish benötigt 40 Teilschlüssel $K_0, \ldots, K_{39}$ sowie vier S-Boxen, die aus dem Benutzerschlüssel abgeleitet werden. Der Algorithmus zur Ableitung der Rundenschlüssel aus dem Benutzerschlüssel benutzt eine Vielzahl von Operationen: die Matrixmultiplikation mit einer konstanten Matrix, die XOR-Verknüpfung, konstante S-Boxen, die durch eine Rechenforschrift definiert sind, die Multiplikation mit einer MDS Matrix, die Pseudo-Hadamard-Transformation sowie die Links- und Rechtsrotation um einen festen Betrag.

5.2.15.2 Verschlüsselung

Twofish ist ein leicht verändertes Typ-1-Feistel-Netzwerk, das auf zwei 64-Bit Blöcken arbeitet. Der Algorithmus läßt sich aber auch so schreiben, das er Definition 12 entspricht.

B_i^n bezeichne im folgenden den n-ten 32-Bit Datenblock in der i-ten Runde. $B_i^{12} = B_i^1 || B_i^2$ bezeichne die Konkatenation der ersten beiden Blöcke in der i-ten Runde zu einem 64-Bit Block. Entsprechend gilt: $B_i^{34} = B_i^3 || B_i^4$.

Vor und nach der Verschlüsselung erfolgt jeweils ein Whitening (siehe Abschnitt 3.1.8). Dabei werden die vier 32-Bit Blöcke mit den ersten acht Teilschlüsseln XOR-verknüpft.

Für die folgenden 16 Runden des Feistel Netzwerkes gilt:

$$\begin{aligned} B_i^{12} &= F_i(B_{i-1}^{12}) \oplus' B_{i-1}^{34} \\ B_i^{34} &= B_{i-1}^{12} \end{aligned}$$

Der Rundenindex i läuft hierbei von 0 bis 15. Bei der $\oplus'$-Operation wird das linke 32-Bit Wort nach der XOR-Verknüpfung um ein Bit nach rechts rotiert und das rechte 32-Bit Wort (also B_{i-1}^4) vor der XOR-Verknüpfung um ein Bit nach links rotiert.

Die Funktion F_i nimmt als Eingabe den 64-Bit großen linken Eingabeblock B_{i-1}^{12} und berechnet $PHT(g(B_{i-1}^1), g(B_{i-1}^2) \lll 8)$. PHT ist hierbei die Pseudo-Hadamard-Transformation (siehe Abschnitt 3.1.7). Das Ergebnis wird in zwei 32-Bit Blöcke aufgespalten und zu diesen wird K_{2i+8}, bzw. K_{2i+9} addiert. Die Addition wird hierbei modulo 2^{32} ausgeführt.

Die Funktion g wiederum nimmt als Eingabe einen 32-Bit Wert, der in vier Bytes aufgeteilt wird. Auf jedes Byte wird dann eine S-Box mit acht Eingangs- und acht Ausgangsbit angewandt. Diese vier Bytes werden dann als Elemente eines Vektors interpretiert und dieser wird mit der gleichen konstanten 4×4 MDS Matrix (Abschnitt 3.1.6) multipliziert, die in der Schlüsselexpansion benutzt wird. Die vier Elemente des Ergebnisvektors werden als 32-Bit Wert interpretiert und sind somit auch das Ergebnis der Funktion g.

5.2.15.3 Entschlüsselung

Da Twofish ein Typ-1-Feistel-Netzwerk ist, erfolgt die Entschlüsselung nach dem in Definition 12 beschriebenen Prinzip. Dabei ist aber zu beachten, daß die Rotationsoperationen, die in jeder Runde vor, bzw. nach der XOR-Verknüpfung vorgenommen werden kompensiert werden.

5.2.15.4 Verwendete Operationen

Die Funktion g bildet den Kern der Verschlüsselungsroutine. In ihr werden schlüsselabhängige S-Boxen und die Multiplikation mit einer konstanten MDS Matrix benutzt. Danach kommt die PHT-Funktion zum Einsatz. Weitere wichtige kryptographische Elemente sind die XOR-Verknüpfung, die Rotationsfunktionen sowie die Addition.

5.3 Analysen

In diesem Kapitel soll nun dargestellt werden, wie die im letzten Abschnitt beschriebenen Algorithmen bezüglich den Kriterien des NIST bewertet wird. Dies stützt sich auf [35] und verschiedene Untersuchungen, die im Laufe der ersten Runde der Evaluation getätigt wurden. Wichtige Aspekte der Untersuchung sind die Sicherheit, die Kosten sowie Implementationscharakteristiken.

5.3.1 Sicherheit

Wichtige Elemente für die Sicherheit eines Algorithmuses sind die Resistenz gegen Kryptanalysen, die mathematischen Grundlagen, die zufällige Verteilung des Chiffretextes sowie die Sicherheit im Vergleich zu anderen Kandidaten. Das NIST unterscheidet hierzu zwischen generellen Angriffen und implementationsabhängigen Angriffen, die vor allem Smartcards betreffen. Die generellen Angriffe werden in „Major Attacks" und in „Minor Attacks" unterteilt.

5.3.1.1 Major Attacks

Major Attacks sind Angriffe, die schwerwiegende Auswirkungen auf die Sicherheit haben und damit ein starker Indikator dafür sind, daß ein Algorithmus nicht in die zweite Runde der Evaluation übernommen wird. Bei fünf der Kandidatenalgorithmen wurden solche Schwächen gefunden:

DEAL Die Version mit 192-Bit Schlüsseln ist nicht sicherer als diejenige mit 128-Bit Schlüsseln.

Frog In [45] wird festgestellt, daß für Teile des Schlüsselraumes sowohl differentielle, als auch lineare Kryptanalysen möglich sind.

HPC Es wird festgestellt, daß es für einen Teil des Schlüsselraumes äquivalente Schlüssel gibt.

Loki97 Gegen diesen Algorithmus gibt es Angriffe mit 2^{56} Known Ciphertexts und ebensovielen Chosen Ciphertexts.

Magenta In [8] wird ein Chosen Plaintext Angriff mit 2^{64} Klartexten und 2^{64} Verschlüsselungen dargestellt.

Diese Kandidaten gehören auch zu den langsamsten. Dies führte dazu, daß sie nicht in der zweiten Runde vertreten sind.

5.3.1.2 Minor Attacks

Unter Minor Attacks werden Angriffe verstanden, die allein noch kein Ausschlußkriterium für die Auswahl als Standard sind. Algorithmen mit solchen Schwächen sind:

CRYPTON Es existieren 2^{32} Schwache Schlüssel für die Version mit 256 Bit Schlüssellänge und es existiert ein Angriff gegen eine Version mit nur sechs Runden.

DEAL In den Versionen mit 192 und 256 Bit Schlüssellänge besitzen jeweils ein $1/2^{64}$ äquivalente Schlüssel.

DFC Es existiert ein Angriff gegen eine Version mit nur sechs Runden, die 2^{70} Chosen Ciphertexts benötigt.

E2 Gegen reduzierte Varianten dieser Chiffre mit 9, bzw. 10 Runden existieren Angriffe[1].

SAFER+ Die Version mit 256-Bit Schlüsseln erlaubt sowohl einen Related-Key Angriff, als auch einen Meet-In-The-Middle-Angriff.

5.3.1.3 Sicherheitsabstand

Als weiteres Kriterium, um die Sicherheit zu beurteilen, wurde die Anzahl der Runden vorgeschlagen, die ein Algorithmus benutzt [5]. Dies führt aber zu verschiedenen Problemen. Zum einen hängt die Sicherheit noch von anderen Gesichtspunkten ab. Hier ist z. B. die Rundenfunktion zu nennen. Ist diese schwach, so benötigt der Algorithmus mehr Runden. Desweiteren lassen sich mit diesem Kriterium Algorithmen mit verschiedenen Strukturen nur schlecht vergleichen. Bei einem SP-Netzwerk wird pro Runde der gesamte Datenblock verändert. Bei einem Feistel-Netzwerk dagegen nur ein Teil des Datenblockes. DES verändert pro Runde z. B. nur die Hälfte der Daten. Eine Runde eines SP-Netzwerkes entspricht also in etwa zwei DES-Runden.

Diese Problematik hat schließlich zum Begriff des Sicherheitsabstandes *(engl. Security Margin)* geführt. Hierunter versteht man die Differenz zwischen der Anzahl der verwendeten Runden und der minimal notwendigen Anzahl von Runden, bei der kein (bekannter) Angriff mehr zum Erfolg führt. Auch dieses Kriterium ist nicht ohne Probleme. Es ist ja möglich, daß

[1] In einem Zusatz zu [35] wird richtiggestellt, daß sich diese Angriffe gegen eine modifizierte Form des Algorithmuses richten und nicht gegen die Originalversion.

sich der Sicherheitsabstand in Zukunft verringert, wenn ein neuer Angriff gegen einen Algorithmus bekannt wird, so daß sich die Relationen zwischen den verschieden Kandidatenalgorithmen verschieben können. Dennoch vergrößert ein hoher Sicherheitsabstand das Vertrauen in die Sicherheit einer Chiffre, insbesonders unter dem Gesichtspunkt, daß differentielle und lineare Kryptanalyse direkt von der Anzahl der Runden abhängen und ab einer bestimmten Rundenzahl nicht mehr möglich sind.

Es existieren verschiedene Möglichkeiten, die untere Grenze für den Sicherheitsabstand zu bestimmen:

1. man benutzt die größte Anzahl der Runden, für die ein Angriff durchführbar ist (1 in der Tabelle) oder

2. man benutzt die geschätzte Anzahl von Runden, für die kein Angriff mehr möglich ist (2 in der Tabelle).

Letzteres führt zu geringeren oder sogar negativen Werten für den Sicherheitsabstand, da diese Zahl durchaus auch größer sein kann, als die Anzahl der Runden, die der Algorithmus tatsächlich benutzt. Im folgenden werden die Werte für beide Möglichkeiten aus [35] wiedergegeben. Die R-Spalten enthalten die Anzahl der Runden und die S-Spalten den daraus resultierenden Sicherheitsabstand.

Algorithmus	Runden	1. R.	1. S.	2. R.	2. S.
CAST-256	48	20	140%	40	20%
CRYPTON	12	10	20%	12	9%
DEAL	6	6	0%	10	-40%
DFC	8	6	33%	9	-11%
E2	12	9	33%	10	20%
FROG	8				
HPC	8				
Loki97	16	16	0%	38	-56%
Magenta	6	6	0%	11	-44%
Mars	32	12	166%	20	60%
RC6	20	16	25%	21	-5%
Rijndael	10	6	66%	8	25%
SAFER+	8	5	66%	7	14%
Serpent	32	15	113%	17	88%
Twofish	16	6	166%	14	14%

Tabelle 5.4: Sicherheitsabstand

Nach der ersten Methode haben also folgende Algorithmen den höchsten Sicherheitsabstand:

1. Mars und Twofish,

2. CAST-256,

3. Serpent,

4. Rijndael und Safer+.

Nach der zweiten Methode ändert sich diese Reihenfolge jedoch folgendermaßen:

1. Serpent,
2. Mars,
3. Rijndael,
4. CAST-256 und E2.

Die Bewertung der Sicherheit eines Algorithmuses im Verhältnis zu anderen hängt also nicht unerheblich von der Wahl der Methode und der Schätzung der benötigten Rundenzahl ab.

5.3.1.4 Sicherheit bei Smartcardimplementationen

Implementationen für Smartcards haben besondere Auswirkungen auf die Sicherheit. Hier sind Timing-Angriffe *(engl. Timing Attacks)* und Power-Angriffe *(engl. Power Attacks)* zu nennen [16]. Bei Timing-Angriffen wird darauf zurückgegriffen, daß die Durchführung einer Operation mit unterschiedlichen Operanden unterschiedliche Zeiträume benötigen kann. Operationen gegen die sich dieser Angriff richtet sind z. B. Shift- oder Rotate-Operationen mit variablen Argumenten, Multiplikation und Exponentiation, aber auch die Addition und Subtraktion.

Power-Angriffe basieren auf dem gleichen Prinzip, jedoch wird hierbei der Stromverbrauch während der Verschlüsselung gemessen. Zu den gegenüber diesen Angriffen anfälligen Befehlen gehören aber auch logische Operationen, Shift- und Rotationsoperationen mit konstanten Operanden und sogar Zugriffe auf Tabellen.

In [35] werden die Algorithmen in drei Gruppen unterteilt. Einfach gegen diese Angriffe zu verteidigen sind CRYPTON, DEAL, Magenta, Rijndael und Serpent. Die zweite Gruppe besteht aus FROG, Loki97, SAFER+ und Twofish. Die letzte Gruppe von Algorithmen benutzt z. B. Multiplikationen und Rotationen mit variablen Argumenten. Zu dieser Gruppe gehören CAST-256, DFC, E2, HPC, MARS und RC6.

Die Problematik der Timing- und Power-Angriffe ist kein primäres Kriterium für die Wahl eines Algorithmuses, da hier nur ein Teilgebiet der Sicherheit betroffen. Desweiteren lassen sich Gegenmaßnahmen ergreifen, um diese Angriffe zu verhindern oder zumindest um diese zu erschweren.

5.3.2 Kosten

Weitere Analysen betreffen die Kosten eines Algorithmuses. Hierunter fallen eventuelle Lizenzen, die Performance und der notwendige Speicher.

5.3.2.1 Performance

Die Performance eines Algorithmuses ist von vielen verschiedenen Aspekten abhängig. Dazu gehören unter anderem die Größe der Datenwörter, mit denen ein Algorithmus arbeitet, welche Operationen er benutzt oder wie viele Runden er benötigt.

Der AES muß auf vielen verschieden Plattformen effizient einzusetzen sein. Smartcards mit 8-Bit Prozessoren werden auch in Zukunft noch vielfach anzutreffen sein. 32-Bit Prozessoren und in Zukunft auch 64-Bit Prozessoren werden in PCs eingesetzt. In [20] werden Ergebnisse für die NIST Referenzplattform dargestellt. [28], [4] und [22] stellen Ergebnisse für weitere Plattformen vor. In [41] werden ebenfalls für verschiedene Prozessoren Performancezahlen angegeben. Desweiteren werden hier auch Einschätzungen für Hardwareimplementationen gegeben. Für einen 8-Bit Prozessor werden in [24] Zahlen aufgezeigt. Die Performance der Java-Implementationen wird in [37] untersucht. Dabei werden verschiedene Werte wie z. B. der Durchsatz für verschiedene Verschlüsselungsmodi gemessen. Diese Werte wurden auf verschiedenen Plattformen mit unterschiedlichen Java-Virtual-Machines und auch mit Just-In-Time-Compilern gemessen.

Das NIST teilt die Kandidatenalgorithmen gemäß ihrer Effizienz in fünf Klassen ein. Auf allen Plattformen gleichmäßig schnell sind Rijndael und Twofish. In die Kategorie mit mittlerer Performance fallen CAST-256, CRYPTON, E2 und Serpent. Langsam sind DEAL, FROG, Loki97, Magenta und SAFER+. Die Effizienz von Mars und RC6 ist abhängig von der Verfügbarkeit bestimmter Operationen auf der Plattform. In die fünfte Klasse fallen schließlich DFC und HPC, die mit 64-Bit Datenwörtern arbeiten.

Schließlich kann man feststellen, daß ein absoluter Vergleich der Leistungsfähigkeit objektiv nicht zu machen ist. Verschiedenen Algorithmen lassen sich auf unterschiedlichen Prozessortypen in unterschiedlichem Maße optimieren. Ein weiterer Gesichtspunkt ist, daß ein schneller Algorithmus nicht unbedingt sicher ist, bzw. eine sichere Variante, dann auch langsamer wäre. Die Performance eines Algorithmuses ist deshalb auch kein primäres Kriterium für eine Auswahl, aber dennoch ein wichtiger Faktor für eine Entscheidung.

5.3.2.2 Speicherplatzanforderungen

Auch die Speicherplatzanforderungen, die ein Algorithmus an ein System stellt sind ein Faktor für die Kosten. Hier sind sowohl die Codegröße der Ver- und Entschlüsselungsroutine, als auch das zum Speichern der Teilschlüssel notwendige RAM. Hierbei gibt es verschiedene Möglichkeiten, die Speicherplatzanforderungen mit der Geschwindigkeit auszutarieren. Einige Algorithmen verwenden zur Ver- und Entschlüsselung die gleiche Routine, wobei sich in diesem Fall die Schlüsselexpansion ändert. Dies wirkt sich sowohl auf die Codegröße, als auch auf die Implementation in Hardware aus. Andere Algorithmen erlauben die Erzeugung der Teilschlüssel „on the fly“. Diese Schlüssel werden also erst dann aus dem Benutzerschlüssel berechnet, wenn sie auch benutzt werden. Hierbei entfällt dann ein großer Teil des notwendigen Speichers.

Tabelle 5.5 gibt die Werte aus [35] wieder, die größtenteils aus [41] entnommen sind. Es wird hier der Speicherverbrauch für das Vorhalten der Teilschlüssel angegeben. Dabei wird wenn möglich der Wert für die Version angegeben, die die Schlüssel nur berechnet, wenn diese auch benötigt werden. Dies ist für CAST-256, Crypton, DEAL, DFC, Mars, Rijndael, SAFER+, Serpent und Twofish der Fall. Die dritte Spalte gibt schließlich den Speicherplatz an, der für die Verschlüsselungsroutine vorgehalten werden muß. Diese kann im ROM abgelegt werden. Allerdings kommt zu diesem Wert eventuell noch weiterer Speicher für die Entschlüsselungsfunktion.

5.3.3 Implementationscharakteristiken

In die dritte Kategorie fallen Eigenschaften wie die Flexibilität eines Algorithmuses, die Implementierbarkeit in Soft- und Hardware sowie die Komplexität, bzw. dessen Einfachheit.

Algorithmus	RAM	ROM
CAST-256	60	6000
Crypton	52	2000
DEAL	50	2000
DFC	100	2000
E2	300	
FROG	2300	
HPC	2000	
Loki97	400	8000
Magenta		
Mars	195	3000
RC6	210	1000
Rijndael	52	1000
SAFER+	50	2000
Serpent	50	1000
Twofish	60	1400

Tabelle 5.5: Speicherplatzanforderungen

5.3.3.1 Eignung für den Einsatz in Smartcards

Die Eignung eines Algorithmuses für eine Smartcardimplementationen ist von vielen Kriterien abhängig. Hierunter fallen besondere Aspekte der Sicherheit, die in 5.3.1.4 diskutiert wurden. Ganz entscheidend für die Möglichkeit einer Smartcardimplementation, ist die Größe des benötigten RAMs, da heutige Smartcards nur wenige hundert Bytes an Speicher besitzen. Das NIST geht in [35] von 256 Bytes an RAM aus, wovon unter Umständen aber ein Teil schon von anderen Applikationen verwendet werden kann. Ideal wäre also ein Speicherverbrauch von nicht mehr als 128 Byte. Tabelle 5.5 zeigt, daß einige Algorithmen diese Grenze deutlich übersteigen und somit nicht mehr auf Smartcards lauffähig sind. Dies sind FROG, HPC und Loki97. Auf High-End-Smartcards lauffähig sind CAST-256, E2, Mars und RC6. Für den Einsatz in Low-End-Smartcards eignen sich schließlich auch Crypton, DEAL, DFC, Magenta, Rijndael, SAFER+, Serpent und Twofish. Hierbei geht das NIST für Low-End-Karten von 2000 Bytes ROM und bei High-End-Karten von 6000 Bytes ROM aus, in denen der Programmcode abgelegt werden kann.

5.3.3.2 NIST-API

In [21] werden die Referenzimplementationen sowie die optimierten C-Implementationen hinsichtlich der Einhaltung des vom NIST vorgegebenen APIs untersucht. Dabei wurde festgestellt, daß viele der Implementationen in unterschiedlichen Bereichen Probleme haben. Dies betrifft die Byte-Order, die ein Algorithmus voraussetzt, die Interpretation von Funktionsargumenten, die Rückgabewerte von Funktionen oder gar die Notwendigkeit von zusätzlichen Funktionsargumenten z. B. um die Blocklänge anzugeben. Die Portabilität einer Implementation leidet vor allem unter der Nicht-Berücksichtigung des ersten Punktes.

5.4 Auswahl des NIST für die zweite Runde der Standardisierung

Das NIST hat fünf Algorithmen in die nähere Auswahl aufgenommen. Dies sind Mars, RC6, Rijndael, Serpent, Twofish. Im folgenden soll die Bewertung dieser Algorithmen durch das NIST dargestellt werden.

5.4.1 Mars

Mars benutzt eine Vielzahl von verschiedenen Operationen. Der Algorithmus ist generell schnell, insbesonders aber auf Plattformen, die Multiplikation und Rotationsoperationen effizient zur Verfügung stellen. Für Mars wurde eine Veränderung vorgeschlagen, die eine On-The-Fly-Erzeugung der Teilschlüssel ermöglicht und somit den Einsatz in Smartcardumgebungen verbessert.

5.4.2 RC6

RC6 sticht durch seine Einfachheit hervor und sollte einfach und effizient in Soft- und Hardware implementierbar sein. Es wird erwartet, daß diese Einfachheit auch der weiteren Analyse zugutekommt. RC6 ist eine Weiterentwicklung von RC5, das aus den gleichen Operationen und Strukturen aufgebaut ist. Die Erkenntnisse aus der Kryptanalyse von RC5 können somit auch in gewissen Maße bei der Analyse von RC6 verwendet werden. Allgemein ist dieser Algorithmus schnell. Auf Plattformen, die Rotationsoperationen zur Verfügung stellen ist er aber besonders schnell. Auch die Schlüsselexpansion ist sehr effizient.

5.4.3 Rijndael

Die Geschwindigkeit dieses Algorithmuses sowie der Schlüsselexpansion ist auf allen Plattformen sehr gut. Es wird erwartet, daß dies auch für Smartcardimplementationen sowie Hardwareimplementationen zutrifft. Desweiteren wird erwartet, daß durch die Implementation von Parallelismus noch weitere Effizienzgewinne zu erzielen sind. Die verwendeten Operation sind sehr konservativ ausgewählt, was der Analyse entgegenkommt und auch Gegenmaßnahmen für bestimmte Angriffe erleichtert.

5.4.4 Serpent

Das hervorstechendste Merkmal von Serpent ist der sehr hohe Sicherheitsabstand: der Algorithmus verwendet doppelt so viele Runden, wie notwendig sind, um aktuelle Angriffe abzuwehren. Dies hat allerdings zur Folge, daß Serpent der langsamste der fünf ausgewählten Algorithmen ist. Dies kann durch spezielle Bitslice-Implementationen abgemildert werden. Serpent ist desweiteren gut an Smartcard- und Hardwareimplementationen anpaßbar. Die konservative Auswahl der verwendeten Operationen und das klare Design ermöglichen eine einfache Analyse und Abwehrmaßnahmen gegen Angriffe auf Smartcardimplemenationen.

5.4.5 Twofish

Twofish hat eine gute Performance auf fast allen Plattformen und sollte ebenfalls gut für Smartcard- und Hardwareimplementationen geeignet sein. Ein Feature dieses Algorithmuses sind die variablen S-Boxen, die vom Benutzerschlüssel abhängen. Desweiteren ermöglicht Twofish verschiedene Optimierungen in Hinblick auf Geschwindigkeit, Speicher, Codegröße oder Platz bei Hardwareimplementationen.

5.4.6 Ausgeschiedene Algorithmen

Die Algorithmen gegen die es Major Attacks gab sind auch die langsamsten, so daß sie nicht in die zweite Runde der Evaluation übernommen wurden. Im folgenden sollen daher nur die kurze Bewertungen von CAST-256, Crypton, DFC, E2 und Safer+ dargestellt werden.

5.4.6.1 CAST-256

CAST-256 favorisiert die Sicherheit gegenüber der Geschwindigkeit, was dazu führt, daß die Performance nur mittelmäßig ist. Desweiteren ist der benötigte ROM-Speicher relativ hoch. Dies führte dazu, daß Serpent diesem Algorithmus vorgezogen wurde.

5.4.6.2 Crypton

Crypton fällt in die gleiche Klasse von Algorithmen wie Rijndael und Twofish, hat aber einen geringeren Sicherheitsabstand.

5.4.6.3 DFC

DFC besitzt einen relativ geringen Sicherheitsabstand. Desweiteren benutzt dieser Algorithmus 64-Bit Multiplikationen, wodurch er auf Prozessoren mit geringerer Wortgröße Performanceeinbußen hinnehmen muß.

5.4.6.4 E2

Auch E2 gehört zur gleichen Klasse wie Rijndael und Twofish, besitzt aber einen geringeren Sicherheitsabstand als diese. Die Performance dieser beiden Algorithmen ist auch besser als die von E2. Dazu kommen noch hohe Anforderungen an RAM und ROM.

5.4.6.5 Safer+

Safer+ gehört zu den langsamsten Algorithmen, hat jedoch im Gegensatz zu Serpent keinen sehr hohen Sicherheitsabstand.

Kapitel 6

Ausblick

Im April 2000 soll die dritte AES-Konferenz in New York abgehalten werden. Dort werden vor allem Sicherheits- und Performanceaspekte diskutiert, die vor allem die Varianten mit 192 und 256 Bit betreffen. Ein weiterer Schwerpunkt wird die Auswertung von Möglichkeiten zur Hardwareimplementation sein. Ebenso werden die Analysen der Sicherheit fortgesetzt.

Die fünf Finalisten decken ein breites Spektrum ab, das von einem als sehr sicher angesehenen, aber langsamen Algorithmus (Serpent) bis hin zu schnellen und vielseitig anpaßbaren Algorithmen (Twofish, Rijndael) reicht. Alle diese Kandidaten eignen sich auch für den Einsatz in Low-End-Systemen, wie z. B. Smartcards.

Die Entscheidung des NISTs für einen Algorithmus – oder vieleich auch mehrere – wird auch Einfluß auf weitere Standardisierungsgremien haben, die diesen Algorithmus übernehmen werden. Es ist zu erwarten, daß der AES in einigen Jahren in vielen Anwendungen eingesetzt wird. Die Anforderungen, die das NIST an einen Standardalgorithmus gestellt hat, berücksichtigen dies schon, indem Wert auf die Möglichkeit von Smartcard- und Hardwareimplementationen gelegt wird.

Die im Vergleich zu DES größere Schlüssellänge, die doppelt so hohe Blocklänge und die Sicherheitsanalysen während des Auswahlprozesses werden die Sicherheit von vielen Anwendungen verbessern.[1] Dies ist insbesonders für viele Anwendungen aus dem Bereich des E-Commerce und des Finanzwesens interessant, wo ein hoher Grad an Sicherheit erforderlich ist.

[1] Dies gilt insbesonders, da die U.S.-Regierung auch entschieden hat, den Export von kryptographischen Produkten zu erleichtern. Der Quellcode der fünf Finalisten ist inzwischen auch über die Web-Seite des NISTs verfügbar.

Kapitel 7

Literaturverzeichnis

[1] Submitted algorithms which were not selected for Round1 of the AES Development Effort.

[2] *Description of Known Answer Tests and Monte Carlo Tests for Advanced Encryption Standard (AES) candidate Algorithm Submissions*, February 1998.

[3] R. Anderson, E. Biham, and L. Knudsen. *Serpent: A Proposal for the Advanced Encryption Standard.*

[4] O. Baudron, H. Gilbert, L. Granboulan, H. Handschuh, A. Joux, P. Nguyen, F. Noilhan, F. Pointcheval, T. Pornin, G. Poupard, J. Stern, and S. Vaudenay. *Report on the AES Candidates.* Ecole Normale Supérieure (CNRS).

[5] E. Biham. *A Note on Comparing the AES Candidates.* Computer Science Department, Technion – Israel Institute of Technology.

[6] E. Biham. Design Tradeoffs of the AES Candidates. Presentation at ASIACRYPT '98.

[7] E. Biham. Cryptanalysis of Multiple Modes of Operation. Technical Report CS0833, Computer Science Department, Technion – Israel Institute of Technology, 1994.

[8] E. Biham, A. Biryukov, L. R. Knudsen, B. Schneier, and A. Shamir. *Cryptanalysis of Magenta*, 1998.

[9] E. Biham and A. Shamir. Differential Cryptanalysis of the full 16-round DES. Technical Report CS0708, Computer Science Department, Technion – Israel Institute of Technology and Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, 1991.

[10] L. Brown and J. Pieprzyk. *Introducing the new LOKI97 Block Cipher*, 1998.

[11] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, J. Stephen M. Matyas, L. O'Connor, M. Peyravian, D. Safford, and N. Zunic. *MARS – a candidate cipher for AES.* IBM Corporation, 1998.

[12] D. Coppersmith. The Data Encryption Standard (DES) and its Strength Against Attacks. Research report, IBM Research Division, December 1992.

[13] Cylink Corporation. *Nomination of SAFER+ as Candidate Algorithm for the Advanced Enryption Standard (AES)*, 1998.

[14] J. Daemen, L. Knudsen, and V. Rijmen. The Block Cipher Square.

[15] J. Daemen and V. Rijmen. *AES Proposal: Rijndael*.

[16] J. Daemen and V. Rijmen. *Resistance Against Implementation Attacks – A Comparative Study of the AES Proposals*, 1999.

[17] EFF – Electronic Frontier Foundation. *Cracking DES – Secrets of Encryption Research, Wiretap Politics & Chip Design*. O'Reilly & Associates, Inc., 1998.

[18] Entrust Technologies, Inc. *CAST-256 Algorithm Specification*.

[19] H. Gilbert, M. Girault, P. Hoogvorst, F. Noilhan, T. Pornin, G. Poubard, J. Stern, and S. Vaudenay. *Decorrelated Fast Cipher: an AES Candidate*. Centre National pour la Recherche Scientifique (CNRS), 1998.

[20] D. B. Gladman. *Implementation Experience with AES Candidate Algorithms*.

[21] L. Granboulan. *AES: Analysis of the RefCode and OptCode submissions*, April 1999.

[22] G. Graunke. *Yet Another Performance Analysis of the AES Candidates*.

[23] B. S. Kaliski Jr. and Y. L. Yin. On the Security of the RC5 Encryption Algorithm. Technical Report TR-602, RSA Laboratories, September 1998.

[24] G. Keating. *Performance Analysis of AES candidates on the 6805 CPU core*, April 1999.

[25] L. R. Knudsen. *DEAL – A 128-bit Block Cipher*, 1998.

[26] S. Y. Kung. *VLSI Array Processors*. Prentice Hall, 1988.

[27] C. H. Lim. *CRYPTON: A New 128-bit Block Cipher*. Information and Communication Reasearch Center, Future Systems, Inc.

[28] H. Lipmaa. *AES Candidates: A Servey of Implementations*.

[29] J. M. J. Jacobson and K. Huber. *The MAGENTA Block Cipher*. Deutsche Telekom AG, 1998.

[30] M. Matsui. Linear Cryptanalysis Method for DES Cipher. In *Abstracts of EUROCRYPT'93*, pages W112–W123, May 1993.

[31] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[32] National Bureau of Standards. *Data Encryption Standard*. U.S. Department of Commerce, January 1977. FIPS PUB 46.

[33] National Institute of Standards and Technology. *DES Modes of Operation*. U.S. Department of Commerce, December 1980. FIPS PUB 81.

[34] National Institute of Standards and Technology, U.S. Department of Commerce. Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES). Federal Register, Volume 62, Number 177, Pages 48051-48058, September 1997.

[35] J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti, and E. Roback. *Status Report on the First Round of the Development of the Advanced Encryption Standard*. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology.

[36] Nippon Telegraph and Telephone Corporation (NTT). *Specification of E2 – a 128-bit Block Cipher*, 1998.

[37] NTT Laboratories. *Java Performance of AES Candidates*.

[38] R. L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. l. Yin. *The RC6 Block Cipher*. RSA Data Security, Inc.

[39] B. Schneier. *Angewandte Kryptographie*. Addison-Wesley, 1. edition, 1996.

[40] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. *Twofish: A 128-Bit Block Cipher*. Counterpane Systems, 1998.

[41] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. *Performance Comparison of the AES Submissions*, 1999.

[42] R. Schroeppel. *An Overview of the Hasty Pudding Cipher*, 1998.

[43] C. E. Shannon. A Mathematical Theory of Communication. Technical report, Bell System, 1948. Reprint with corrections from the *Bell System Technical Journal*, Vol. 27, pp. 379–423, 623–656.

[44] TecApro Internacional S.A. *FROG*.

[45] D. Wagner, N. Ferguson, and B. Schneier. *Cryptanalysis of FROG*, 1998.

[46] Y. Zheng, T. Matsumoto, and H. Imai. On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. In *Advances in Cryptology — CRYPTO '89*, number 435 in Lecture Notes in Computer Science, pages 461–480. Springer-Verlag, 1989. Extended Abstract.

Viele Unterlagen finden sich auf dem Server des NIST unter `http://www.nist.gov/aes`, darunter die Bewerbungsunterlagen und Dokumentationen der verschiedenen Kandidatenalgorithmen sowie viele Diskussionspapiere. Unter `http://aes.nist.gov/aes` finden sich Diskussionsforen zu den einzelnen Algorithmen.

Weitere Resourcen sind auf der „Block Cipher Lounge“ von Lars Knudsen und Vincent Rijmen zu finden: `http://www.ii.uib.no/~larsr/aes.html`.

Implementationen und Vergleiche zu der Effizienz der Kandidaten finden sich auf der Homepage von Brian Gladman: `http://www.btinternet.com/~brian.gladman/cryptography_technology/aes`.

Die Crypto Group des UCL Microelectronics Laboratory stellt unter `http://www.dice.ucl.ac.be/crypto/CAESAR/caesar.html` weitere Informationen zur Verfügung.