Skriptum zur VL

Grundlagen der IT-Sicherheit und Kryptographie

Ao. Univ.Prof. Dr. Andreas Uhl

RIST++ und Institut für Scientific Computing

Universität Salzburg

Adresse:

Andreas Uhl
Institut für Scientific Computing
Jakob-Haringerstr.2
A-5020 Salzburg
Österreich
Tel.: ++43/(0)662/8044/6303
Fax: ++43/(0)662/8044/172
E-mail: uhl@cosy.sbg.ac.at

Contents

Chapter 1

Einleitung

Die Fachsprache in der Kryptographie ist Englisch. Daher hat sich auch eine englische TErminologie eingebürgert, die ich in diesem Skriptum verwenden möchte. Sie finden daher immer wieder englische Ausdrücke in deutschen Sätzen. Das liest sich hin und wieder vieleicht etwas eigenartig. Dafür sind Sie mit den Fachausdrücken vertraut und können so einfacher auch englische Fachliteratur lesen.

Literatur:

- B. Schneier: Applied Cyptography
- W. Stallings: Cryptography and Network Security
- Ch. P. Pfleeger: Security in Computing

1.1 Politik und Geschichte

Kryptographie wird seit Jahrtausenden benutzt, um mitlitärische und diplomatische Kommunikation zu schützen. Diese Tatsache führte zu der Annahme, Kryptographie sei ein Vorrecht der Regierung. Die meisten Regierungen üben heute eine Kontrolle auf die kryptographischen Techniken - wenn nicht sogar auf die Forschung in diesem Bereich - aus. Ein Beispiel dafür sind die Export/Import Bestimmungen der USA für kryptographische Geräte. Diese entsprechen dem Gesetz für Waffenexport.

Dawn of Information Age: Bedarf nach Kryptographie im privaten Bereich.

Die Wissenschaft des "Code knackens" nennt sich Kryptoanalyse und ihr Bedarf ist nicht sehr akzeptiert.

1929: Black Chamber H.L. Stimson (U.S. secretary of state) “gentlemen do not read each others mail´´. Die kodierten diplomatischen Kanäle wurden routinemäßig gebrochen.

1940 wiedereröffnet, um die japanischen Verschlüsselungssysteme zu knacken.

Meilenstein: Shamir bricht Merkle-Hellman Trapdoor-Knapsack Public Key Kryptosystem.

Die Anzahl der ForscherInnen in geheimen Labors ist um vieles größer als die Anzahl der Beteiligten in der öffentlichen Forschung. Diesen offenen Bereich gibt es erst seit etwa 15 Jahren. Es ergibt sich hier folgendes Problem: Kann bzw. soll ein(e) ForscherIn aus dem universitären Umfeld neue Erkenntnisse publizieren, die der eigenen Regierung schaden können , z.B. nicht knackbare Systeme?

In den USA soll die kryptographische Forschung streng reglementiert werden: Regierungsstellen müssen geheime NSA Verfahren verwenden. Diese Systeme werden auch privaten Stellen zur Verfügung gestellt. Wenn sich dies durchsetzen sollte, bleibt die offene Forschung auf internationale Systeme beschränkt.

Die Kryptographie kann in verschiedenen Epochen dargestellt werden.

inf → **1949:** Diese Zeit kann als präwissenschaftliche Epoche bezeichnet werden und die Kryptographie ist eher eine Kunst denn eine Wissenschaft.

Julius Cäsar etwa verwendete folgendes System, mit z=3:

$$CAESAR \rightarrow FDHVDU$$

$$y = x \oplus z \qquad x.....A = 0, B = 1, ...z = 25$$

$$z \text{geheimer Schlüssel}$$

$$y \text{verschlüsselter Buchstabe}$$

$$\oplus \text{Addition Modulo 26}$$

Augustus verwendete das selbe System mit z=4.

In der folgenden Zeit war die Kryptoanalyse der Kryptographie bei weitem überlegen.

1926 G.S.Vernam entwickelte ein neues System. Jedes Bit eines Textes x wird mit einem eigenem Schlüsselbit z verschlüsselt. Der codierte Text $y = x \oplus z$ (mit $\oplus$Addition Modulo 2 ist so nicht knackbar. Die Größe des Schlüssels entspricht allerdings der Größe des Textes.

1949: Im zweiten Weltkrieg wurden erstmals MathematikerInnen in die Forschung mit einbezogen, z.B. Turing in England, der das deutsche System Enigma knackt.

C.E. Shannon veröffentlicht "Communication Theory of Secrecy Systems´´. Darin beweist er etwa die Unberechenbarkeit des Vernam Verfahrens. Dies hatte allerdings noch keinen großen Einfluß auf die wissenschaftliche Gemeinschaft.

1976: W. Diffie und M.E. Hellman veröfftenlichen "New Directions in Cryptography´´. Dies enthält erstmals sogenannte Public Key Kryptographie. Dies sind Verfahren, die eine geheime Kommunikation ohne Transfer des Schlüssels zwischen Sender und Empfänger ermöglichen.

1.2 Terminologie

In diesem Kapitel möchte ich Sie mit den gängigen und im folgenden immer wieder verwendeten Begriffen vertraut machen.

1.2.1 Einführende Begriffe

Sender -Empfänger: Ein Sender möchte eine Nachricht so senden, daß ein potentieller Lauscher diese nicht lesen kann. Eine Nachricht ist ein Text, deren Inhalt zugänglich ist und heißt auch Plain Text oder Clear Text. Plain Text wird versschlüsselt (encryption), die verschlüsselte Nachricht heißt dann Ciphertext. Ciphertext muß vor dem Lesen entschlüsselt werden (decription) (encipher - decipher).

Kryptographie: Das ist die Kunst bzw. Wissenschaft Nachrichten sicher zu verschüsseln.

Kryptoanalyse: Das ist die Kunst bzw. Wissenschaft verschlüsselte Nachrichten zu brechen bzw. zu entziffern.

Kryptologie: Das ist ein mathematisches Teilgebiet, das sich mit Kryptographie und Kryptoanalyse beschäftigit. Die benötigten mathematischen Theorien hierfür stammen aus der theoretischen Mathematik, der Statistik und der Zahlentheorie.

Plaintext (M oder P): stream of bits, textfile, bit...., video, ... Am Computer sind das natürlich binäre Daten

Ciphertext (C): Das sind ebenfalls binäre Daten, $\sharp bits(C) \geq \sharp bits(M)$.

Encription Function (E): E(M)=C

Decription Function (D): D(C)=M

$$\Rightarrow D(E(M)) = M!!$$

Authentication (Authentifizierung): Das ist die eindeutige Identifizierung einer Person. Der Empfänger soll den Ursprung der Nachricht sicher kennen. Ein Eindringling soll nicht vortäuschen können, jemand anders zu sein.

Integrity: Der Empfänger soll sicher sein können, daß die Nachricht nicht verändert worden ist.

Nonrepudiation: Der Sender soll nicht fälschlicherweise leugnen können, daß er eine Nachricht geschickt hat.

"Restricted Algorithm´´: Das ist ein kryptographischer Algorithmus (Cipher), dessen Sicherheit aus der Geheimhaltung des verwendeten Verfahrens besteht. Anwendungen mit solche Algorithmen werden auch als

"Low Security Applications´´ bezeichnet und haben eher historische Bedeutung. Jede mögliche nach außen gedrungene Information über das verwendete Verfahren muß eine Änderung des Verfahrens nach sich ziehen, da sonst die Sicherheit gefährdet ist. Aber nicht nur die Geheimhaltung ist ein Problem: Es kann keine Standardisierung des Verfahrens erfolgen und daher auch keine weit verbreitete Verwendung. Weiters fehlt die Qualitätskontrolle einer breiten Öffentlichkeit.

Trotz dieser sehr offensichtlichen Nachteilen gibt es immer wieder Institutionen, die glauben, daß die größte Sicherheit in der Geheimhaltung liegt. In Deutschland etwa, wurde vor wenigen Jahren die Verschlüsselung der Daten auf der Sozialversicherungskarte "geheim´´ entwickelt. Die Karte war nur sehr kurze Zeit in Verwendung, ehe der Verschlüsselungalgorithmus (ein simples XOR Verfahren) geknackt wurde.

"State of the art´´ sind also nicht geheime sondern allseits bekannte und überprüfte Algorithmen, die geheime Schlüssel, (Key(K)) verwenden. Key(K) kann jeden Wert aus einer großen Menge von Möglichkeiten annehmen. Der Wertebereich des Schlüssels heißt **Keyspace**.

$$E_K(M) = C \qquad D_K(C) = M$$

Manche Algorithmen benutzen unterschiedliche Schlüssel für das ver- und entschlüsseln.

$$E_{K_1}(M) = C \quad D_{K_2}(C) = M \qquad D_{K_2}(E_{K_1}(M)) = M$$

Die Sicherheit dieser Systeme beruht auf der Geheimhaltung bzw. teilweisen Geheimhaltung von Schlüsseln und nicht auf algorithmischen Details. Daher können solche Algorithmen veröffentlicht und analysiert werden.

Kryptosystem: Algorithmus + Plaintext + Ciphertext + Keys

1.2.2 Symmetrische Algorithmen

Bei diesen Algorithmen kann der Verschlüsselungskey aus dem Entschlüsselungskey berechnet werden und umgekehrt. Meist sind sie sogar identisch.

Bei den sogenannten "One Key Algorithms´´ müssen sich SenderIn und EmpfängerIn auf einen gemeinsamen Key einigen. Dazu müssen sie sicher kommunizieren können. Die Sicherheit dieses Systems liegt in der Sicherheit des Schlüssels, der geheim bleiben muß.

Grundsätzlich können zwei Kategorien unterschieden werden:

Stream Ciphers arbeiten zu einem bestimmten Zeitpunkt t an einem einzelnen Bit (manchmal auch Byte).

Block Ciphers arbeiten zu einem bestimmten Zeitpunkt t an einer Gruppe von Bits. Typischerweise sind dies 64 Bits.

1.2.3 Public-Key Algorithmen

Diese Klasse von Algorithmen werden auch als asymmetrische Verfahren bezeichnet. Der Verschlüsselungskey und der Entschlüsselungskey sind nicht identisch. Der Entschlüsselungskey kann nicht (in vernünftiger Zeit) aus dem Verschlüsselungskey berechnet werden.

> **Public Key**: Der Verschlüsselungskey kann öffentlich bekannt gegeben werden und wird daher auch als Public Key bezeichnet. Jede Person, die Zugang zu diesem Schlüssel hat, kann eine Nachricht verschlüsseln. Eine Entschlüsselung ist damit aber nicht mehr möglich.
>
> **Private Key**: Der Entschlüsselungskey muß geheim gehalten werden und wird daher auch als Private Key bezeichnet. Jene Person, die ihn besitzt, kann eine mit dem zugehörigen Public Key verschlüsselte Nachricht entschlüsseln.

Diese Verfahren werden auch für digitale Unterschrifen verwendet. In diesem Fall erfolgt die Verschlüsselung mit dem Private Key, die Entschlüsselung mit dem Public Key.

Public-Key Algorithmen haben den großen Vorteil, daß kein sicherer Kanal für die Übertragung eines gemeinsamen Schlüssels benötigt wird.

1.2.4 Cryptoanalysis

Es ist anzunehmen, daß die Cryptoanalytiker (LauscherIn, FeindIn) kompletten Zugang zu den Kommunikationen zwischen Sender und Empfänger hat. Daraus ergibt folgende Frage:

“Ist eine Erzeugung des Plaintextes ohne Kenntnis der Schlüssel möglich?´´

> **Attack, Attacke**: Eine versuchte Kryptoanalyse wird als Attack bezeichnet. Dabei wird davon ausgegangen, daß der/die FeindIn detaillierte Kenntnis des verwendeten Verfahrens hat.
>
> **Compromise**: Ein Verlust des Schlüssels durch nicht-kryptoanalytische Methoden (Indiskretion, ...) heißt Compromise.

Es gibt mehrere mögliche Attacken:

Ciphertext-only Attack: Der/die FeindIn besitzt den Ciphertext von verschiedenen Nachrichten, die mit dem gleichen Algorithmus verschlüsselt wurden.
geg: $C_1, \dots C_i$
ges: $P_1, \dots P_i$ oder Algorithmus um P_{i+1} aus C_{i+1}berechnen zu können.

Known Plaintext Attack: Der/die FeindIn kennt Ciphertexte und Plaintexte.
geg: $P_1, C_1, \dots P_i, C_i$
ges: Key oder Algorithmus um P_{i+1} aus C_{i+1}berechnen zu können.

Chosen Plaintext Attack: Der/die FeindIn kann den Plaintext auswählen, der verschlüsselt wird. Dadurch können bestimmte Plaintexts verwendet werden, die mehr Information über die Schlüssel liefern.
geg: P_1, $C_1, \ldots P_i$, C_i, wobei $P_1, \ldots P_i$ frei wählbar sind
ges: Key oder Algorithmus um P_{i+1} aus C_{i+1} berechnen zu können.

Adaptive-Chosen Plaintext Attack: Der/die FeindIn kann den Plaintext adaptiv verändern, je nach Ergebnis des letzten Ver- und Entschlüsselungsprozesses.

Chosen-Ciphertext Attack: Der/die FeindIn kann verschiedene Ciphertexte auswählen, die entschlüsselt werden und hat Zugang zum entschlüsselten Plaintext. Diese Attacke wird vor allem bei Public Key Systemen verwendet. geg: C_1, $P_1, \ldots C_i$, P_i, wobei $P_1, \ldots P_i$ frei wählbar sind
ges: Key

Rubber-Hose Cryptoanalysis: Verwendung von Folter und Bedrohung um einen Key zu bekommen. Das ist oft am effizientesten.

1.2.5 Algorithmen Sicherheit

Verschiedene Algorithmen haben verschiedene Sicherheitsgrade:

- Wenn die Kosten einen Algorithmus zu brechen höher sind als der Wert der Daten, dann ist der Algorithmus wahrscheinlich sicher.
- Wenn die Zeit, einen Algorithmus zu brechen größer ist, als die Zeit, die die Daten geheim sein müssen, dann ist der Algorithmus wahrscheinlich sicher.

Wahrscheinlich: Es gibt immer wieder unglaubliche Durchbrüche in der Kryptographie.

Kategorien des Brechens von Algorithmen:

1. **Total Break**: Der Schlüssel wird gefunden.
2. **Global Deduction**: Es wird ein alternativer Algorithmus zum Berechnen von $D_K(C)$ gefunden.
3. **Local Deduction**: Es wird der Plaintext aus dem Ciphertext einmalig ermittelt.
4. **Information Deduction**: Eine Information über den Schlüssel oder den Plaintext wird genommen.

   Unconditionally Secure: Es ist egal wieviel Information über den Ciphertext vorhanden ist. Es reicht nicht aus, um damit den Plaintext zu ermitteln. (Ausnahme: One-Time Pad, siehe später.)

   Brute-Force Attack: Es wird jeder mögliche Schlüssel ausprobiert. (So ist jedes System durch eine Ciphertext only Attacke knackbar.)

   Computationally Secure: Ein Algorithmus ist mit den zur Verfügung stehenden Mitteln (Daten, Speicherbedarf, Komplexität) nicht knackbar

1.3 Grundlagen: Protokolle, Hash-Funktionen, grundlegende Methoden

1.3.1 Einfache Algorithmen

Diese Algorithmen werden auch als Char-basierte Algorithmen bezeichnet, ein Char im Plaintext wird durch einen Buchstaben im Ciphertext ersetzt. Stammen aus der vor-Computerzeit !

1.3.1.1 Substitution Ciphers

Simple Substitution Ciphers: Ein Char im Plaintext wird durch ein entsprechendes Char im Ciphertext ersetzt. (Caesar Cipher, ist extrem einfach.)

Homophonic: Ein Char im Plaintext hat mehrere Möglichkeiten der Abbildung in den Ciphertext.

Polygram: Blöcke im Plaintext werden durch Blöcke im Ciphertext ersetzt. (Playfair WW1)

Polyalphabetic: Das ist eine Zusammensetzung aus mehreren Substitution Ciphers, deren Verwendung von der Position im Plaintext abhängt. (wurde im amerikanischen Bürgerkrieg verwendet, Word Perfect)

1.3.1.2 Transposition Ciphers

Die Zeichen des Plaintextes bleiben gleich. Es wird nur die Anordnung geändert, z.B. die Zeilen als Spalten angeordnet. Solche Verfahren werden auch bei der Verschlüsselung von Fernsehkanälen verwendet.

Da die Verteilung der Buchstaben gleich bleibt, wie in der normalen Sprache, bietet diese einen verwertbaren Hinweis auf die Verschiebung.

1.3.1.3 Rotor Maschinen

Das sind mechanische Maschninen zur Verschlüsselung, die aus einem Keyboard mit Rotoren besteht. Je nach Anordnung ist eine beliebige Permutation und Substitution möglich. Das bekannteste Beispiel hierfür ist die im 2. Weltkrieg verwendete (und geknackte) Maschine Enigma. Diese verwendete 5 Rotoren, Transposition, Reflektor, ...

1.3.1.4 XOR

XOR wird auch als ausschließendes Oder bezeichnet. Das mathematische Symbol ist $\oplus$ bzw. ^ in der Programmiersprache C.

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

Symmetrische Algorithmen mit Schlüssel:

$$P \oplus K = C$$

$$C \oplus K = P$$

weil $a \oplus b \oplus b = a$.

Wird zum Teil als "almost as secure as DES´´ bezeichnet. NSA erlaubte in den USA den amerikanischen Telefongesellschaften das Benutzen dieses Systems.

Das System ist allerdings leicht zu knacken:

1. Die Key length kann durch counting coincidences bestimmt werden. Dabei wird ein XOR des Ciphertextes auf geshiftete (verschobene) Varianten des Ciphertextes angewendet und die gleich bleibenden Bytes gezählt.

   Mehrfaches der Key length: 6% gleichbleibende Bytes
   SOnstige Positionen: 0.4% gleichbleibende Bytes

   $\Longrightarrow$ Index of coincidence.
   Der kleinste Shift bis 6% ergibt die Keylength.

2. Wenn der Ciphertextes um die Größe der Keylength verschoben und mit dem unverschobenen Text ein XOR angewendet wird, wird der Key entfernt. Es bleibt dann der Plaintext übrig, auf den mit dem verschobenen Plaintext ein XOR angewendet wurde. Der Plaintext ist dann einfach durch die inverse Operation zu rekonstruieren.

1.3.1.5 One-Time Pads

Große, sich nicht wiederholende Menge von wirklich zufälligen Key-Buchstaben.

Die Verschlüsselung erfolgt dann auf folgende Weise:
$\oplus$26 des Plaintext und One-Time Pad. Jedes Keyelement wird nur einmal in einer Nachricht verwendet. Bei einer neuen Nachricht muß ein neuer Key benutzt werden. Das soll die perfekte Sicherheit gewährleisten. Ein zufälliger Key macht aus einem nicht-zufälligen Plaintext einen zufälligen Ciphertext.

Attacke: 1. Eine mögliche Attacke richtet sich gegen das zufällige Erzeugen von Keys (PRNG)!

2. Problem: Key darf nie wieder verwendet werden. Für bits: analog mit XOR

Probleme: Diese Methode ist nur für kleine Nachrichten geeignet. Das Senden und Rekonstruieren muß synchronisiert werden.

Anwendung:
- "ultra secure´´ low-bandwidth,
- "Rotes Telefon´´ USA - Moskau.

1.3.2 Protokolle

Kryptographie soll zum Lösen von Problemen verwendent werden. Sind nur die kryptographischen Algorithmen bekannt, bleibt es eine rein akademische Disziplin. Zur Verwendung im "Alltag´´ müssen weitere Abmachungen getroffen werden. Diese Abmachungen werden Protokolle genannt.

> **Protokoll**: Ein Protokoll ist eine Szenerie von Vorgängen zwischen zwei oder mehreren Parteien, um eine Aufgabe zu bewältigen. Dabei ist die Reihenfolge der Vorgänge meist von entscheidender Bedeutung.

- Jede/r muß alle Schritte kennen.
- Jede/r muß zustimmen den Schritten zu folgen.
- Protokoll ist eindeutig.
- Für jede Situation gibt es einen definierten Schritt.

In einem Protokolle gibt es **MitspielerInnen** mit festgelegten Rollen:

Alice	initiiert alle Protokolle.
Bob	antwortet Alice.
Carol, Dave	sind dritte und vierte TeilnehmerInnen.
Eve	ist die Feindin bzw. Lauscherin.
Mallory	ist ein bösartiger, aktiver Angreifer.
Trent	ist der Vertrauensmann der desinteressiert Protokolle vervollständigt. Alle Parteien akzeptieren seine Aktivitäten als wahr, (z.B. Anwälte, etwa bei einer Bezahlung mit Scheck, prüfen, ob der Scheck o.k. ist und bezahlen erst bei Erhalt der Waren.)
Walter, Peggy, Victor	je nach Bedarf.

> **Arbitrated Protocol**: Ein Vertrauensmann wickelt so ein Protokoll ab. Im Netz ist es schwierig jemanden vertrauenswürdigen zu finden, ist ein Kommunkationsbottleneck, Frage nach Finanzierung !

> **Adjudicated Protocol**: Zweite Ebenene, wird nur invoked wenn ein Problem auftritt, nach dem Motto "nicht Betrug verhindern sondern aufdecken".

Self Enforcing Protocol: Das Protokoll selbst garantiert die Fairness.

1.3.2.1 Attacken gegen Protokolle

Es können verschiedene Attacken gegen Protokolle unterschieden werden:

Passive Attack: Eve kann das Protokoll mithören. Entspricht einer Ciphertext-only attack.

Active Attack: Mallory verändert das Protokoll zu seinem Vorteil:

- Er täuscht vor, ein anderer zu sein.
- Erfindet neue Nachrichten im Protokoll.
- Löscht, ersetzt, usw ....

Passive Cheating: Die AkteurInnen dieser Attackie sind diverse ProtokollteilnehmerInnen, die zwar dem Protokoll folgen, aber mehr Information wollen, als ihnen zusteht.

Active Cheating: Die AkteurInnen dieser Attackie sind diverse ProtokollteilnehmerInnen, die während der Teilnahme das Protokoll ändern.

1.3.2.2 Kommunikation mit symmetrischer Kryptographie

Bei einem symmetrischen Kryptographiealgorithmus sind folgende Kommunikationen bzw. Aktionen notwendig:

1. Alice und Bob einigen sich auf ein bestimmtes Kryptographiesystem.
2. Alice und Bob einigen sich auf einen gemeinsamen Key.
3. Alice verschlüsselt ihren Plaintext und erzeugt dadurch einen Ciphertext.
4. Alice schickt den Ciphertext an Bob.
5. Bob entschlüsselt den erhaltenen Ciphertext mit dem Key und erhält den Plaintext.

Dabei sind folgende **Störungen** möglich:

Eve hört mit bei Schritt 4. Das ist eine Ciphertext only Attakcke. Schritt 2 muß dann auf jeden Fall geheim passiert sein.

Mallory kann die Kommunikation unterbrechen und eigene Texte einschleusen, usw. ....

Alice könnte den Key an Eve weitergeben. Bob und Alice müssen sich daher vertrauen.

Diese möglichen Störungen zeigen die Probleme symmetrischer Verfahren:

1. Die Schlüsselverteilung muß geheim sein. Das ist bei großen Systemen natürlich problematisch.
2. Wenn der Key verloren geht, kann etwa Eve Bob vortäuschen, Alice zu sein und falsche Nachrichten verschicken.
3. Wenn in einem Netzwerk für jedes AnwenderInnen Paar ein verschiedener Key verwendet, ergeben sich $\frac{n(n-1)}{2}$ Keys, mit $n \ldots \#$Keys. Das ist natürlich nur bei einer kleinen Zahl von AnwenderInnen praktikabel.

1.3.2.3 One-Way Functions

One-Way Functions sind ein wesentlicher Bestandteil der Public-Key Kryptographie und verwenden Funktionen mit folgenden Eigenschaften:

1. Bei gegebenem x ist es leicht $f(x)$ zu berechnen.
2. Bei gegebenem $f(x)$ ist es sehr aufwendig x zu berechnen (=entschlüsseln).

Dieses Verfahren hat aber folgendes **Problem**: Nicht nur eine Attacke sondern auch das autorisierte Entschlüsseln wird damit erschwert bzw. unmöglich gemacht.

Daher wurden einige etwas modifizierte Verfahren entwickelt:

Trapdoor One-Way Function: Das sind Funktionen mit den bereits beschriebenen Eigenschaften und zusätzlich mit einem "Geheimnis" Y. Wer immer Y kennt, kann x schnell aus $f(x)$ berechnen.

One-Way Hash Function: Zu dieser Kategorie gehören etwa Kompressions- und Kontraktions Funktionen, Fingerprints usw.
Als Input wird ein pre-image mit variabler Länge benutzt. Die Funktion berechnet einen Output mit fixer Länge (hash value).

Ein einfaches Beispiel für ein pre-image wäre ein XOR aller Inputs. Aber Achtung, das ist keine echte one-way function.

Ziel des Verfahrens ist ein Fingerabdruck des pre-images. Das Verfahren erkennt aus einem Hash nicht notwendigerweise völlig eindeutig, ob zwei pre-images gleich sind, gibt aber sehr gute Hinweise.

Collision-free: schwer (unmöglich), zwei Pre-images aus einem Hash-value zu berechnen.

Message Authentication Code: Das ist eine One-Way Hash Funktion mit einem geheimen Key. Hash-value ist Funktion des pre-image und des Key.

1.3.2.4 Kommunikation mit Public-Key Kryptographie

Bei diesen Verfahren gibt es zwei Schlüssel:

- Public Key
- Private Key

Jeder, der den Public Key besitzt, kann eine Nachricht verschlüsseln aber nicht mehr entschlüsseln. Nur jene Person, die den Private Key besitzt, kann die Nachricht wieder entschlüsseln. Die mathematische Grundlage hierfür sind Trapdoor One-Way Functions. Verschlüsseln bedeutet $f(x)$ aus x zu berechnen. Entschlüsseln ist die Berechnung von x aus $f(x)$, das "Geheimnis" Y ist der Private Key.

1. Alice und Bob einigen sich auf ein Public Key Kryptrographiesystem
2. Bob schickt Alice seinen Public Key.
3. Alice verschlüsselt ihre Nachricht mit Bobs Public Key und schickt sie Bob.
4. Bob entschlüsselt die Nachricht mit seinem Private Key.

Daraus ergibt sich ein viel einfacheres Schlüsselmanagement. In der Praxis wird das meist so gehandhabt, daß zuerst ein System festgelegt wird. Jede/r TeilnehmerIn besitzt einen Private und einen Public Key. Der Public Key liegt in einer für alle zugänglichen Datenbank. Alice kann sich dann den Public Key von Bob in dieser Datenbank selbst abholen. Dabei muß garantiert sein, daß ein in der Datenbank deponierter Public Key auch wirklich von der richtigen Person stammt. Hierfür gibt es je nach Anzahl und geographischer Verteilung der TeilnehmerInnen verschiedene Methoden, z.B. Web of Trust, Notariate,....

1.3.2.5 Hybride Systeme

Public Key Systeme sind KEIN Ersatz für symmetrische Algorithmen:

1. Public Key Systeme sind langsamer (etwa um den Faktor 1000).
2. Sie sind durch Chosen-Plaintext Attacken verwundbar.
   $C = E(P)$, wobei P ein Plaintext aus einer Menge von n möglichen Plaintexten ist. Eine Kryptoanalyse muß nur alle n verschlüsseln und das Resultat mit C vergleichen, um P zu bestimmen.

In den meisten praktischen Einsatzbereichen wird Public Key Kryptographie verwendet, um Keys eines symmetrischen Verfahrens verschlüsselt zu verteilen:

1. Bob schickt Alice seinen Public Key.
2. Alice erzeugt einen zufälligen Session Key K, verschlüsselt diesen mit Bobs Public Key und schickt in zu Bob.
   $$E_B(K)$$

3. Bob entschlüsselt Alices Nachricht mit seinem Private Key und erhält den Session Key.
$$D_B(E_B(K)) = K$$

4. Beide ver- und entschlüsseln ihre Kommunikation mit dem Session Key.

Bei diesem Verfahren wird also ein Session Key erzeugt und nach der Verwendung wieder zerstört. Das verringert die Gefahr, ihn zu verlieren. Es bleibt natürlich die Verwundbarkeit des Private Key.

1.3.3 Digitale Unterschriften

Warum sind Unterschriften so wichtig?

1. Eine Unterschrift ist authentisch und identifizeirt die/den Unterzeichnende/n (Unterschreiber ist wer er vorgibt zu sein),
2. nicht fälschbar (nicht von jemand anderem imitierbar),
3. nicht wiederverwendbar, z.B. auf einem anderen Dokument.
4. Das unterschriebene Dokument kann nicht verändert werden.
5. Die/der Unterschreibende kann die Unterschrift nicht leugnen.

Diese Aufzählung entspricht natürlich etwas mehr dem Wunschdenken als der Wirklichkeit. Trotzdem kommt der Unterschrift große Bedeutung zu. Am Computer existieren weitere Probleme:

- Dateien können kopiert werden.
- Unterschriften können mit Cut und Paste auf ein neues Dokument "geklebt´´ werden.
- Dateien können leicht modifiziert werden, ohne daß es jemand bemerkt.

1.3.3.1 Signing mit symmetrischen Kryptographiesystemen und einem Arbitrator

Alice schickt Nachrichten an Bob. Trent kann mit Alice und Bob kommunizieren. Er teilt den geheimen Key K_A mit Alice und K_B mit Bob.

1. Alice verschlüsselt ihre Nachricht an Bob mit K_A und schickt sie an Trent.
2. Trent entschlüsselt mit K_A und verschlüsselt mit K_B.
3. Trent schickt die Nachricht Bob.
4. Bob entschlüsselt mit K_B und liest die Nachricht mit der Bestätigung, Daß sie von Alice ist.

1.3.3.2 Digitale Unterschriften mit Public Key

1. Alice verschlüsselt die Nachricht mit ihrem Private Key-
2. Alice schickt die Nachricht an Bob.
3. Bob entschlüsselt die Nachricht mit dem Public Key von Alice.

$\Longrightarrow$ Es wird kein Trent benötigt.

Zusätzlich werden One-Way Hash Funktionen verwendet. Sie haben den Vorteil, daß sie schnell zu berechnen sind.

1. Alice erzeugt den One-Way Hash eines Dokumentes.
2. Alice verschlüsselt den Hash mit ihrem Private Key (=Unterschrift).
3. Aice schickt das Dokument mit dem Unterschriftshash an Bob.
4. Bob produziert One-Way Hash des Dokumentes und entschlüsselt den unterschriebenen One-Way Hash. Stimmen die beiden überein, ist die Unterschrift gültig.

Folgende Bezeichnungen sind üblich: $S_K(m)$ Unterschrift mit Private Key,
$V_K(m)$ überprüfen der Unterschrift.

1.3.3.3 Digitale Unterschriften mit Verschlüssselung

1. Alice unterschreibt m mit ihrem Private Key $S_A(m)$.
2. Alice verschlüsselt unerschriebenes m mit Bobs Public Key und schickt ihm $E_B(S_A(m))$.
3. Bob entschlüsselt m mit seinem Private Key $D_B(E_B(S_A(m))) = S_A(m)$.
4. Bob überprüft die Unterschrift von Alice mit ihrem Public Key $V_A(S_A(m)) = m$.

Somit ergibt sich also eine Unterschrift sowohl auf dem Umschlag als auch auf dem Brief. Es werden für das Verschlüsseln und Unterschreiben verschieden Schlüsselpaare verwendet (außerdem: timestamps). .

1.3.3.4 Digitale Empfangsbestätigungen

Immer, wenn Bob eine Nachricht empfangen hat, schickt er sie als Bestätigung wieder zurück.

1. Alice unterschreibt, verschlüsselt und schickt $E_B(S_A(m))$.
2. Bob entschlüsselt und überprüft die Unterschrift $V_A(D_B(E_B(S_A(m)))) = M$.

3. Bob unterschreibt M mit seinem Private Key, verschlüsselt dann mit Alices Public Key und schickt das ganze an Alice zurück $E_A(S_B(m))$.

4. Alice entschlüsselt und überprüft die Unterschrift. Stimmen die Nachrichten überein, hat Bob die Nachricht korrekt bekommen.

Wenn für die Verschlüsselung und die Unterschriftsprüfung der gleiche Algorithmus verwendet wird, gibt eine mögliche Attacke:

$$V_x = E_x \quad \text{und} \quad S_x = D_x$$

Mallory besitzt einen eigenen Private und Public Key. Er liest Bobs Mail und behält die Nachricht von Alice an Bob. Nach einiger Zeit schickt er diese an Bob mit der Behauptung, daß sie von ihm käme. Bob entschlüsselt die Nachricht mit seinem Private Key und versucht Mallorys Unterschrift mit dessen Public Key zu überprüfen. Das ergibt das folgende Resultat:

$$E_M(D_B(E_B(D_A(m)))) = E_M(D_A(m))$$

Bob folgt dem Protokoll und schickt Mallory eine Empfangsbestätigung (receipt).

$$E_M(D_B(E_M(D_A(m))))$$

Mallory entschlüsselt nun mit seinem Private Key, verschlüsselt mit Bobs Public Key, entschlüsselt mit seinem eigenen Private Key und verschlüsselt mit Alices Public Key. Damit hat er die Nachricht M.

Das ganze Szenario ist nicht unrealistisch, etwa beim Senden von automatischen Empfangsbestätigungen (receipt).

Es gibt aber diverse Verbesserungen:

- Verwendung verschiedener Operationen (Algorithmus oder Key),
- Timestamps,
- Unterschriften mit One-Way Hash Funktionen.

Key Exchange: Häufig wird jede individuelle Kommunikation mit einem Session Key verschlüsselt. Die Verteilung dieses Schlüssels kann sehr kompliziert sein.

1.3.3.5 Key Exchange mit symmetrischer Kryptographie

Annahme: Alice und Bob teilen je einen geheimen Schlüssel mit dem Key Distribution Center (KDC). In unsererm Szenario fungiert Trent als KDC.

1. Alice verlangt von Trent den Session Key, um mit Bob kommunizieren zu können.

2. Trent generiert einen zufälligen Session Key. Er verschlüsselt zwei Kopien: Einen mit Alices Key und einen mit Bobs Key. Trent schickt dann beides zu Aice.
3. Alice entschlüsselt ihre Kopie.
4. Alice schickt Bob seine Kopie.
5. Bob entschlüsselt seine Kopie.
6. Alice und Bob verwenden den Key, um zu kommunizieren.

Das Verfahren hat einige bekannte Probleme:

- Die Sicherheit liegt ganz bei Trent.
- Trent ist ein “Bottleneck´´.

1.3.3.6 Key Exchange mit Public Key Kryptographie

Zur Erinnerung eine kurze Wiederholung:

1. Alice hat Bobs Public Key vom KDC.
2. Alice generiert den Session Key, verschlüsselt ihn mit Bobs Public Key und schickt ihn an Bob.
3. Bob entschlüsselt Aices Nachricht mit seinem Private Key.
4. Beide verwenden den Schlüssel zur Kommunikation.

Dabei gibt es eine bekannte Attacke **Man in the Middle Attack**:

Eve kann nur lauschen (Ciphertext-Only Attack). Mallory kann mehr:

1. Alice schickt Bob ihren Public Key. Mallory fängt den Key ab und schickt Bob seinen eigenen Public Key.
2. Bob schickt Alice seinen Public key. Wie vorher fängt Mallory den Key ab und schickt Alice seinen eigenen Public Key.
3. Wenn Alice an Bob eine Nachricht schickt, fängt Mallory diese ab, entschlüsselt sie, liest sie, verschlüsselt mit Bobs Key und schickt das Resultat an Bob.
4. Wenn Bob an Alice etwas schickt, handelt Mallory analog.

Diese Attacke funktioniert auch, wenn die Schlüssel in einer Datenbank liegen. Mallory muß nur vorher die Datenbank anzapfen.

Aber auch für diese Attacke gibt es eine mögliche Abhilfe: das **Interlock Protokoll**:

1. Alice schickt Bob ihren Public Key.

2. Bob schickt Alice seinen Public Key.

3. Alice verschlüsselt ihre Nachrict mit Bobs Key und schickt die halbe Nachricht an Bob.

4. Bob verschlüsselt seine Nachricht mit Alices Key und schickt die Hälfte an Alice.

5. Alice schickt die zweite Hälfte.

6. Bob setzt die beiden Hälften zusammen und entschlüsselt sie. Dann schickt er seine zweite Hälte.

7. Alice setzt die beiden Hälften zusammen und schlüsselt Bobs Nachricht.

Dieses Verfahren verhindert die "Man in the Middle´´ Attacke, da die Hälfte einer Nachricht ohne die andere Hälfte nicht entschlüsselt werden kann. Daraus läßt sich ein **Block Algorithmus** ableiten:

- Alle gesendeten Blöcke werden halbiert.
- Die Entschlüsselung hängt von der Initialisierung ab. Diese wird erst in der zweiten Hälfte gesendet.
- Die beiden halben Blöcke enthalten folgenden Inhalt:
  Erste Hälfte: One-Way Hash Funktion der verschlüsselten Nachricht.
  Zweite Hälfte: Die Nachricht selbst.

Dieses Verfahren vereitelt Mallorys Attacke. Wenn er in Punkt (3) die halbe Nachricht abfängt, kann er sie nicht entschlüsseln, da ja nur vollständige Nachrichten entschlüsselt werden können. Er muß also eine neue Nachricht erfinden und weiterleiten. Um unerkannt zu bleiben, müßte er Bob und Alice imitieren können. Dies ist aber ein anderes Problem.

Neben der soeben vorgestellten Methode gibt es noch eine weitere Abhilfe gegen die "Man in the Middle´´ Attacke. Dies ist ein Key Exchange Verfahren mit Digitalen Signaturen. (Trent muß bestätigen, daß die jeweiligen Schlüssel von Alice und Bob sind.)

1.3.3.7 Authentication

Wenn Sie sich auf einem Computer anmelden, müssen Sie ein Passwort angeben, um sich zu authentifizieren. Dazu muß der Computer allerdings die Passwörter kennen, um gültige von ungültigen unterscheiden zu können. Diese werden nicht im Klartext sondern als One-Way Funktionen gespeichert:

1. Alice schickt ihr Passwort.

2. Der Computer führt die One-Way Funktion des Passwortes durch.

3. Dann vergleicht er das Ergebnis mit seinen gespeicherten Daten.

Dadurch ist die Liste der Passwörter nutzlos, da sie nur Bilder der One-Way Funktion enthält. Aber auch in diesem Fall gibt es eine mögliche Attacke.

Dictionary Attack: Eine Liste von häufigen Passwörtern wird mit der (bekannten) One-Way Funktion verschlüsselt und mit den Einträgen in der Passwortliste verglichen. Diese Attacke ist sehr erfolgreich, da viele Menschen bei der Passwortgenerierung nicht sehr phantasievoll sind.

Salt: bietet eine mögliche Abhilfe. Ein zufälliger String wird einem Eintrag angehängt, ehe die Funktion angewendet wird.

Feldmeier /Karn: Eine Liste mit 732.000 Passwörtern und je 4096 salt values (12 Bit salt). Etwa 30 % aller Passwörter kann mit dieser Liste geknackt werden.

SKEY: Dieses System besteht aus einer One-Way Funktion f und einer zufälligen Zahl R.
$f(R), f(f(R)), \ldots f^{100}(R) \Rightarrow x_1, x_2, \ldots x_{100}$ Alice merkt sich die Zahlen und der Computer speichert x_{101}. Wenn Alice sich einloggen will, gibt sie x_{100} an. Der Computer berechnet $f(x_{100})$ und vergleicht das Ergebnis mit x_{101}. Wenn die beiden übereinstimmen, wird Alice akzeptiert. Der Computer ersetzt x_{100} und x_{101}. Alice streicht x_{100} von ihrer Liste.

1.3.3.8 Authentication mit Public Key Kryptographie

In einem Netzwerk ergibt sich manchmal die Situation, daß der Zugang zu einem bestimmten Computer oder System nur über diverse Zwischenstationen (Computer) möglich ist. Erschwerend kommt oft dazu, daß diese Zwischenstationen außerhalb des als sicher eingestuften Bereiches liegen.

Um so einen Zugang sicher zu gestalten, muß der ausgewählte Computer den Public Key der/des AnwenderIn gespeichert haben. Jede/r AnwenderIn besitzt den Private Key:

1. Der Computer schickt Alice einen zufälligen String.
2. Alice verschlüsselt den String mit dem Private Key und schickt das Ergebnis mit ihrem Namen zurück.
3. Der Computer entschlüsselt den String und vergleicht diesen mit dem Original.

Problem: Es ist immer unsicher, irgendwelche zufälligen Strings zu verschlüsseln.

1.3.3.9 Multiple Key Public Key Kryptographie

Das verallgemeinerte Konzept der Public Key Kryptographie wird als Multiple Key Public Key bezeichnet.

Alice	K_A
Bob	K_B
Carol	K_C
Dave	$K_A \& K_B$
Ellen	$K_B \& K_C$
Frank	$K_C \& K_A$

Das könnte etwa so ausschauen:

Alice verschlüsselt mit K_A
$\Longrightarrow$ Ellen oder Bob & Carol gemeinsam können entschlüsseln.

Dave K_A wie Alice
K_B wie Bob
oder so daß nur Carol entschlüsseln kann.

Encryption	**Decryption**
K_A	$K_B \& K_C$
K_B	$K_A \& K_C$
K_C	$K_A \& K_B$
$K_A \& K_B$	K_C
$K_A \& K_C$	K_B
$K_B \& K_C$	K_A

$\longrightarrow$ Ausweitbar auf n (beliebig viele) Keys: ist eine Teilmenge von Schlüsseln verwendet worden um eine Nachricht zu verschlüsseln, wird der Rest der Schlüssl benötigt um zu entschlüsseln.

Anwendung: Eine Gruppe von Leuten und man will mit vorher nicht bekannten Teilgruppen geheim kommunizieren. Dies bräuchte:

- Einen Schlüssel für jede mögliche Kombination
  $\rightarrow$ viele Schlüssel.
- Jede Nachricht muß für jede Kombination extra verschlüsselt werden
  $\rightarrow$ viele Nachrichten.

Mit der Multiple Key Public Key Kryptographie ist die Handhabung einfacher.

1.3.3.10 Secret Splitting

Secret Splitting bedeutet, daß eine Nachricht in mehrere Teile aufgeteilt wird. Jede einzelne Teil ist für sich alleine bedeutungslos. Alle gemeinsam ergeben erst die Nachricht.

z.B. Baupläne in Firmen, Vermeiden der Probleme beim Weggehen eines Mitarbeiters (siehe: Sicherheit von krypt. Algorithmen durch Geheimhaltung !),..

Das Verfahren ist einfach. Trent splittet M zwischen Alice und Bob:

1. Trent generiert einen zufälligen String R gleich lang wie M.

2. Trent wendet ein XOR auf M und R an, $\to S$:

$$M \oplus R = S$$

3. Trent schickt R an Alice und S an Bob.

Das ganze Verfahren ist eigentlich eine Verschlüsselung mit einem One-Time Pad , wo Alice das Pad besitzt und Bob den Ciphertext.

Secret Splitting funktioniert auch mit mehreren TeilnehmerInnen.

1. Trent generiert drei Strings R, S, T.
2. $$M \oplus R \oplus S \oplus T = U$$

   usw.

Wenn allerdings ein einiger Teil verloren geht, kann das Verfahren nicht zu Ende geführt werden.

1.3.3.11 Secret Sharing

(m, n)-threshold Verfahren: ein einfaches Beispiel solcher Verfahren: Die Nachricht wird in n Stücke (shadows) geteilt, sodaß mit m Teilen die Nachricht rekonstruiert werden kann. Wichtig dabei ist m‘n!

Chapter 2

Klassische Kryptographische Algorithmen

2.1 DES

2.1.1 Geschichte

DES ist die Abkürzung für **Data Encryption Standard** und ist ein seit 20 Jahren standardisiertes Verfahren. Der richtige Name ist eigentlich DEA, Data Encryption Algorithm nach ANSI in den USA oder DEA-1 nach ISO.

Anfang der 70er Jahre suchte das NBS (National Bureau of Standards) nach einem Verschlüsselungssystem, das standardisiert werden konnte, und der Öffentlichkeit zum Schutz wichtiger Daten zur Verfügung gestellt werden sollte. Es hat bis zu der Zeit bereits verschiedenste mehr oder weniger ausgereifte Systeme gegeben. Das Problem war, daß diese aber untereinander nicht kompatibel waren.

Nach einem ersten erfolglosen Versuch wurde 1974 das System Lucifer von IBM für das NBS weiterentwickelt und zum DES Standard erklärt. Der dabei verwendete Algorithmus war seit längerem der Öffentlichkeit bekannt, und mehrmals auf seine Zuverlässigkeit überprüft worden. Die Mitarbeit des NBS an seiner Weiterentwicklung ließ allerdings das Gerücht aufkommen, das NBS hätte heimlich eine Hintertür für den eigenen Gebrauch (Trapdoor) eingebaut.

Am 23.11.1976 wurde der DES Standard schließlich veröffentlicht.

Schon 1987 gab es Diskussionen um die Wiederzertifizierung, jedoch wurde DES sogar 1993 nochmals zertifiziert (obwohl es technologisch schon gar nicht mehr auf der Höhe der Zeit war, “never say not”). 1998 wiederholte sich die Ablösediskussion, ein Verfahren zur Ablöse wurde initiiert. Nachfolgealgorithmus wird AES heißen, Informationen dazu unter `http://www.nist.gov/aes`.

2.1.2 Funktionsweise

DES ist ein symmetrisches Blockcipher Verfahren, das auf 64 Bits aufbaut Die Schlüssellänge ist 56 Bits. Eigentlich werden 64 Bits angenommen, aber dann jedes 8. wieder weggelassen. Die gesamte Sicherheit des Systems liegt im Schlüssel!

Der Algorithmus selber besteht aus einer einfachen Kombination von Confusion und Diffusion nach Shannon.

> **Confusion**: Ein Algorithmus mit Confusion verschleiert die Beziehung zwischen Plaintext und Ciphertext. Statistische Muster und Redundanz werden verrückt (Substitution).
>
> **Diffusion**: Ein Algorithmus mit Diffusion verteilt die Information des Plaintextes über den gesamten Ciphertext (Permutation). Eine Änderung im Plaintext bewirkt eine Änderung an vielen Stellen des Ciphertextes.

Das Kernstück des DES Verfahrens - der “fundamental building block´´ kombiniert Confusion und Diffusion unter Verwendung eines Schlüssels in einer Schleife. Diese Schleife wird 16 mal abgearbeitet.

DES arbeitet mit einem 64 Bit Plaintext Block. Nach einer initialen Permutation wird der Block in zwei Hälften geteilt (je 32 Bit). In den 16 Runden identischer Operationen werden die Daten mit dem Schlüssel kombiniert. Am Ende werden die beiden Hälften wieder zusammengeführt und eine finale Permutation beendet den Algorithmus.

In jeder Runde wird der Schlüssel geshiftet und von 56 Bit 48 ausgewählt. Die rechte Hälfte der Daten wird auf 48 Bits vergrößert (Expansion, Permutation) und ein XOR auf diese Daten und den geshifteten und permutierten Schlüssel angewendet. Die resultierenden Daten werden durch 8 sogenannte S-Boxen geschickt (= 32 Bits) und dann nochmals permutiert. Dieser gesamte Vorgang wird mit der Funktion f dargestellt. Auf das Ergebnis von f und die linke Hälfte der Daten wird dann ein XOR angewendet. Das Ergebnis dieser Operation wird dann die neue rechte Hälfte. Die alte rechte Hälfte wird die neue linke Hälfte.

Der gesamte bis jetzt beschriebene Vorgang wird 16 mal wiederholt.

2.1.2.1 Die Initial-Permutation

Diese Permutation wird vor der 1. Schleife ausgeführt. Dabei werden Bits im Plaintext verschoben:

$$\begin{array}{ccc} 58.\text{ Bit} & \rightarrow & 1.\text{Bit} \\ 50.\text{ Bit} & \rightarrow & 2.\text{Bit} \\ 42.\text{ Bit} & \rightarrow & 3.\text{Bit} \\ \ldots & \rightarrow & \ldots \end{array}$$

Die initiale als auch die finale Permutation, betreffen nicht die Sicherheit sondern werden durchgeführt, um Plain- und Ciphertext Daten in Byte großen Teilen auf DES Chips laden zu können.

2.1.2.2 Die Key-Permuation

Der 64-Bit Key wird durch Weglassen jedes 8. Bits auf 56 Bits reduziert. Diese Bits können verwendet werden, um zu prüfen, ob der Schlüssel fehlerfrei ist. Der Schlüssel wird dann zweigeteilt und die Hälften jeweils zirkulär in Abhängigkeit von der aktuellen Schleife nach links geshiftet.

Anschließend weden 48 Bits ausgewählt und permutiert (Compressions Permutation). Das Bit 18, etwa, wird ignoriert. Auf Grund des Shiftens wird in jedem Subkey eine andere Teilmenge der Key Bits verwendet.

2.1.2.3 Die Expansion-Permutaion

Die rechte Hälfte der Daten wird von 32 auf 48 Bits erweitert. Dies hat zwei Gründe:

- Die Daten sind gleich lang wie der Key und können dann bei der Substitution wieder komprimiert werden.
- Ein Bit betrifft zwei Substitutionen. Dies erzeugt den sogenannten Lawineneffekt

2.1.2.4 Der Lawinen Effekt / Avalanche Effekt

Dieser Effekt wird manchmal auch als E-Box bezeichnet. Jedes Bit des Ciphertextes ist abhängig von jedem Bit des Plaintextes und des Schlüssels. Das ganze sollte außerdem möglichst schnell berechnet werden können.

$$\begin{array}{ccc}
\text{3. Bit} & \rightarrow & \text{4.Bit} \\
\text{4. Bit} & \rightarrow & \text{5.Bit} \\
\text{4. Bit} & \rightarrow & \text{7.Bit} \\
\dots & \rightarrow & \dots
\end{array}$$

2.1.2.5 Die S-Box Substitution

Nach der XOR Operation mit dem Schlüssel wird das 48 Bits lange Ergebnis einer Substitution unterworfen.

Jede S-Box hat 6-Bit Input und 4-Bit Output. Es gibt acht verschiedene S-Boxen. Die 48 Bits werden in 6-Bit Blöcke aufgeteilt. Jede S-Box hat 4 Zeilen (0-3) und 16 Spalten (0-15). Jeder Eintrag ist eine 4-Bit Zahl. Die 6 Input Bits bestimmen, wo in der Tabelle nachgeschaut werden muß.

$$\begin{array}{llll}
b_1, \dots, b_6\colon & b_1, b_6 & \text{laufen von 0-3:} & \text{zeile} \\
 & b_2, b_3, b_4, b_5 & \text{laufen von 0-15:} & \text{spalte}
\end{array}$$

Im folgenden Beispiel ist die 6. S-Box dargestellt.

$$\begin{array}{lll} 110011: & 11 \to 3 & \\ & 1001 \to 9 & \to 14 \\ \downarrow & & \\ 110010 & & \end{array}$$

Die S-Box Substitutionen sind keine linearen Operationen und für die Sicherheit von DES von zentraler Bedeutung. (Falls es wirklich eine Hintertür für dieses Verfahren geben sollte, wäre dies der geeignete Platz.)

Nach der S-Box Substitution wird der Output zu einem 32-Bit Block kombiniert.

2.1.2.6 Die P-Box Permutation

In diesem Schritt wird kein Bit zweimal verwendet und keines ignoriert.

Auf das Ergebnis wird mit der linken Hälfte des ursprünglichen 64-Bit Blockes ein XOR angewendet. Dann werden die beiden Rollen vertauscht und ein eine neue Schleife begonnen.

2.1.2.7 Die Finale Permutation

Diese Permutation ist invers zur initialen. So kann der Algorithmus sowohl zum Ent- als auch zum Verschlüsseln verwendet werden.

2.1.2.8 Entschlüsselung

Die Entschlüsselung funktioniert analog zur Verschlüsselung, nur daß die Schlüssel in der umgekehrten Reihenfolge verwendet werden. Auch der Key-Generierungsalgorithmus ist zirkulär, nur daß hier ein Rechts Shift verwendet wird (Position: 0,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1).

2.1.3 Anwendung

Für diesen Algorithmus wurde eigene Hardware entwickelt. Experimenteller DEC/Compac Chip 16.8 Mio Blöcke pro Sekunde (1 Gigabit/Sekunde). VLSI Tech. 6868 64 MByte/Sek. (1995), Softwarelösung z.B. 154.000 Blöcke pro Sekunde auf DEC/Compac ALPHA 4000/610.

2.1.4 Verwendungsarten

2.1.4.1 Electronic Codebook Mode (ECM)

Ein Block Plaintext wird in einen Block Ciphertext überführt. Dadurch ergeben sich diverse Vorteile aber auch Probleme.

Vorteil 1: Die Reihenfolge muß bei der Ver- bzw. Entschlüsselung nicht eingehalten werden. Dies eignet sich daher besonders gut für Verwendung in Datenbanken bzw. für die Parallelisierung.

Problem 1: Der Vorteil der einzelnen Blöcke macht diesen Modus anfälliger für diverse Attacken. Es kann ein Codebook aller Plaintext- Ciphertext Paare erstellt werden. Große Gefahr ergibt sich daher vor allem am Anfang und am Ende von Nachrichten, da sich hier bei bestimmten Texttypen oft wiederholende Stereotype befinden, z.B. Begrüßung.

Vorteil 2: Fehler im Code betreffen nur einen spezifischen Block. Wenn Blockgrenzen kontrolliert werden, ist auch ein Fehler von Bits kein Problem.

Problem 2: Block Replay Mallory könnte die verschlüsselten Nachrichten verändern, ohne daß es bemerkt wird.

z.B.:	B_1	Senderbank	$1,5$ Blocks
	B_2	Empfängerbank	$1,5$ Blocks
	Einzahlender		6 Blocks
	Konto		2 Blocks
	Summe		1 Blocks

Ein Block besteht dabei aus 8 Byte.
Mallory zeichnet die Kommunikation zwischen Banken auf und transferiert selbst 100 $ von B1 nach B2 und wiederholt das später. Er analysiert seine Aufzeichnungen auf zwei identische Nachrichten, die er findet. Nun kann er diese Nachricht nach Belieben einspielen und bekommt jedes Mal 100 $.
Abhilfe: Time stamp davor: 1 Block zusätzlich
Mallory kann allerdings herausfinden, wo es in den Nachrichten um seinen Namen geht und in jede Nachricht Originaldaten mit seinen Daten ersetzen.
Abhilfe: Die Schlüssel sollten oft gewechselt werden, denn dann muß auch Mallory schneller sein.

2.1.4.2 Cipher Block Chaining Mode (CBC)

Ein zusätzlicher Feedback Mechansimus wird eingefügt. Die Ergebnisse der Verschlüsselung des vorherigen Blocks werden zur Verschlüsselung des aktuellen Blocks verwendet. Jeder Ciphertext Block hängt nicht nur vom entsprechenden Plaintext Block ab, sondern auch von allen vorherigen Plaintext Blöcken.

Auf den Plaintext Block wird mit dem vorhergehenden Ciphertext Block ein XOR angewendet, ehe er verschlüsselt wird.

$$C_i = E_K(P_i \oplus C_{i-1})$$

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

Problem: Zwei identische Texte werden gleich verschlüsselt. Wenn sie sich ab einem bestimmten Punkt unterscheiden, unterscheiden sich auch die beiden Ciphertexte an der genau gleichen Stelle.
Abhilfe: Ein zufälliger Initialisierungsvektor wird als erster Block verschlüsselt.

Error Propagation: Ein Fehler im Plaintext betrifft zwar den gesamten Ciphertext, beim Entschlüsseln bleibt aber trotzdem nur der ursprüngliche Fehler zurück.

Ciphertext Fehler: (Channel oder Storage Error:) Ein Bit zerstört den gesamten Block und ein Bit des nächsten Plaintextes an der Fehlerstelle (error extension). Der zweite Block nach dem fehlerhaften Bit ist allerdings wieder korrekt ("self recovering´´).

2.1.4.3 Cipher Feedback Mode (CFB)

Block Ciphers werden auch als "self-synchronizing stream ciphers´´ verwendet. Diese Verwendungsart wird als Cipher Feedback Mode bezeichnet. Im CBC Mode muß gewartet werden, bis der gesamte Block vorhanden ist. Bei dieser Variante genügt es, wenn n Bits zur Verfügung stehen.

Ein Block Algorithmus im CFB Mode arbeitet mit einer Queue die genau Blockgröße hat. Zu Beginn wird die Queue mit einem Initialisierungsvektor gefüllt. Die Queue wird verschlüsselt und auf die n-Bit am linken Rand der Queue wird mit dem Plaintext ein XOR angewendet und an das rechte Ende der Queue gestellt. Alle anderen Bits der Queue werden nach links verschoben. Dann kommt der nächste n-Bit Block an die Reihe.

Die Entschlüsselung ist genau umgekehrt, aber beide im Verschlüsselungsmodus.

Der Initialisierungsvektor muß bei verschiedenen Anwendungen immer verschieden sein!

Error Propagation: Analog zum CBC Mode betrifft ein Plaintextfehler den gesamten Ciphertext. Nach der Entschlüsselung ist der Fehler aber wieder auf die Ausgangsstelle reduziert.

Ciphertext Fehler: Ein Fehler im Ciphertext verursacht einen Fehler im Plaintext. Die Fehlerstelle wird im Shift-Register aber langsam über äußeren Rand geschoben. Nach dem "Hinauswurf´´ des Fehlers, erholt sich das System wieder.

2.1.4.4 Output Feedback Mode (OFB)

Das ist eine Methode um einen Block Cipher als synchronen Stream Cipher zu verwenden. Das Verfahren ist dem CFB sehr ähnlich und verwendet ebenfalls Feedback. Allerdings werden hier n-Bits des verschlüsselten Shift Registers wieder im Shift Register verwendet. Die Entschlüsselung erfolgt wieder genau umgekehrt zur Verschlüsselung. Hier ist der Feedback u. a. von Plain- und Ciphertext, wird auch als "internal feedback" bezeichnet.

Der gesamte Verschlüsselungsvorgang kann geschehen, bevor der Plaintext vorhanden ist, da nur noch ein XOR angewendet wird.

Error Propagation: Hier gibt es keine Error Extension. Ein Bit-Fehler im Ciphertext bewirkt einen Bit-Fehler im Plaintext.

Sicherheits Problem: Der OFB sollte nur verwendet werden, wenn die Feedback Größe genau der Block Größe entspricht, da ein XOR des Keystreams mit dem Text ausgeführt wird. Wenn die Größen nun übereinstimmen, permutiert der Block Cipher die Shift Register Werte und die Periodenlänge ist im Schnitt $2^m - 1$, mit m als Blockgröße. Wenn die Feedbacklänge kleiner ist, wird die Periodenlänge kleiner, z.B. 2^{32} bei halber Blocklänge. Das ist aber nicht mehr genügend sicher.

2.1.5 Sicherheit von DES

DES hat einige Schwachstellen, insbesondere bezueglich spezieller Schlüssel.

Weak Keys: Durch das Halbieren und u.a. Shiften bleiben die Schlüssel gleich, die nur aus 0, 1 oder zur Hälfte aus 0, 1 bestehen.

Semiweak Key Pairs: Schlüsselpaare, die das gleiche Ergebnis liefern.

Possible Weak Keys: geben in den Keyshifts nur vier Schlüssel.

Complement Keys: Das Bitweise Komplement eines Schlüssels verschlüsselt das Komplement des Plaintext Blocks ist das Komplement des Ciphertextblocks. Sei X das Original und X' das Komplement:

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

Bei einer chosen Plaintext Attacke muß daher nur halbe Schlüsselanzahl getestet werden ($2^5 5$ statt $2^5 6$). Dieser Effekt entsteht durch das XOR der Subkeys nach der Expansionspermutation.

Algebraische Struktur: 64-Bit Plaintext Blocks ergeben 64 Bit Ciphertext Blöcke auf 2^{64} verschiedene Arten. DES benutzt 2^{56} (etwa 10^{17}) davon.

Mit mehrfacher Verschlüsselung scheint es möglich zu sein, einen größeren Anteil auszunutzen.

Ist DES eine abgeschlossene Gruppe (im algebraischen Sinn), gibt es für jedes Schlüsselpaar K_1 und K_2 ein K_3, sodaß

$$E_{K_2}(E_{K_1}(P)) = E_{K_3}(P)$$

1992 wurde allerdings bewiesen, daß DES keine Gruppe darstellt.

Schlüssellänge: Mit dem Vorhandensein von leistungsfähigeren Computern werden immer längere Schlüssel benötigt. 1993 wurde gezeigt, daß mit einem Computer im Werte von etwa einer Million Dollar eine Brute Force Attacke in 3, 5 Stunden ausführbar ist, es gibt solche Rechner. Je schneller die Computer werden, desto größer wird dieses Sicherheitsproblem.

Anzahl der Runden: 1982 wurde DES mit 3 und 4 Runden gebrochen, 1988 DES mit 6 Runden. Jeder DES mit weniger als 16 Runden kann mit "Differential Cryptoanalysis´´ weitaus effizienter als mit einer Brute Force Attacke gebrochen werden.

S-Box Design: Es gibt Strukturen, die das System stärken, und auch solche, die es schwächen. Die NSA veränderte das ursprüngliche Design der Box, um es zu Verbessern, eventuell auch um eine Hintertür von IBM zu entfernen und eine eigene einzuführen.

2.1.5.1 Differential Cryptoanalysis

Differential Cryptoanalysis ist eine Chosen Plaintext Attacke, die effizienter ist, als eine Brute Force Attacke.

Es werden Paare von Ciphertexten benutzt, deren Plaintexte spezifische Unterschiede haben. Die Entwicklung dieser Unterschiede währende der Rundendurchläufe mit dem gleichen Schlüssel wird betrachtet. Auf Grund der Unterschiede in den Ciphertexten werden verschiedene Schlüssel verschiedene Wahrscheinlichkeiten zugeordnet. Durch die Analyse von vielen Ciphertext Paaren wird ein Schlüssel zum Wahrscheinlichsten.

DES mit 16 Runden ist relativ resistent gegen diese Technik, da die Designer von DES diese Attacke kannten. Mit 19 oder mehr Runden, wird die Attacke theoretisch unmöglich, weil mehr als 2^{64} Plaintexte benötigt würden.

2.1.5.2 Linear Cryptoanalysis

Bei dieser Attacke wird eine lineare Approximation verwendet um die Wirkungsweise des Block Ciphers zu beschreiben. Die Grundidee sieht folgendermaßen aus:

$$\left\{\begin{array}{l}\text{XOR von Plaintext Bits}\\ \text{XOR von Ciphertext Bits}\end{array}\right\}\text{XOR}$$

Das Ergebnis ist ein XOR von einigen Key-Bits. Das wird als lineare Approximation bezeichnet, die eine bestimmte Wahrscheinlichkeit p hat korrekt zu sein. Ist $p \neq \frac{1}{2}$, wird hier angesetzt.

2.2 RSA

Die Bezeichnung RSA stammt von den Entwicklern R.Riverst, A. Shamir und L. Adleman (MIT 1977).

2.2.1 Mathematische Grundlagen - Zahlentheorie

Public-Key Verfahren beruhen nicht auf geheimen Schlüsseln sondern auf sogenannten “computational unfeasible problems´´. Das sind Probleme, die in der Komplexitätstheorie als “NP-complete´´ eingestuft werden.

Modulare Arithmetik:

$a \equiv b \pmod{n}$	"a kongruent b modulo n´´ Das ist die Schreibweise für $a = b + kn$, $k \in Z$. Ist a nicht negativ und $0 \leq b < n$ kann b als Divisionsrest von $\frac{a}{n}$ aufgefaßt werden. n wird als Modul, b als Residuum bezeichnet.
$0 \ldots n-1$	ist ein komplettes Restklassensystem modulo n (alle Residuen mod n, set of residues).
$a \bmod n$	Operation, die das Residuum von a bezüglich n liefert.
$17 \equiv 2 \pmod{3}$	

$$17 = 2 + 5 \cdot 3$$

$$17 \bmod 3 = 2$$

Das Ergebnis ist positiv. In den meisten Programmiersprachen sind allerdings auch negative Ergebnisse zugelassen. (Daher immer prüfen!)

Modulararithmetik ist kommutativ, assoziativ und distributiv. Jedes Zwischenergebnis kann genommen werden, das Ergebnis bleibt gleich.

Es gibt diverse Gründe, warum die Modular Arithmetik für die Kryptographie sehr interessant ist:

- Diskrete Logarithmen und Quadratwurzelberechnungen $\bmod\ n$ sind "computational infeasible´´.
- Für ein k-Bit Modul n ist das Zwischenergebnis jeder Addition, Subtraktion und Multiplikation auf $2k$ Bits beschränkt.
  $\rightarrow$ Exponentation kann in der Modular Arithmetik ohne große Zwischenergebnisse durchgeführt werden.
  $a^x \bmod n : (a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a) \bmod n$
  besser: $((a^2 \bmod n)^2 \bmod n)^2 \bmod n$
  $a^x \bmod n$, wenn $x \neq 2^n$: siehe folgendes
  Beispiel:
  $$\begin{array}{lcr} x & = & 25 = 2^4 + 2^3 + 2^0 \\ a^{25} \bmod n & = & (a \cdot a^{24}) \bmod n \\ & = & (a \cdot a^8 \cdot a^{16}) \bmod n \\ & = & (a \cdot ((a)^2)^2)^2 \cdot (((a^2)^2)^2)^2) \bmod n \\ & = & ((((a^2 \cdot a)^2)^2)^2 \cdot a) \bmod n \end{array}$$
  Wenn die Zwischenergebnisse gespeichert werden, benötigt die Berechnung nur sechs Multiplikationen.

  Primzahlen: Eine Primzahl ist eine positive ganze Zahl größer als 1, deren einzige Faktoren 1 und die Zahl selbst sind. Keine andere Zahl teilt eine Primzahl ohne Rest. Es gibt unendlich viele Primzahlen. Public Key Verfahren benutzen große Primzahlen. Es gibt ungefähr10^{151} Primzahlen mit 512 oder weniger Bits. $\sharp Primzahlen \leq n \sim \frac{n}{\log n}$.

  Warum sind Primzahlen interessant, wenn das Faktorisieren schwierig ist? Es ist schlecht, eine Zahl zu nehmen, eine Faktorisierung zu versuchen

und bei Fehlschlag annzunehmen, daß es eine Primzahl ist.
$\rightarrow$ Es gibt probalistische Primzahl Tests, die mit hoher Wahrscheinlichkeit bestimmen, ob es sich um eine Primzahl handelt oder nicht.

Größter Gemeinsamer Teiler: Zwei Zahlen heißen relativ prim , wenn sie außer 1 keine gemeinsamen Faktoren haben ("teilerfremd´´). $(a, b) = 1$
Der größte gemeinsame Teiler von a und b ist 1.
z.B. $(3, 5) = 1$, $(3, 6) = 3$
Eine Primzahl ist zu allen anderen Zahlen außer ihren eigenen Vielfachen relativ prim.

Euklidischer Algorithmus: Das ist das klassische Verfahren zur Berechnung des größten gemeinsamen Teilers von x und y.

Voraussetzung: $x, y > 0$

```
while (x > y)
    {
        g = x;
        x = y % x;
        y = g;
    }
return g;
```

Inversion in Modular Arithmetik: In der klassischen Arithmetik ist 4 invers $\frac{1}{4}$, weil $4 \cdot \frac{1}{4} = 1$
In der modularen Arithmetik $4 \cdot x \equiv 1 \pmod{n}$
$\Leftrightarrow$ gesucht sind x und $k \in Z$, sodaß $4x = kn + 1$
Allgemein: $a^{-1} \equiv x \pmod{n}$ hat eine eindeutige Lösung wenn $(a, n) = 1$.

Die Inversion kann mit dem erweiterten Euklidischen Algorithmus bestimmt werden.

Kleiner Satz von Fermat ("Kleiner Fermat´´): Sei p eine Primzahl und a kein Vielfaches von p.
$\Rightarrow a^{p-1} \equiv 1 \pmod{p}$

Beispiel: $p = 3$, $a = 2$

$$\begin{array}{ccccc} 2^2 & = & 4 & \equiv & 1 \pmod{3} \\ 7^2 & = & 49 & \equiv & 1 \pmod{3} \end{array}$$

Eulersche Phi-Funktion: Das reduzierte Restklassensystem modulo n ist jene Teilmenge des kompletten Restklassensystems deren Residuen relativ prim zu n sind. Ist n eine Primzahl, so ist reduziert = komplett nur ohne 0. 0 ist nie Teil des reduzierten Systems. Die Eulersche Phi-Funktion $\phi(n)$

gibt die Anzahl der Elemente des reduzierten Resklassensystems modulo n an. Dies ist gleichbedeutend mit $\phi(n) = \#m \in N$, $m < n$ mit $(m, n) = 1$.
Ist n eine Primzahl: $\phi(n) = n - 1$.
Ist $n = p \cdot g$ mit p und g Primzahlen: $\phi(n) = (p-1)(g-1)$.

Eulersche Verallgemeinerung des kleinen Fermat: $(a, n) = 1 \Rightarrow a^{\phi(n)} \equiv 1 \bmod n$ bzw. $a^{\phi}(n) \bmod n = 1$.
Wir hatten für inverses: $(a \cdot x) \bmod n = 1$
$\Rightarrow x = a^{\phi(n)-1} \bmod n$ ist inverses zu a.

Als Beispiel rechnen wir das Inverse von $5 \pmod 7$:
$\phi(7) = 7 - 1 = 6$, $5^{6-1} \bmod 7 = 3$
Das Inverse ist also 3.

Beide Methoden zur Bestimmung des Inversen lassen sich verallgemeinern: $((a, n) = 1)$, $(ax) \bmod n = b$
Euler: Löse $x = (ba^{\phi(n)-1}) \bmod n$.
Euclid: Löse $x = (b(a^{-1} \bmod n)) \bmod n$.

Chinesischer Restsatz: Sei $n = p_1 \dots p_t$ eine Primfaktorisierung und $(x \bmod p_i) = a_i$ mit $i = 1, 2, \dots, t$
$\Rightarrow x$ ist eindeutig und kleiner n. (Zwei Primzahlen dürfen gleich sein.)
$a < p$, $b < q$ mit p und q prim.
Dann gibt es genau ein x ($\exists^1 x$) mit $x < pq$, sodaß
$x \equiv a \pmod p$ und $x \equiv b \pmod q$.
Bestimmen Sie u mit dem Euklidschen Algorithmus, sodaß

$$uq \equiv 1 \pmod p$$

$$x = (((a - b)u) \bmod p)q + b$$

Quadratische Residuen: Sei p eine Primzahl. a ist ein quadratisches Residuum von p, wenn $x^2 \equiv a \pmod p$ für beliebige x.

$$\text{Beispiel: } p = 7 \quad \begin{array}{ccccc} 1^2 & = & 1 & \equiv & 1 \pmod 7 \\ 2^2 & = & 4 & \equiv & 4 \pmod 7 \\ 3^2 & = & 9 & \equiv & 2 \pmod 7 \\ 4^2 & = & 16 & \equiv & 2 \pmod 7 \\ 5^2 & = & 25 & \equiv & 4 \pmod 7 \\ 6^2 & = & 36 & \equiv & 1 \pmod 7 \end{array}$$

Ist p eine Primzahl und $q < p$, so ist q eine Primitivwurzel (primitives Element) PW modulo p, wenn

$$\forall b \;\; 1 \leq b \leq p - 1 \;\; \exists a, \;\; \text{sodaß} \;\; q^a \equiv b \pmod p.$$

Es können mit q also alle Zahlen bis p durch potenzieren erzeugt werden.

Galois Feld: Modular Arithmetik modulo einer Primzahl p: $GF(p)$.

Sehr häufig wird auch Modular Arithmetik mit irreduziblen Polynomen betrieben. ($x^2 + 1$ ist ireduzibel, $x^3 + 2x^2 + x$ist es nicht, da $= x(x + 1)(x + 1)$.) Gern: $p(x) = x^n + x + 1$
Koeffizienten sind Integer modulo q, $p(x)$ hat Grad n: $GF(q^n)$.

> **Diskrete Logarithmen**: $a^x \bmod n$ ist leicht zu berechnen. Die Umkehrung ist schwieriger:
> Suche x, sodaß $a^x \equiv b \pmod{n}$.
>
> Ist p prim, so ist die Komplexität diskrete Logarithmen zu berechnen in etwa vergleichbar mit dem Faktorisieren einer großen Zahl n, die aus zwei gleich großen Primzahlen zusammengesetzt ist. Intensive Berechnungen müssen nur einmal pro $GF(p)$ durchgeführt werden. Individuelle Logarithmen können dann schnell berechnet werden. Dies könnte bei der Anwendung in einem Verschlüsselungsverfahren zu einem Sicherheitsproblem führen.

2.2.2 RSA

2.2.2.1 Funktionsweise

Das RSA-Verfahren beruht auf der Schwierigkeit der Faktorisierung großer Zahlen. Der Public und der Private Key sind Funktionen eines Paares großer (100 oder 200 Stellen) Primzahlen. Um den Plaintext aus dem Public Key und dem Ciphertext zu ermitteln muß im Wesentlichen das Produkt der beiden Zahlen faktorisiert werden.

Das Verfahren basiert auf folgenden Schritten

1. Um die Schlüssel zu erzeugen, wählen Sie zwei zufällig generierte Primzahlen p und q (möglichst gleich groß).
2. Berechnen Sie das Produkt $n = pq$.
3. Wählen Sie zufällig den Verschlüsselungs Key e, sodaß e und $(p - 1)(q - 1)$ relativ prim sind.
4. Berechnen sie den Entschlüsslungs Key d, sodaß
   $ed \equiv 1 \pmod{(p-1)(q-1)}$
   (d ist also invers zu e.)

e und n sind der Public Key, d ist der Private Key. p und q werden nicht mehr benötigt. Um eine Nachricht m zu verschlüsseln wird m in mehrere numerische Blöcke $m_i < n$ aufgeteilt.

Verschlüsselung: $c_i = m_i^e \bmod n$
Entschlüsselung: $m_i = c_i^d \bmod n$

Dabei ergibt sich folgendes Problem:
Ist $(m_i, n) = p$ oder q kann die Faktorisierung von $n$berechnet werden, indem im wesentlichen (m_i, n) berechnet wird. Dies kann allerdings vermieden werden, indem

$(m_i, n) = 1$ für jeden Block überprüft wird. Ist $(m_i, n) \neq 1$, wird m_i nicht verwendet. Die Wahrscheinlichkeit, daß $(m_i, n) \neq 1$ ist, ist sehr klein für große p und q:

$$1 - \frac{\phi(n)}{n} = \frac{1}{p} + \frac{1}{q} - \frac{1}{pq}$$

Das RSA-Verfahren ist um einiges langsamer als DES: als Hardware Implementierung etwa 1000 mal, als Software etwa 100 mal. Wichtig ist eine gute Wahl von e. Sehr oft wird 3, 7, $65537(2^{16} + 1)$ genommen. Zahlen mit wenig 1 in der binären Darstellung erlauben effiziente Berechnung.

2.2.2.2 Sicherheit von RSA

Die Sicherheit von RSA liegt in der Faktorisierung von n. Es ist allerdings nie bewiesen worden, daß dies die einzige Möglichkeit ist, um die Nachricht m aus c und e zu berechnen.

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

Möglichkeiten der Attacke bestehen etwa in einem Rateversuch von $(p-1)(q-1)$ oder im Brute Force Angriff auf d. Beide Möglichkeiten sind allerdings nicht effizienter als eine Faktorisierung.

Im Folgenden werden einige mögliche Attackenszenarien vorgestellt.

2.2.2.3 Chosen Ciphertext Attacke

Szene 1: Eve bekommt c von Alice, das mit RSA und ihrem Public Key verschlüsselt wurde. Eve will $m = c^d$ berechnen. Sie wählt $r < n$ zufällig und holt e. Dann berechnet sie

$$\begin{aligned} x &= r^e \bmod n \\ y &= xc \bmod n \quad x = r^e \bmod n \Rightarrow r = x^d \bmod n \\ t &= r^{-1} \bmod n \end{aligned}$$

Eve bringt Alice dazu, y mit ihrem Private Key zu unterschreiben und dabei y zu entschlüsseln. (Muß die Nachricht, nicht den Hash unterschreiben.) Alice schickt Eve dann $u = y^d \bmod n$

Eve rechnet

$tu \bmod n = r^{-1}y^d \bmod n = r^{-1}x^d c^d \bmod n = c^d \bmod n = m$, weil $x^d = r$ ist.

Szene 2: Trent hat die Aufgabe, Nachrichten mit RSA zu signieren. Mallory hat ein m', die er von Trent unterschrieben haben möchte, was Trent normalerweise nicht täte.

Mallory wählt ein x beliebig und berechnet $y = x^e \bmod n$, $m = ym' \bmod n$ und schickt m zu Trent. Trent retourniert $m^d \bmod n$.

Mallory berechnet

$(m^d \bmod n)x^{-1} \bmod n$

$= ((x^e m')^d \bmod n)x^{-1} \bmod n$

$= x^{ed}x^{-1} \bmod n \cdot m'^d \bmod n = m'^d \bmod n$

Szene 3: Eve möchte, daß Alice m_3 unterschreibt. Sie generiert m_1 und m_2, sodaß $m_3 \equiv m_1 m_2 \pmod{n}$.
Wenn Alice m_1 und m_2 unterschreibt, kann Eve m_3 berechnen:
$m_3^d = (m_1^d \bmod n)(m_2^d \bmod n)$.

Diese Szenen zeigen, daß RSA nie benutzt werden darf, um von jemandem Fremden ein zufälliges Dokument zu unterzeichnen. Es muß zuvor immer eine One-Way Hash Funktion benutzt werden!

2.2.2.4 Common Modulus Attacke

Einige mögliche RSA Implementierung gibt jedem/r das gleiche n aber verschiedene e und d. Wird die gleiche Nachricht mit zwei relativ primen Exponenten verschlüsselt, kann der Plaintext ohne Private Key entschlüsselt werden.

$c_1 = m^{e_1} \bmod n$
$c_2 = m^{e_2} \bmod n$

Der/die FeindIn kennt n, e_1, e_2, c_1, c_2. $(e_1, e_2) = 1 \Rightarrow \exists r$ und s, sodaß $re_1 + se_2 = 1$, $r < 0$. Dann ist nur noch das folgende zu berechnne: c_1^{-1} und $(c_1^{-1})^{-r} \cdot c_2^s = m^{e_1 r} \cdot m^{e_2 s} = m \bmod n$.

2.2.2.5 Low Encription Exponent Attacke

Berechnungen sind schneller bei kleinem e. Es gibt eine Attacke gegen RSA, wenn $\frac{e(e+1)}{2}$ linear abhängige Nachrichten mit dem gleichen e aber verschiedenen Public Keys verschlüsselt werden. Wenn die Nachrichten identisch sind, droht Gefahr bei der Verschlüsselung von e Nachrichten. Wenn die Anzahl der Nachrichten kleiner als e ist oder die Nachrichten keine Beziehungen haben, gibt es für diese Attacke keinen Angriffspunkt.

Die Eliminierung dieser Sicherheitslücke ist sehr einfach. Es muß nur ein Padding mit unabhängigen zufälligen Werten verwendet werden.

2.2.2.6 Low Decription Exponent Attacke

Eine weitere Sicherheitslücke gibt es, bei $d \leq \frac{n}{4}$, $e < n$. Diese Konstellation kann allerdings bei kleinem e nicht auftreten. Daher d groß wählen !

2.2.2.7 Attacke gegen Verschlüsselung und Signierung

Alice verschlüsselt eine Nachricht mit Bobs Public Key und unterschreibt mit ihrem Private Key

$$(m^{e_B} \bmod n_B)^{d_A} \bmod n_A$$

Bob kann behaupten, Alice habe ihm m' und nicht m geschickt. Er kennt n_B und dessen Faktorisierung. Daher kann er den Diskreten Logarithmus bezüglich n_B berechnen. Er sucht ein x, sodaß $m'^x = m \bmod n_B$.

Wenn er nun xe_B als neuen Public Exponent publiziert und n_B behält, kann er behaupten Alice hätte ihm m' verschlüsselt mit xe_B geschickt.

Abhilfe bringt ein fixer Verschlüsselungsexponent.

Chapter 3

Weitere kryptographische Algorithmen

3.1 DES Varianten

3.1.1 Double Encription

$$C = E_{K_2}(E_{K_1}(P))$$
$$P = D_{K_1}(D_{K_2}(C))$$

Das ist nur sinnvoll, wenn DES als Gruppe nicht abgeschlossen ist (schon geklärt !).

“Meet in the middle Attacke´´: CryptoanalystIn kennt P_1, C_1, P_2, C_2, sodaß

$$C_1 = E_{K_2}(E_{K_1}(P_1))$$

$$C_2 = E_{K_2}(E_{K_1}(P_2))$$

> Für alle möglichen K (K_1 oder K_2) wird $E_K(P_1)$ berechnet und im Speicher abgelegt. Danach wird $D_K(C_1)$ für alle K berechnet und das identische Ergebnis im Speicher gesucht. Wird es gefunden, kann der aktuelle Schlüssel K_2 sein und der im Speicher K_1.
>
> Wenn nun versucht wird, P_2 mit K_1 und K_2 zu verschlüsseln und man bekommt C_2, so ist ziemlich sicher, daß K_1 und K_2 gefunden wurden. Wenn nicht, wird einfach weiter gesucht. Ein Problem ist allerdings der Speicherbedarf von 2^n Blocks. Bei DES : 2^{56} 64-Bit Blöcke. Das sind etwa 10^{17} Bytes.

3.1.2 Triple Encription

Es gibt verschiedene Triple Encription Verfahren. Sie unterscheiden sich unter anderem in der Anzahl der verwendeten Schlüssel.

3.1.2.1 Triple Encription mit zwei Keys

$$C = E_{K_1}(D_{K_2}(E_{K_1}(P)))$$
$$P = D_{K_1}(E_{K_2}(D_{K_1}(C)))$$

Dieses Verfahren wird auch EDE (encrypt-decrypt-encrypt mode) genannt. Während der gewöhnliche Block Algorithmus einen n-Bit Key hat, benutzt dieser einen $2n$-Bit Key. K_1 und K_2 wechseln sich ab, um eine Meet in the Middle Attacke auszuschließen.

Known Plaintext Attacke:

1. Eraten Sie einen Zwischenwert a.
2. Tabulieren Sie für alle K_1 den zweiten Zwischenwert, b, wenn der erste Zwischenwert a ist, einen bekannten Plaintext benutzend.
   $b = D_{K_1}(C)$.
3. Suchen Sie in der Tabelle für alle möglichen K_2 Elemente mit passendem zweiten Zwischenwert b:
   $b = E_{K_2}(a)$.
4. Die ERfolgswahrscheinlichkeit ist $\frac{p}{m}$, wobei p die Anzahl der bekannten Plaintexte und m die Blockgröße ist. Wenn Sie keinen Treffer erziehlt haben, versuchen Sie es wieder.

Diese Attacke braucht $\frac{2^{n+m}}{p}$ Zeit und p Speicher. Bei DES mit $p > 256$: $\frac{2^{120}}{p}$ ist schneller als eine vollständige Suche.

3.1.2.2 Triple Encription mit drei Keys

Ist so sicher wie man glaubt daß die Double Encription ist.

3.1.2.3 Triple Encription mit Minimum Key (TEMK)

Hier werden aus zwei Keys drei erzeugt, sodaß die Sicherheit der Triple Encription erhalten bleibt.

3.1.2.4 Triple Encription Modes

Die Triple Encription kann in verschiedenen Varianten benutzt werden.

Inner CBC: Die gesamte Nachricht wird im CBC Mode dreimal verschlüsselt. Es werden drei verschiedene Initialisierungsvektoren (C_0, S_0, T_0) benötigt.

$$C_i = E_{K_3}(S_i \oplus C_(i-1)$$
$$S_i = D_{K_2}(T_i \oplus S_(i-1)$$
$$T_i = E_{K_1}(P_i \oplus T_(i-1)$$

Outer CBC: Die gesamte Nachricht wird im CBC Mode dreimal verschlüsselt.

$$C_i = E_{K_3}(D_{K_2}(E_{K_1}(P_i \oplus C_{i-1})))$$
$$P_i = C_{i-1} \oplus D_{K_1}(E_{K_2}(D_{K_3}))$$

ECB-CBC Kombination: Weiters gibt es auch eine Kombination zwischen ECB, CBC in verschiedenen Variationen.

Von diesen Varianten ist die Outer CBC Version die sichererste.

3.1.3 DES mit unabhängigen Subkeys

Für jede DES Runde wird ein anderer Key verwendet - 48 bit pro Runde, die Keylänge ist also 768 bit. Dadurch wird eine Brute-Force Attacke wesentlich erschwert, jedoch mit differential cryptoanalysis und 2^{61} chosen plaintexts zu brechen. Es scheint daß Veränderungen Bei den Keys DES nicht sicherer macht.

3.1.4 DESX

Diese Variante verwendet zusätzlich ein sogenanntes "Whithening". Zusätzlich zum normalen Schlüssel gibt es noch den 64-Bit Whithening Key. Vor der ersten Runde wird ein XOR auf diesen Schlüssel und den Plaintext angewendet. Aus den 120 Key-Bits werden mit einer One-Way Funktion 64 neue Bits erzeugt. Nach der letzten Runde wird dann ein XOR auf diese neuen Bits und den Ciphertext angewendet. Diese Prozedur erhöht die Sicherheit von DES.

3.1.5 Generalized DES (GDES)

Diese Variante wurde entworfen, um den Standard DES-Algorithmus zu beschleunigen und um die Sicherheit zu erhöhen. Die Blockgröße wächst, während die Komplexität erhalten bleibt. GDES operiert mit Blöcken verschiedener Größe. Diese Blöcke werden in q 32-Bit Blöcke aufgeteilt. Die Funktion f (siehe DES **??** auf Seite ??) wird in jeder Schleife einmal auf den rechtesten Block angewendet. Auf das Ergebnis und die restlichen Blöcke wird danach noch ein XOR angewendet und nach rechts verschoben.

GDES benutzt eine variable Anzahl von Runden. Dabei wird die letzte Runde so gebaut, daß der Algorithmus als ganzes symmetrisch ist. Das bedeutet, daß sich Ver- und Entschlüsselung nur in der Anordnung der Subkeys unterscheiden.

Jedes GDES Schema, das schneller als das ursprüngliche ist, ist auch weniger sicher (Sicherheit versus Geschwindigkeit).

3.1.6 DES mit anderen S-Boxen

S-Boxenwurden gegen eine Differential Cryptoanalysis Attacke optimiert – dies hilft allerdings nicht gegen eine lineare Attacke. Es gibt bessere "Kompromiss-Boxes".

Diese Boxen müssen aber sorgfältig ausgewählt werden, da eine zufällige Wahl mit hoher Wahrscheinlichkeit sehr schlechte Ergebnisse liefert.

3.1.7 RDES

Bei dieser Variante wird der Austausch der linken und rechten Blockhälfte nach jeder Schleife durch ein Schlüssel abhängiges Tauschen ersetzt.

Noch besser ist allerdings die folgende Variante:
In jeder rechten Hälfte jeder Schleife werden Bits getauscht und/oder ein weiterer Tausch wird in Abhängigkeit der Input Daten durchgeführt (im ursprünglichen RDES in Abhängigkeit vom Schlüssel).

Je nachdem wie das Tauschen durchgeführt wird, gibt es diverse RDES- Varianten RDES-1, ..., RDES-4. Diese sind sicherer insbesondere gegenüber linearer und differentialer Cryptoanalysis.

3.1.8 s^n DES

s^n DES ist eine DES Variante, die veränderte S-Boxen verwendet. Dabei sind s^1 DES und s^2 DES eher schlecht, s^3 DES gut mit leichter Änderung, sowie s^4 und s^5 DES sehr gut geeignet.

3.1.9 DES mit Key-abhängigen S-Boxen

Diese DES Variante erschwert die lineare und differentiale Cryptoanalysis. Der folgende Algorithmus verwendet zusätzlich 56 Key Bits.

1. Die S-Boxen werden umgetauscht: 24673158.
2. Dann werden die ersten 16 der Key Bits verwendet:
   Wenn (1.Bit = 1) dann werden die ersten beiden Zeilen mit den letzten beiden Zeilen von Box Eins vertauscht.
   Wenn (2.Bit = 1) dann werden die ersten 8 Spalten mit den zweiten 8 Spalten vertauscht.
   Dann wird dieser Vorgang mit den anderen S-Boxen fortgesetzt.
3. Auf die restlichen Keybits wird der Reihe nach mit den Einträgen der S-Boxen (4 Bits für eine S-Box) ein XOR angewendet.

Diese Vorgangsweise hat den Vorteil, daß einige DES Chips mit verschiedenen S-Boxen betrieben werden können. Daher kann existierende Hardware weiterverwendet werden. Diese DES Variante kann auf Grund dieser Vorteile als gelungen bezeichnet werden.

3.2 Weitere Block Cipher Algorithmen

3.2.1 AES - Advanced Encryption Standard

AES ist der offizielle Nachfolger von DES. In einem offenen Ausschreibungsverfahren wurde aus 15 Submissions durch ein Expertenkommittee der “beste” Algorithmus ausgewählt. Juni 1998 AES Submission deadline, August 1999 wurden die 5 Finalisten bekanntgegeben, im Oktober 2000 stand der an der KU Leuven (Belgien) entwickelte “Rijndael” cipher als AES fest. Die Entscheidung ist ein Kompromiss zwischen Sicherheit und Ausführungseffizienz. Der Blockcipher “Square” ist der Vorgänger dessen bekannte Schwächen ausgebessert wurden.

AES ist ein klassischer Blockcipher mit 128 Bits Blockgrösse (Rijndael erlaubt auch 192 und 256 Bits), mit Schlüssellänge 128, 192 oder 256 Bits und einer variablen Anzahl von Runden (9 bei 128 Bits Key, 11 bei 192 Bits Key und 13 bei 256 Bits Key). AES operiert auf Byteebene.

Eine einzelne Runde von AES besteht aus folgenden Elementen (siehe Fig. 3.1):

- Byte Sub: jedes Byte im 128 Bits Block wird ersetzt durch einen in einer S-Box vordefinierten Wert ersetzt. Die S-Box entsteht durch ein Ersetzen eines Byte durch dessen Inverses im Galoisfeld GF(2^8), anschließender bitweiser Matrixmultiplikation modulo 2 und abschließendem XOR mit Hexadezimal 63. Diese Operation weist hohe Nicht-Linearität auf und wird als Lookuptable implementiert.
- Shift Row: betrachtet man die Bytes in Matrixschreibweise angeordnet wobei die Spalten zukzessive befüllt werden, so werden die Zeilen dieser Matrix zirkulär nach links geshiftet (0 - 4 Positionen).
- Mix Column: Jede Spalte der resultierenden Matrix wird mit einer speziellen Matrix multipliziert. Diese Multiplikation wird wiederum im GF(2^8) ausgeführt, d.h. die involvierten Bytes werden als Polynome behandelt und nicht als Zahlen. Hat das Ergebnis mehr als 8 Bits, wird mit dem erzeugenden Polynom von GF(2^8) XORed bis 8 Bits erreicht sind. Die Matrixmultiplikation kann sehr effizient auf Bitebene durch XOR ausgeführt werden. Diese Operation, gemeinsam mit Shift Row, führt zu hoher Diffusion.
- Add Round Key: ein XOR mit dem für die aktuelle Runde gültigem Key.

Um nun eine vollständige AES Verschlüsselung durchzuführen wird zuerst ein Add Round Key Schritt durchgeführt, dann die entsprechende Anzahl vollständiger Runden und eine finale Runde ohne den Mix Column Step. Für die Entschlüsselung muß die inverse S-Box und die Inverse der Matrix im Mix Column Step verwendet werden. Die Schlüssel für die verschiedenen Runden werden relativ aufwendig durch interne Bytemanipulationen (Shift und XOR), Anwendung der S-Box aus dem Byte Sub Schritt und XOR mit rundenabhängiger Konstante erzeugt.

Obwohl sowohl differenzielle und lineare Kryptoanalyse als auch die bekannte Attacke gegen Square für AES modifiziert werden können gibt es derzeit keine bekannten vernünftig durchführbaren Attacken gegen AES.

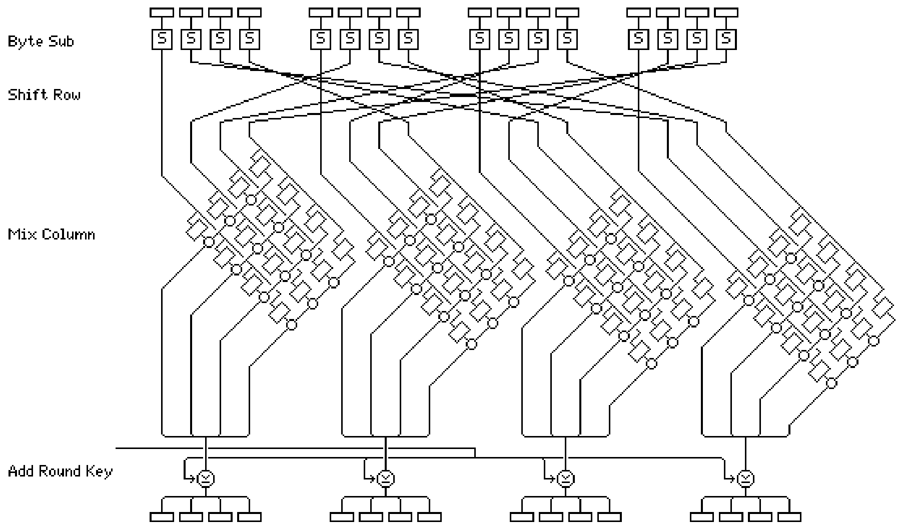

Figure 3.1: Funktionsweise von AES

3.2.2 IDEA

IDEA ist ein Standard, der 1990 entwickelt wurde. Allerdings hieß er am Anfang noch PES (proposed encryption standard) bzw. IPES. Erst 1992 wurde daraus IDEA (International Data Encryption Standard).

IDEA ist ein Blockcipher Algorithmus, der 64-Bit Plaintext Blöcke mit 128-Bit Keys verwendet. Verschlüsselungs- und Entschlüsselungsalgorithmus sind identisch.

Das Designkriterium für diesen Standard war der Mix von Operationen aus drei verschiedenen algebraischen Gruppen:

- XOR $\oplus$,
- Addition modulo 2^{16} $\boxplus$,
- Multiplikation modulo $2^{16}+1$. (Dies kann als eine S-Box von IDEA angesehen werden.) $\odot$

Alle diese Operationen arbeiten auf 16-Bit Subblöcken. Dabei wird der 64-Bit Block in 4 16-Bit Blöcke augeteilt: x_1, x_2, x_3, x_4. Diese Subblöcke sind die Eingabe für die erste Schleife. Acht Schleifen werden ausgeführt. In jeder einzelnen Schleife werden auf Block x_i mit Block x_j sowie mit sechs 16-Bit Subkeys die oben angeführten Operationen angewendet (XOR, Addition, Multiplikation). Zwischen den einzelnen Schleifen wird jeweils der zweite und der dritte Block vertauscht. Am Ende werden x_i und vier Subkeys zur Endtransformation kombiniert.

Eine einzelne Schleife besteht aus den folgenden Schritten, wobei mit x_i die Blöcke und mit z_i die Subkeys angegeben sind.

1. $x_1 \odot z_1$

2. $x_2 \uplus z_2$
3. $x_3 \uplus z_3$
4. $x_4 \odot z_4$
5. XOR (1) und (3)
6. XOR (2) und (4)
7. (5) $\odot z_5$
8. (7) $\uplus$ (6)
9. (8) $\odot z_6$
10. (7) $\uplus$ (9)
11. XOR (1) und (9) $\rightarrow$ nächste Schleife
12. XOR (3) und (9) $\rightarrow$ nächste Schleife
13. XOR (2) und (10) $\rightarrow$ nächste Schleife
14. XOR (4) und (10) $\rightarrow$ nächste Schleife

Nach der 8. Schleife:

1. $x_1 \odot z_1$
2. $x_2 \uplus z_2$
3. $x_3 \uplus z_3$
4. $x_4 \odot z_4$

Subkeys: Es gibt 52 Subkeys. Sechs werden pro Schleife und vier noch am Ende benötigt. 128 Bits werden in acht 16-Bit Blöcke aufgeteilt. Weitere Subkeys können generiert werden, indem die Blöcke immer um 25 Bits nach links zirkuliert werden.

Der Ablauf ist in der Abbildung 3.2 auf der nächsten Seite dargestellt. Dabei gelten folgende Notationen:
X_i: 16-Bit Plaintext Block
Y_i: 16-Bit Ciphertext Block
Z_i^r: 16-Bit Key Subblock von Schleife r.

Geschwindigkeit: Die Geschwindigkeit von IDEA ist bei einer Softwareimplementierung etwa doppelt so schnell wie bei DES.

Sicherheit: Es gibt keine bekannten Attacken gegen IDEA mit acht Schleifen.

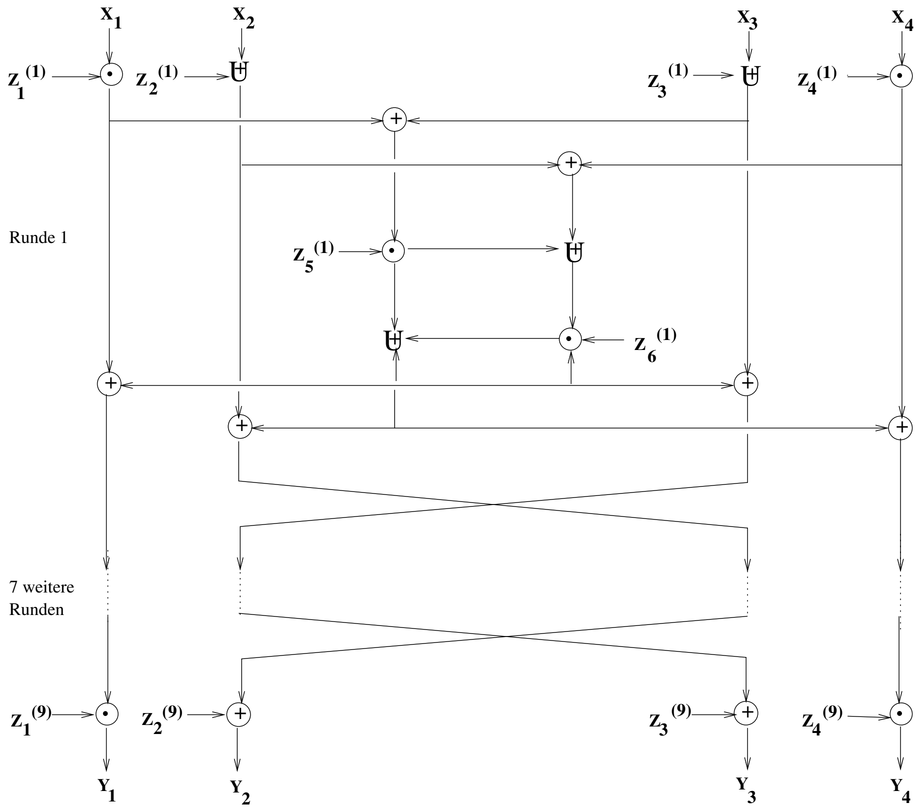

Figure 3.2: Funktionsweise von IDEA

3.2.3 GOST

GOST steht für "Gosudarstvennyi Standard" und ist ein russischer Verschlüsselungsstandard. Das ist ein 64-Bit Block Algorithmus mit einem 256-Bit Schlüssel und zusätzlichem Bitmaterial. GOST besteht aus 32 Schleifen.

Der Block wird in zwei Hälften L und R geteilt. Subkey für Runde i ist k_i.

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

wobei f: R_{i-1} und K_i addiert $\bmod\ 2^{32}$. Der Rest wird dann in acht 4-Bit Blöcke zerteilt und durch S-Boxen geschickt. Die S-Box führt eine Permutation der Zahlen 0-15 durch. Diese Permutation ist geheim. Daher werden zusätzliche Key-Bits benötigt. Nach den S-Boxen werden die Ergebnisblöcke rekombiniert und ein 11-Bit linkszirkulärer Shift durchgeführt. Auf diesen neuen Block und L_{i-1} wird dann ein XOR angewandt. Danach werden die Rollen vertauscht und eine neue Schleife begonnen.

Die Funktionsweise von GOST ist in der Abbildung 3.3 auf der nächsten Seite dargestellt.

Subkeyerzeugung: Der 256-Bit Schlüssel wird in 32-bit Blöcke k_1 - k_8 aufgeteilt. Diese werden dann wie in Tabelle 3.1 abgebildet verwendet.

Schleife:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Subkey:	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Schleife:	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Subkey:	1	2	3	4	5	6	7	8	8	7	6	5	4	3	2	1

Table 3.1: Verwendung von GOST Subkeys in den verschiedenen Schleifen

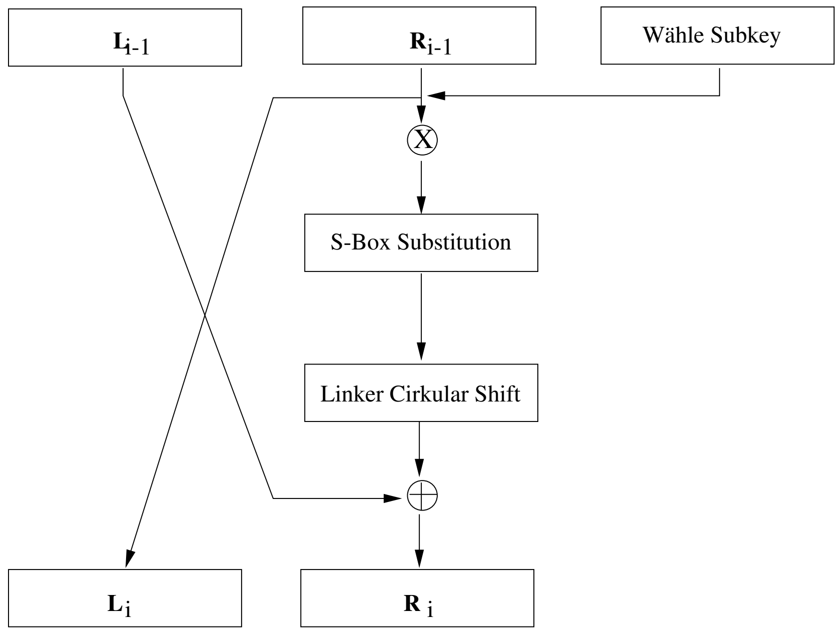

Figure 3.3: Funktionsweise von GOST

S-Boxenerzeugung: Die Struktur von S-Boxen wird entweder zufällig erzeugt oder zentral vorgegeben, aber in jedem Fall geheim gehalten. Dabei werden je nach AnwenderIn entweder besonders gute oder besonders schlechte (leicht knackbare) Boxen verteilt.

Sicherheit: Dieser Algorithmus ist wahrscheinlich sicherer gegen lineare und differentiale Cryptoanalyse als DES. Dies liegt in erster Linie an den geheimen S-Boxen und der höheren Rundenanzahl. GOST braucht acht Schleifen, damit einzelne Änderungen im Input alle Output-Bits betreffen. Bei DES ist dies bereits nach 5 Schleifen der Fall. Dies schwächt natürlich die Sicherheit. 32 Runden sind aber sehr aufwendig.

Der Grund für die Entwicklung von GOST war der Wunsch, einen Algorithmus zu designen, der besser als DES für die Software Implementierung geeignet ist. Da sich die EntwicklerInnen über die Sicherheit nicht im Klaren waren, versuchten sich durch Zusatzmaßnahmen das Knacken zu erschweren:

- Hohe Schleifenanzahl,
- langer Schlüssel,

- geheime S-Boxen.

GOST ist ein Kompromiß zwischen Effizienz und Sicherheit.

3.3 One-Way Hash Funktionen

Die Aufgabe einer One-Way Hash Funktion ist es, einen Fingerabdruck einer Nachricht zu liefern. Die Funktion $H(M)$ operiert auf einer beliebig langen Nachricht M und produziert einen Wert h mit fixer Länge m.

$$h = H(M)$$

Die Grundlage von Hash Funktionen ist daher die Kompression. Eine lange Nachricht M wird in einen kleineren Output h überführt. Zusätzlich sollen die folgenden Anforderungen erfüllt sein:

1. Bei gegebenem M soll h leicht zu berechnen sein.
2. Bei gegebenem h soll es schwierig sein M zu berechnen, sodaß $H(M) = h$ ist.
3. Bei gegebenem M soll es schwierig sein, eine andere Nachricht M' zu finden, sodaß $H(M) = H(M')$ ist.

Wenn Mallory die Punkte (2) und (3) durchführen kann, ist H nicht genügend sicher und Betrug leicht. Wenn, z.B. Alice eine Nachricht M digital unterschrieben hat, indem sie die Hash Funktion H auf M anwendet: $H(M)$, könnte Mallory M' erzeugen mit $H(M) = H(M')$. Somit könnte Mallory behaupten, Alice hätte ihm M' geschickt, was natürlich nicht der Wahrheit entspricht.

Für manche Anwendungen sind die geforderten Eigenschaften der One-Way Hash Funktion nicht ausreichend, da auch “Collision Resistence” benötigt wird.

> **Collision Resistence**: Eine Funktion H ist Collision resistent, wenn es schwierig ist zwei zufällige Nachrichten M und M' zu finden, sodaß $H(M) = H(M')$ ist.

Diese Eigenschaft ist wichtig bei der sognannten “Geburtstags Attacke” wichtig:

1. “Wie viele Menschen müssen in einem Raum sein, damit die Wahrscheinlichkeit größer $\frac{1}{2}$ ist, das eine der Personen an einem vorgegebenen Tag Geburtstag hat?” (183)
2. “Wie viele Menschen müssen in einem Raum sein, damit die Wahrscheinlichkeit größer $\frac{1}{2}$ ist, das zwei Personen am gleichen Tag Geburtstag haben?” (23)

Diese beiden Fragestellungen sind analog zu den folgenden:

1. Gegeben ist $H(M)$, suche M', sodaß $H(M) = H(M')$.

2. Suche M und M', sodaß $H(M) = H(M')$.

Eine Hash Funktion produziert einen m-Bit Output. Um eine Nachricht zu finden, die auf einen gegebenen Wert abgebildet wird, müssen 2^m produziert werden. Um zwei Nachrichten zu finden, die den selben Hash-Wert liefern, müssen nur $2^{\frac{m}{2}}$ Nachrichen produziert werden. Dies ist ein großer Unterschied.

Das folgende Beispiel zeigt, wie diese Schwäche ausgenützt werden kann.

1. Alice besitzt zwei Versionen eines Vertrages, wobei eine Version die echte ist, und die andere zu Ungunsten von Bob verändert worden ist.
2. Alice verändert beide geringfügig und berechnet die Hash-Werte der veränderten Dokumente (2^{32} verschiedene Dokumente).
3. Alice sucht ein Paar von Hash-Werten, die gleich sind, und rekonstruiert die Dokumente.
4. Alice und Bob unterschreiben den Vertrag, der für Bob in Ordnung ist mit einem Protokoll, bei dem er den Hash des Dokumentes unterschreibt.
5. Irgendwann ersetzt Alice die Originalversion mit der anderen - equivalent unterschrieben von Bob !!

DAS OUTING VON ALICE - FAST LIVE :-)

Eine Hash-Funktion mit 64-Bit Output ist zu wenig. Die meisten Funktionen, die in Verwendung sind, erzeugen 128 Bit.

Der folgende Algorithmus erhöht die Länge des Outputs. Die Sicherheit dieser Variante ist allerdings nicht ganz klar.

1. Hash erzeugen
2. Hash an Nachricht anfügen
3. Hash der verlängerten Nachricht erzeugen.
4. Hash anhängen und neuen Hash erzeugen, kann beliebig oft wiederholt werden.

Der Input (die Nachricht M) und der Output der Funktion werden also auf vorhergehende Nachrichtenteile angewendet.

$$h_i = f(M_i, h_{i-1})$$

Zwei Beispiele für verwendete Hash Funktionen sind SNEFRU und der N-Hash. Beide sind allerdings nicht sehr sicher.

Bessere Funktionen werden in den folgenden Kapiteln vorgestellt.

3.3.1 MD5 (Message Digest)

MD5 wurde von R. Rivest 1994 veröffentlicht. Die Nachricht M wird in 512-Bit Blöcke aufgeteilt, gruppiert in 16 32-Bit Unterblöcken. Der Output besteht aus 4 32-Bit Blöcken, vereinigt in 128-Bit Hash-Werten.

Die Nachrichten werden durch ein Padding auf einheitliche Länge gebracht. Die Anzahl der Bits beträgt ein Vielfaches von 512 minus 64 ($\sharp$ Bits $=512x - 64$). Die ursprüngliche Länge der Nachricht wird mit einer 64-Bit Zahl kodiert und an die verlängerte Nachricht angehängt. Somit ist die Länge des Ergebnisses ein exaktes Vielfaches von 512.

Es werden vier 32-Bit Initialvariablen A, B, C, D (chaining variables) verwendet. Die Hauptschleife besteht aus vier Durchläufen. In jedem Durchlauf wird eine bestimmte Operation 16 mal verwendet, wobei in jedem Durchlauf eine andere Operation verwendet wird.

Jede dieser Operationen wendet eine nicht lineare Funktion auf drei der Hilfsvariablen a, b, c, d an, addiert das Ergebnis zur vierten Variablen, zu einem Textblock M_j und einer Konstanten t_j. Das Ergebnis wird nach links geshiftet (variabel!) und zu einer der Variablen addiert. Das Ergebnis ersetzt dann a, b, c oder d, die Hilfsvariablen werden dann zu den chaining Variablen addiert.

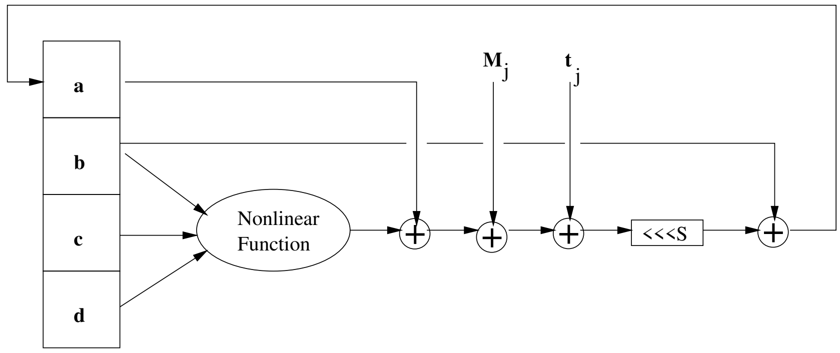

Figure 3.4: Eine einzelne MD5 Operation

Für jeden Schleifendurchlauf gibt es eine eigene nicht lineare Funktion F, G, H und I.

$$
\begin{aligned}
F(X,Y,Z) &= (X \wedge Y) \vee ((\neg X) \wedge Z) \\
G(X,Y,Z) &= (X \wedge Y) \vee (Y \wedge (\neg Z)) \\
H(X,Y,Z) &= X \oplus Y \oplus Z \\
I(X,Y,Z) &= Y \oplus (X \vee (\neg Z))
\end{aligned}
$$

M_j ist der j-te Nachrichtenblock (0-15) und $<<< s$ ist eine linear zirkulärer Shiftoperation. $FF(a, b, c, d, M_j, s, t_i)$ bedeutet dann

$$a = b \oplus ((a \oplus F(b, c, d) \oplus M_j \oplus t_i)^{<<<s}).$$

GG, HH, II sind analog definiert.
$t_i = \text{int}(2^{32}|\sin(i)|)$ in Schleife i, i in radians.

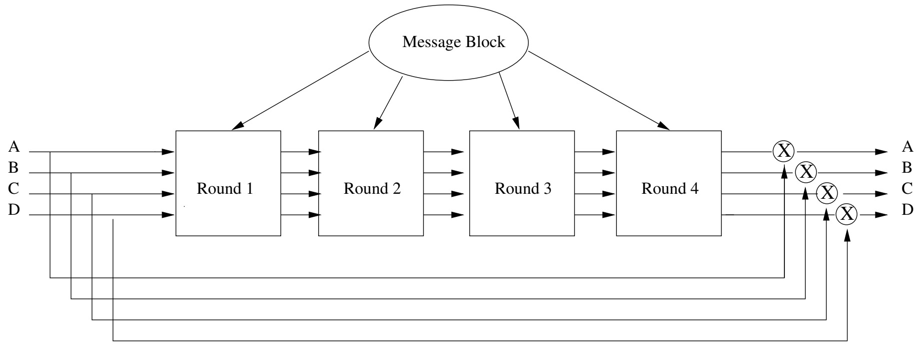

Figure 3.5: MD5 Hauptschleife

Sicherheit: MD5 ist eine Verbesserung der Funktion MD4 und kann trotz einer theoretischen Schwäche in der Kompressionsfunktion als sicher bezeichnet werden.

Eine weitere sichere Funktion ist der "Secure Hash Algorithm" (SHA), der einen 160-Bit Hash verwendet (1994 vom NIST veröffentlicht).

3.3.2 One-Way Hash Functions mit symmetrischen Block Ciphers

Die Idee hinter dieser Hash Variante ist die Folgende:
Wenn der gewählte Block-Cipher sicher ist, müßte es auch die zugeordnete Hash Funktion sein.

Dieser Algorithmus ist ganz offensichtlich ein CBC oder CFB mit fixem Schlüssel und Initialisierungsvektor. Der letzte Ciphertext Block ist der Hash Value. Dies ist aber nicht genug. Besser ist es, wenn die Nachricht als Schlüssel und der vorige Hash als Input verwendet wird. Somit entspricht die Blocklänge des Ciphers der Länge des Hashs.

Das tatsächliche angewendete Schemata ist allerdings etwas komplizierter!

$$H_0 = I_H$$
$$H_i = E_A(B) \oplus C$$

wobei I_H der Initialwert ist und A, B und C $M_i, H_{i-1}, (M_i \oplus H_{i-1})$ oder eine Konstante sein können.

> **Hash Rate**: Das ist die Anzahl der n-Bit Nachrichten Blöcke, die pro Verschlüsselung verarbeitet werden. n ist die Blockgröße des Ciphers.

Sicherheit: Vier sichere Varianten siehe Folie.

3.3.3 Modified Davies-Mayer (mit IDEA)

$$H_0 = I_H$$
$$H_i = E_{H_{i-1},M_i}(H_{i-1})$$

In dieser Variante hashed die Funktion die Nachricht in 64-Bit Blöcken und produziert einen 64-Bit Hash.

Sicherheit: Dieser Algorithmus gilt als sicher.

3.3.4 Tandem Davies Mayer (mit IDEA)

tandem davies mayer davies mayer tandem

$$G_0 = I_G \text{ initial } H_0 = I_H$$
$$W_i = E_{G_{i-1},M_i}(H_{i-1})$$
$$G_i = G_{i-1} \oplus E_{M_i,W_i}(G_{i-1})$$
$$H_i = W_i \oplus H_{i-1}$$

G_i und H_i werden 128-Bit Hash vereinigt. Schema siehe Folie.

Sicherheit: Dieser Algorithmus gilt als sicher.

3.3.5 One-Way Hash Functions mit Public Key Algorithmen

M ist wie immer die Nachricht.
$n = p \cdot q$
$(e, (p-1)(q-1)) = 1$
$H(M) = M^e \bmod n$.
Der private Key wird vernichtet.

Noch einfacher : große Primzahl p,

$H(M) = M^e \bmod p$.

Ist so sicher wie Algorithmen die auf diskreten Logarithmus beruhen.

Geschwindigkeit: **SEHR** langsam.

3.3.6 Message Authentication Codes MAC

Das ist eine schlüsselabhängige One-Way Hash Funktion. Nur jemand mit dem identischen Schlüssel kann den Hash verifizieren.

Beispiel: Es soll überprüft werden, ob Daten verändert wurden. One-Way Hash Funktin $\rightarrow$ MAC: Hash Value wird mit symmetrischem Algorithmus verschlüsselt.
$\leftarrow$: Schlüssel veröffentlichen.

3.4 Weitere Public-Key Algorithmen

3.4.1 Knapsack Algorithmen

Diese Algorithmen wurde von R. Merkle und M. Hellman erstmals für die Kryptographie verwendet. Das Knapsack Problem ist das der Sicherheit zugrundeliegende NP-vollständige Problem.

Knapsack Problem: geg: $M_1, M_2, \ldots, M_n$ sowie S
ges: $b_i \in \{0, 1\}$ sodaß $S = b_1 M_1 + \ldots + b_n M_n$

Beispiel: M_i: $1, 5, 6, 11, 14, 20$
$S = 22$ funktionert, $S = 24$ funktioniert nicht

Die Komplexität jedes Algorithmus zu Lösung des Knapsack Problems ist NP-complete (NP-vollständig). Das bedeutet, das es zur Verschlüsselung verwendet werden könnte, wenn eine geeignet Anwendung gefunden werden kannn.

Eine Nachricht wird als Lösung zu einer Reihe von Knapsack Problemen kodiert. Ein Plaintext Block, der gleich lang ist wie die Anzahl der M_is (also n), bestimmt die b_i. Der Ciphertext ist Ergebnissumme S.

	Plaintext	1 1 1 0 0 1	0 1 0 1 1 0
Beispiel:	Knapsack	1 5 6 11 14 20	1 5 6 11 14 20
	Ciphertext	1 + 5 + 6 + 20 = 32	5 + 11 + 14 = 30

Es gibt zwei verschiede Knapsack Probleme. Eines davon ist in linearer Zeit lösbar, das andere nicht. Das einfache Knapsack Problem kann verändert werden, um ein schwierigeres zu erhalten. Der Public Key ist das “schwierige Problem´´, das zum Verschlüsseln verwendet wird. Der Private Key ist das einfache Knapsack Problem, mit dessen Hilfe entschlüsselt werden kann. Wenn der Private Key nicht bekannt ist, müßte zum Entschlüsseln das schwierige Problem gelöst werden.

Einfaches Knapsack Problem: Ein Knapsack Problem wird als “einfach´´ bezeichnet, wenn die Liste der M_i eine superincreasing Sequence ist. Eine Folge besitzt genau dann diese Eigenschaft, wenn jedes Element größer ist, als die Summe aller vorherigen Elemente.

$$\text{superincreasing} \Longleftrightarrow \forall i \leq n : \sum_{j=1}^{i-1} M_j < M_i$$

Beispiel: $\{1, 3, 6, 13, 27, 52\}$ ist so eine Folge.

Wie kann diese “einfache´´ Problem nun gelöst werden? S wird mit der großen Zahl in der Folge veglichen.

$$\begin{array}{llll} If & S < M_n & : & b_n = 0 \\ ElseIf & S \geq M_n & : & b_n = 1 \\ S = S - M_n & & & \\ \text{Gehe zu } n-1 & & & \end{array}$$

Ist $S = 0$ geworden, gibt es eine Lösung (die Gefundene), andernfalls nicht.

Beispiel: $S = 70$, $M_i = \{1, 3, 6, 13, 27, 52\}$, Plaintext = 1 1 0 1 0 1

Schwieriges Knapsack Problem: Ein Knapsack Problem wird als “schwierig´´ bezeichnet, wenn die Liste der M_i **keine** superincreasing Sequence ist.

Für das Lösen des schwierigen Knapsack Problems gibt es keinen intelligenten Algorithmus. Die einzige Möglichkeit, eine Lösung zu finden, ist das methodische Testen von möglichen Lösungen. Die schnellsten Algorithmen haben einen exponentiellen Zeitaufwand bezüglich n (Anzahl der M_i). Dies bedeutet, daß jedes M_i mehr, die benötigte Zeit verdoppelt.

3.4.1.1 Merkle-Hellman Algorithmus

Der Merkle-Hellman Algorithmus verwendet den im Kapitel 3.4.1 auf der vorherigen Seite vorgestellten Knapsack Algorithmus zur Verschlüsselung. Der Private Key ist eine Folge von M_i eines einfachen Knapsack Problems. Der Public Key ist eine Folge von M_i eines schwierigen Knapsack Problems, wobei beide die selbe Lösung besitzen müssen.

Für das Finden der beiden Schlüssel gibt es einen Algorithmus, bei dem aus einem einfachen ein schwieriges Knapsack Problem mit der selben Lösung erzeugt werden kann:

1. Suche ein einfaches Knapsack Problem.
2. Wähle zwei Zahlen n und m mit $m > \sum M_i$ und $(n, m) = 1$.
3. Multipliziere alle M_i mit $n \bmod m$. Die daraus resultierenden M_i' bilden dann ein schwieriges Knapsack Problem mit der selben Lösung.

Beispiel: $K : \{2, 3, 6, 13, 27, 52\}$, $m = 105$, $n = 31$
$\rightarrow K' : \{62, 93, 81, 88, 102, 37\}$.

Verschlüsselung: Die Nachricht wird in Blöcke in der Länge des Knapsacks aufgeteilt.

Entschlüsselung: Der Private Key (=superincreasing Knapsack) sowie n und m müssen bekannt sein. Somit kann n^{-1} ($nn^{-1} \equiv 1 \bmod m$) bestimmt und jeder Ciphertextwert mit $n^{-1} \bmod m$ multipliziert werden.

Beispiel: Für die Verschlüsselung muß der Ciphertext mit dem Pubic Key erzeugt werden. $K' : \{62, 93, 81, 88, 102, 37\}$:

$$\begin{array}{ll} 011000 & 93 + 81 = 174 \\ 110101 & 62 + 93 + 88 + 37 = 280 \\ 101110 & 62 + 81 + 88 + 102 = 333 \end{array} \quad \Rightarrow \text{Ciphertext } 174, 280, 333.$$

Für die Entschlüsselung werden Private Key n und m benötigt:
$K : \{2, 3, 6, 13, 27, 52\}$, $m = 105$, $n = 31$, $n^{-1} = 61$

$$\begin{array}{lclcll} 174(61 \bmod 105) & = & 9 & = & 3 + 6 & 011000 \\ 260(61 \bmod 105) & = & 70 & = & 2 + 3 + 13 + 52 & 110101 \\ 333(61 \bmod 105) & = & 48 & = & 2 + 6 + 13 + 27 & 101110 \end{array}$$

Für konkrete Anwendungen werden ungefähr 250 M_is verwendet, wobei ein einzelnes M_i mit etwa 200 bis 400 Bits und der Modul mit 100 bis 200 Bits kodiert werden. Für die Erzeugung dieser Zahlen werden oft Generatoren für Pseudozufallszahlen verwendet. Knapsack Probleme dieser Größenordnung können nicht mehr Brute Force gebrochen werden.

Sicherheit: Diese Art von Verschlüsselung ist sehr unsicher. Shamir und Zippel demonstrierten das Knacken des Merkle-Hellman Algorithmus mit einem Apple II in ca. 15 Minuten. Warum: die Methode der Erzeugung des schwierigen Problems ist die Sicherheitslücke !!

3.4.2 Pohling-Hellman Algorithmus

Der Pohling-Hellman Algorithmus ist dem RSA Verfahren sehr ähnlich. Er ist nicht symmetrisch, da für die Ver- und Entschlüsselung verschiedene Schlüssel verwendet werden. Da die beiden Schlüssel aber einfach von einander ableitbar sind, gehört der Algorithmus auch nicht zur Gruppe der Public Key Verfahren. Beide Schlüssel müssen geheim bleiben.

$$C = P^e \bmod n$$
$$P = C^d \bmod n$$

wobei $ed \equiv 1$ (modulo einer bestimmten Zahl).

n ist nicht als Primzahlprodukt definiert sondern bleibt Teil des geheimen Schlüssels. Wenn e und n bekannt sind, kann d berechnet werden. Wenn e oder d nicht bekannt sind, muß $e = \log_P C \bmod n$ berechnet werden, ohne P zu kennen.

3.4.3 Rabin Algorithmus

Die Sicherheit des Verfahrens von Rabin besteht in der Schwierigkeit Quadratwurzeln modulo einer zusammengesetzten Zahl zu bestimmen. Dieses Problem ist äquivalent zur Faktorisierung.

Wähle zwei Primzahlen p und $q \equiv 3 \bmod 4$. Sie sind der Private Key, $n = pq$ ist der Public Key.($M < n$).

Die Verschlüsselung wird mit
$$C = M^2 \bmod n$$
erzeugt.

Die Entschlüsselung ist möglich, wenn p und q bekannt sind. Mittels des Chinesischen Restsatzes können vier mögliche Lösungen bestimmt werden.

$$\begin{aligned}
m_1 &= C^{\frac{p+1}{4}} \bmod p \\
m_2 &= (p - C^{\frac{p+1}{4}}) \bmod p \\
m_3 &= C^{\frac{q+1}{4}} \bmod q \\
m_2 &= (q - C^{\frac{q+1}{4}}) \bmod q
\end{aligned}$$

$$\begin{aligned}
a &= q(q^{-1} \bmod p) \\
b &= p(p^{-1} \bmod q)
\end{aligned}$$

$$\begin{aligned}
M_1 &= (am_1 + bm_3) \bmod n \\
M_2 &= (am_1 + bm_4) \bmod n \\
M_3 &= (am_2 + bm_3) \bmod n \\
M_4 &= (am_2 + bm_4) \bmod n
\end{aligned}$$

Bei einem natürlich sprachlichen Text M ist das Bestimmen der richtigen Lösung sehr einfach: Welcher Text ergibt einen Sinn. Bei einem zufälligen bit-stream muß den Daten ein Header vorangestellt werden, der die Auswahl der richtigen Lösung ermöglicht. Dieser Header ist dann natürlich Teil der Message.

3.4.4 El Gamal Algorithmus

Die Sicherheit dieses Verfahrens besteht in der Schwierigkeit diskrete Logarithmen in endlichen Körpern zu berechnen. Dieses Verfahren wurde ursprünglich für digitale Unterschriften verwendet.

Zwei Primzahlen p sowie $q \in \boldsymbol{N}$, $x \in \boldsymbol{N}$, q, x beliebig und $< p$.

$$y = q^x \bmod p$$

y, q, p sind der Public Key, wobei q und p von mehreren AnwednerInnen verwendet werden können. x ist der Private Key.

Wähle K mit $(K, p-1) = 1$ und berechne $a = q^K \bmod p$.

Erw. Euklidsche Algorithmus: $M = (xa + Kb) \bmod (p-1)$, wobei b gesucht ist.

Die Unterschrift ist das Paar a und b. K muß geheim bleiben.

Um die Unterschrift zu verifizieren, muß folgendes berechnet werden:

$$y^a a^b \bmod p = q^M \bmod p$$

wobei

$$\begin{aligned} y^a a^b \bmod p &= y^{q^K} a^b \\ &= y^{q^K} q^{Kb} \\ &= q^{xq^K} q^{Kb} \\ &= q^{xa} q^{Kb} \end{aligned}$$

Jede Unterschrift benötigt ein neues K und muß zufällig gewählt werden.

Für die Verschlüsselung muß ein K mit $(K, p-1) = 1$ gewählt werden. $a = q^K \bmod p$
$b = y^K M \bmod p$
a und b sind der Ciphertext (mit doppelter Länge).

Für die Entschlüsselung muß

$$M = \frac{b}{a^x} \bmod p$$

berechnet werden.

$$\begin{aligned} a^x &\equiv q^{Kx} \bmod p \\ \tfrac{b}{a^x} &\equiv \tfrac{y^K M}{a^x} \\ &\equiv \tfrac{q^{xK} M}{q^{Kx}} \bmod p \\ &\equiv M \bmod p \end{aligned}$$

3.5 Digital Signature Algorithmen (DSA) und Identifikations Schemata

3.5.1 DSA

1991 wurde der **D**igital **Signature A**lgorithmus als Signatur Standard vorgeschlagen. Dies führte zu einem Sturm der Entrüstung in der Industrie, das diese auf Grund der der Verbreitung von RSA einen RSA basierten Standard bevorzugt hätte.

Bei der Entwicklung von DSA war so wie auch bei RSA die NSA beteiligt. Das Standardisierungs- verfahren wurde ohne öffentliche Ausschreibung durchgeführt. Diese Vorgehensweise schürte natürlich das Mißtrauen diverser Institutionen.

p ist Primzahl mit L-Bits, $512 \leq L \leq 1024$, $L = K64$.
q ist ein 160-Bit Primfaktor von $p - 1$
$g = h^{\frac{p-1}{q}} \bmod p > 1$, mit $h < p - 1$.
wähle $x < q$ und berechne
$y = g^x \bmod p$.

DSA ist ein dem El Gamal Algorithmus sehr ähnliches Verfahren. Es verwendet eine One-Way Hash Funktion $H(M)$, den sogenannten Secure Hash Algorithmus. p, q und g sind öffentlich. y ist der Public Key. x ist der Private Key.

Das zughörige Protokoll sieht folgendermaßen aus:

1. Alice generiert ein zufälliges $K < q$.
2. Alice generiert
   $r = (g^K \bmod p) \bmod q$

$s = (K^{-1}(H(M) + xr)) \bmod q$,
wobei r und s ihre Unterschrift darstellen. Sie schickt beides an Bob.

3. Bob berechnet
$$\begin{aligned} w &= s^{-1} \bmod q \\ u_1 &= (H(M)w) \bmod q \\ u_2 &= (rw) \bmod q \\ v &= ((g^{u_1} y^{u_2}) \bmod p) \bmod q \end{aligned}$$
Wenn v gleich r ist, ist die Unterschrift verifiziert.

Da r nicht von der Nachricht abhängt, können K, K^{-1} und r vorberechnet werden, um das tatsächliche Signieren zu beschleunigen. Der Standard enthält weiters eine Methode, die Primzahlen zu generieren. Das schließt die Verwendung gefährlicher Moduli aus. Mit speziellen Parametern kann DSA auch für El Gamal und RSA Verschlüsselung verwendet werden.

Sicherheit: Da nur ausgewählte Moduli verwendet werden, ist die Sicherheit höher als bei RSA. Sehr wesentlich für die Sicherheit ist die Verwendung eines guten Zufallszahlen Generators für K!

Geschwindigkeit: DSA ist beim Verifizieren wesentlich lansamger, beim Signieren (mit Verbesserungen) aber wesentlich schneller als RSA.

3.5.2 Feige-Fiat Shamir Identification Scheme

1986 beantragten die drei Autoren für ihren Algorithmus ein Patent bei der zuständigen Stelle. Auf Grund militärische Interesses an diesem Verfahren erließ das Patentamt eine sogenannte "secrecy order´´: Die Veröffentlichung gefährdet die öffentliche Sicherheit. Jeder der eine Veröffentlichung vornimmt kann mit Freiheitsenzug bis zu zwei Jahren oder mit einer Zahlung von \$ 10.000 bestraft werden. Das kuriose daran war, daß der Algorithmus bereits allgemein bekannt war, da die Autoren diesen bereits auf diversen Konferenzen vorgestellt hatten und in der Presse bereits Meldungen veröffentlicht worden sind. Der Bescheid des Patentamtes wurde dann nach zwei Tagen wieder zurückgezogen. Dieses Vorgegen zeigt aber, wie stark Politik und Militär in das Thema Kryptographie involviert sind.

Die Funktionsweise des Algorithmus selber soll zuerst an einem vereinfachten Verfahren erklärt werden.

Arbitrator wählt zufälligen Modul $n = pq$, $n \geq 512$ Bits $\rightarrow 1024$. v muß ein quadratisches Residuum modulo n sein. ($x^2 \equiv v \bmod n$ hat eine Lösung und v^{-1} existiert.)

Das zugehörige Protokoll sieht folgendermaßen aus:

TeilnehmerInnen: Victor, Peggy
Peggys Public Key: v
Peggys Private Key: s mit $s \equiv \sqrt{(v^{-1})} \bmod n$

Protokollablauf

1. Peggy wählt zufälliges $r < n$ und berechnet $x = r^2 \mod n$. Dann schickt sie x an Viktor.

2. Victor schickt Peggy ein zufälliges Bit b.

3. $b = 0$: Peggy schickt r.
   $b = 1$: Peggy schickt $y = rs$.

4. $b = 0$: Viktor überprüft, ob $x = r^2 \mod n$ ist. D.h. Peggy kennt $\sqrt(x)$.
   $b = 1$: Viktor überprüft, ob $x = y^2 v \mod n$ ist. D.h. Peggy kennt $\sqrt(v^{-1})$.

Das ist eine einzelne Runde - Akkreditierung - des Protokolls. Peggy und Viktor wiederholen das Protokoll, bis Viktor überzeugt ist, daß Peggy s kennt.

Kennt Peggy s nicht, kann sie r so wählen, daß sie Viktor täuscht, wenn er $b = 0$ oder $b = 1$ schickt, allerdings nicht beides. D.h 50 % Chance ! Bei t Runden aber $p = \frac{1}{2^t}$.

Der tatsächliche Algorithmus ist nun etwas aufwendiger:
n ist wie vorher. Für Peggys Schlüssel müsen K verschiedene Zahlen $v_1, \ldots, v_K$ gewählt werden. Diese v_i sind quadratisches Residuen modulo n und bilden den Public Key. Der Private Key ist das kleinste s_i, sodaß gilt: $s_i = \sqrt(v_i^{-1}) \mod n$.

Das zugehörige Protokoll sieht folgendermaßen aus:

TeilnehmerInnen: Victor, Peggy.

Protokollablauf

1. Peggy wählt ein zufällig $v < n$ und berechnet $x = v^2 \mod n$. Dann schickt sie x an Viktor.

2. Viktor schickt einen zufälligen K-Bit String $b_1 \ldots b_K$.

3. Peggy berechnet $y = r(s_1^{b_1} \cdot \ldots \cdot s_K^{b_K}) \mod n$ und schickt y an Viktor.

4. Viktor überprüft $x = y^2(v_1^{b_1} \cdot \ldots \cdot s_K^{b_K}) \mod n$.

Wenn dieser Ablauf t mal wiederholt wird, ist die Täuschungswahrscheinlichkeit $\frac{1}{2^{Kt}}$. Die Autoren schlagen für $K = 5$ und für $t = 4$ vor, um eine “vernünftige´´ Sicherheit zu erhalten. Je größer die Paranoia, desto größer können die Werte gewählt werden.

3.6 Key Exchange Algorithmen

3.6.1 Diffie-Hellman

Das Diffie-Hellman Verfahren ist ein Public Key Algorithmus (1976), der ebenso wie El Gamal oder DSA auf dem diskreten Logarithmen Problem beruht.

Für den geheimen Schlüssel werden n große Primzahlen und g benötigt. g ist eine Primitivwurzel modulo n. Diese Zahlen müssen nicht geheim sein. Das bedeutet, sie können auch über einen unsicheren Kanal bezogen werden und von einer größeren BenutzerInnengruppe geteilt werden.

Der Ablauf des Verfahrens kann am besten an Hand des zugehörigen Protokolles erklärt werden.

TeilnehmerInnen: Alice, Bob.

Protokollablauf:

1. Alice wählt eine zufällige große Zahl x und schickt Bob $X = g^x \bmod n$.
2. Bob wählt eine zufällige große Zahl y und schickt Alice $Y = g^y \bmod n$.
3. Alice berechnet $K = Y^x \bmod n$.
4. Bob berechnet $K' = X^y \bmod n$.

$K = K' = g^{xy} \bmod n$. Niemand kann das berechnen, weil nur n, g, X, Y über den Kanal geschickt wurden. Es müßte ein diskreter Logarithmus berechnet werden, um x oder y zu erhalten. K ist also der geheime Schlüssel, den beide unabhängig berechnet haben.

Bei dem ganzen Verfahren ist es wichtig, daß n sehr groß gewählt wird und auch $\frac{n-1}{2}$ eine Primzahl ist.

Das Verfahren läßt sich auf drei TeilnehmerInnen erweitern:

TeilnehmerInnen: Alice, Bob, Carol.

Protokollablauf:

1. Alice wählt x und berechnet $X = q^x \bmod n$. Dann schickt sie X an Bob.
2. Bob wählt y und schickt Carol $Y = q^y \bmod n$.
3. Carol wählt z und schickt Alice $Z = q^z \bmod n$.
4. Alice schickt Bob $Z' = Z^x \bmod n$.
5. Bob schickt Carol $X' = X^y \bmod n$.
6. Carol schickt Alice $Y' = Y^z \bmod n$.
7. Alice berechnet $K = Y'^x \bmod n$.
8. Bob berechnet $K = Z'^y \bmod n$.
9. Carol berechnet $K = X'^y \bmod n$.

$K = q^{xy^2} \bmod n$! Analog läßt sich das Verfahren auch für vier oder mehr TeilnehmerInnen ausbauen.

3.7 Elliptic Curve Crypto Systems

Der große Vorteil dieser Verfahren liegt darin, daß sehr viele der bisher besprochenen Kryptoalgorithmen in diesem Framework gleich sicher aber mit wesentlich kürzeren Schlüsseln realisiert werden können, d.h. die Modulo Operationen können schneller durchgeführt werden (und sind daher auch nicht so anfällig gegen sog. "timing-attacken"). Diese Eigenschaften machen solche Kryptosysteme interessante Kandidaten für mobile environments (z.B. Handy-verschlüsselung). Der Grund für die "kürzeren Schlüsseln" liegt in der Tatsache begründet daß noch keine effizienten Algorithmen zur Lösung des "diskreten Log Problems" auf elliptischen Kurven gefunden wurden. Die Parameterwahl ist aber wesentlich aufwendiger als bei klassischen Verfahren und muß auf leistungsfähigen Architekturen durchgeführt werden.

Elliptische Kurven sind keine Ellipsen (siehe Graphik) ! Der Name kommt daher, daß man mit ähnlichen Ausdrücken den Umfang von Ellipsen berechnet. Eine elliptische Kurve wird durch folgende Gleichung beschrieben:

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

. Es gibt auch den sog. Unendlichpunkt 0, der als Nullelement der Addition auf solchen Kurven definiert wird. Liegen drei Punkte auf einer elliptischen Kurve auf einer Geraden ist ihre Summe 0 (d.h. auch daß eine Garade eine elliptische Kurve in genau drei Punkten schneidet). Eine vertikale Gerade schneidet so eine Kurve in zwei Punken mit den selben x-Koordinaten $P_1(x, y)$ und $P_2(x, -y)$ und im Unendlichpunkt 0. Daher ist $P_1 + P_2 + 0 = 0$ und daher $P_1 = -P_2$. Um zwei Punkte mit unterschiedlichen x-Koordinaten zu addieren verbindet man diese und sucht den Schnittpunkt der Gerade mit der Kurve P_1. Dann ist $Q + R + P_1 = 0$ und somit $Q + R = -P_1$. Um einen Punkt Q zu verdoppeln, wird die Tangente an Q gezogen (denn die Tangente verbindet die "beiden" Punkte Q und Q) und der Schnittpunkt S mit der Kurve gesucht. Damit ist $Q + Q + S = 0$ und somit $2 * Q = -S$ womit eine Multiplikation auf einer elliptischen Kurve definiert ist. Diese Multiplikation ist das Analogon zur Exponentiation in klassischer Muldulararithmetik, das Analogon zum diskreten Log. Problem ist die Frage bei bekannten Q und k*Q das k zu ermitteln (was entsprechned schwierig ist).

Für Kryptographie auf elliptischen Kurven betrachtet man eine eingeschränkte Klasse von elliptischen Kurven, die auf endlichen Körpern definiert sind. Man wählt zwei nicht-negative Integer $a, b < p$ mit p prim sodaß

$$4a^3 + 27b^2 (\pmod p)) \neq 0$$

. $E_p(a, b)$ ist dann die elliptische Gruppe modulo p deren Elemente (x,y) Paare nicht-negativer Integer sind und $x, y < p$ für die gilt:

$$y^2 \equiv x^3 + ax + b \pmod p$$

.

Als Beispiel, betrachten wir $p = 23$ und die Kurve $y^2 = x^3 + x + 1$ (somit $a = b = 1$). Obige Bedingung für a, b ist erfüllt. Für die elliptische Gruppe $E_{23}(1, 1)$ sind nun nur die ganzzahligen Punkte die die Gleichung modulo 23 von Interesse. Um diese Punkte zu ermitteln, wird für alle $x < p$ die rechte Seite des Ausdrucks berechnet - falls der Ausdruck eine Quadreadwurzel modulo p besitzt, ist der Punkt (und sein negativer Gegenpart) ein Element von $E_p(a, b)$. Die Regeln für Addition in $E_p(a, b)$ sind die exakte Umsetzung der geometrischen Regeln von vorher. Die Addition von $P = (x_1, y_1)$ und $Q = (x_2, y_2)$ sodaß $P + Q = (x_3, y_3)$ ist wie folgt definiert:

$$x_3 \equiv \lambda^2 - x_1 - x_2 \pmod p \quad \text{und} \quad y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod p$$

. wobei $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ für $P \neq Q$ und $\lambda = \frac{3x_1^2 + a}{2y_1}$ für $P = Q$. Berechnen Sie für $P = (3, 10)$ und $Q = (9, 7)$ $P + Q$ und $2 * P$!

Um die Funktionesweise eines solchen Kryptosystems zu demonstrieren folgt eine kurze Beschreibung eines Analogons zum Diffie-Hellman key Exchange Algorithmus: Zuerst müssen p, a, b entsprechend gewählt werden. Dann wird ein sog. "Generator-Punkt" $G = (x_1, y_1)$ gewählt sodaß $n * G = 0$ erst bei großem n auftritt. $E_p(a, b)$ und G sind öffentlich. Der key Exchange läuft dann ab wie folgt:

- Alice wählt Integer n_A kleiner als n (ihr private Key) und berechnet den Key der über den unsicheren Kanal zu Bob geht: $P_A = n_A * G$.
- Ebenso wählt Bob einen private Key n_B und berechnet analog P_B, den er an Alice schickt.
- Alice generiert den geheimen session Key durch: $K = n_A * P_B$, analog berechnet Bob $K = n_B * P_A$.

Das geht sich aus weil

$$n_A * P_B = n_A * (n_B * G) = n_A * n_B * G = n_B * n_A * G = n_B * (n_A * G) = n_B * P_A$$

Damit ein Angreifer erfolgreich sein könnte müßte er aus P_A oder P_B und G eines von n_A oder n_B berechnen können (was eben nicht in endlicher Zeit geht). Um aus dem generierten Zahlenpaar einen Schlüssel für klassische Verschlüsselung zu generieren, muß daraus eine Zahl gebildet werden.

Gleiche Sicherheit zu klassichem RSA wird erreicht durch:

Elliptic curve	RSA
Generator-Punkt (Bits)	Modulus (Bits)
106	512
132	768
160	1024
224	2048

3.8 Probabilistische Kryptographie und Quantenkryptographie

Probabilistische Algorithmen enthalten Zufall, sie generieren aus einem Plaintext verschiedene Ciphertexte. Im klassischen Fall hat man $C = E_k(M)$ und $C' = E_k(M')$. Ist nun $C' = C$ so folgt dass $M' = M$ ist. Wird probabilistische Kryptographie verwendet kann man nicht beweisen, daß ein bestimmter Ciphertext zu einem bestimmten

Plaintext gehört. Adaptive chosen Plaintext Attacken werden so sinnlos. Als Beispiel sei hier kurz ein einfaches Beispiel das den Blum-Blum-Shub Generator verwendet erklärt. Man verwendet wie bei RSA p, q als private key und $n = pq$ als public key. p, q müssen bestimmte Eigenschaften haben ("Blumsche Zahlen"). Man wählt eine nicht durch p, q teilbare Zufallszahl x und berechnet:

$$x_0 = x^2 \pmod{n} \quad , x_1 = x_0^2 \pmod{n} \ldots$$

Die Folge der niedrigsten Bits der Folge $x_0, x_1, \ldots$ wird als One-time-Pad verwendet, der Wert x_{t+1} wird an den Ciphertext angehängt (bei Plaintextlänge von t Bit). Kennt man p, q kann man mit dieser Information $x_0, x_1, \ldots$ rekonstruieren (im Wesentlichen durch Wurzelziehen modulo n). Da x zufällig gewählt ist, liefert jede Verschl"usselung einen anderen Ciphertext.

In der Quantenkryptographie wird mit der Polarisierung von Photonen gearbeitet. Die Polarisierung kann 4 verschiedene Ausprägungen haben: oben/unten, rechts/links, hauptdiagonal, nebendiagonal. Ein Polarisationsfilter erlaubt manchen Photonen unveränderten Durchtritt und bei manchen wird die Polarisierung verändert. Es gibt Rechtecksfilter (für oben/unten und rechts/links Polarisation) und Diagonalfilter. Wird ein hauptdiagonal oder nebendiagonal polarisiertes Photon durch einen Rechtecksfilter geschickt, wird dessen Polarisation zufällig auf oben/unten oder rechts/links verändert.

Angenommen Alice und Bob wollen über einen unsicheren Kanal sicher kommunizieren. Um das zu ermöglichen muß zuvor ein gemeinsamer Key ausgetauscht werden - dies soll mit Hilfe von Quantenkryptographie wie folgt vor sich gehen:

- Der Key ist ein Photonenstrom, jedes dieser Photonen repräsentiert ein Bit, die Polarisierung definiert 0 oder 1.
- Alice wechselt jetzt beim Senden zufällig zwischen rechteckiger und diagonaler Polarisierung.
- Bob wechselt beim Empfangen ebenfalls zufällig zwischen rechteckiger und diagonaler Polarisierung. Nur wenn sein Filter korrekt ist (also für das aktuelle Photon identisch zur Einstellung von Alice) kann er die korrekte Information auslesen.
- Nun kommunizieren Alice und Bob über einen möglicherweise unsicheren Kanal und Alice teilt Bob mit für welches Photon sie welche Art von Polarisierung verwendet hat, nicht jedoch 0 oder 1 und Bob sagt wo er die richtige Einstellung hatte.
- Für die Photonen bei denen beide die identische Einstellung benutzt haben, kann Bob die Information extrahieren (also bei ca. 50% der gesendeten Photonen). Diese dienen nun als Key oder sogar one-time Pad.

Ein möglicher Angreifer erfährt über den unsicheren Kanal zwar die Positionen an denen die Einstellungen von Alice und Bob identisch waren, seine eigenen Einstellungen müssen aber ebenso zufällig sein und stimmen mit denen von Bob sicher nicht überein. Daher werden die Photonen an den Stellen wo die Einstellungen des Angreifers und von Bob verschieden sind verändert und der Schlüssel damit nutzlos.

Zusätzlich erlaubt Quantenkrysptographie Intrusion Detection, also das Aufspüren eines möglichen Angreifers. Wenn Eve beim Lauschen am Photonenkanal an einer Position wo Bob korrekt mißt den falschen Filter verwendet, verändert sie die Polarisierung des Photons. Alice und Bob müssen nun nur noch ein Subset des Keymaterials vergleichen (über den unsicheren Kanal) – stimmt es überein und ist das Subset ausreichend groß, hat niemand gelauscht.

Chapter 4

Netzwerksicherheit

Grundsätzlich können Sicherheitsfeatures auf unterschiedlichen Ebenen realisiert werden.

- Applikationsebene: Vorteil ist, daß nur die Applikationen sicher sind die diese Eigenschaft auch wirklich brauchen (kein overhead), andererseits muß sich jede Applikation bzw. auch jeder Nutzer um die Sicherheit kümmern.
- Netzwerkebene: Vorteil ist die völlige Transparenz, nachteil der enorme overhead da unabhängig von der Applikation Sicherheitsfeatures eingesetzt werden.
- Linkebene: v.a. im militärischen Bereich verwendet, hier ist jeder Netzwerklink mit Kryptographischer Hardware versehen. Die Sicherheit ist exzellent, solange hinter den Links "sicheres" Gebiet ist. Im Internet natürlich nicht einsetzbar.

Infrastrukturebene oder der Anwendungsebene zur Verfügung gestellt werden.

4.1 Sicherheit auf IP und DNS Ebene

Die Hauptprobleme der existierenden Infrastruktur liegen vor allem in zwei Bereichen:

- Internet-Protokoll (IP) als Basisprotokolls des gesamten Datenverkehrs.
- Domain-Name-System (DNS) als Basis für alle Adreßumsetzungen

Sämtliche Anwendungen sind von einer korrekten Funktionsweise des IP-Basisprotokolls und der Adreßumsetzung abhängig.

4.1.1 Sicherheit des IP Protokolls: IPSec

Die Verbesserung des IP-Protokolls bezüglich einer Verschlüsselung und Authentisierung ist bereits seit mehreren Jahren im Gespräch. Für Version 4 (also das derzeit aktuelle Protokoll) arbeitet die IETF-Arbeitsgruppe "Internet Protocol Security Protocol (ipsec)” bereits seit 1993 an einer Lösung. Für Version 6 (IPng, die Nachfolgeversion) wurden diese Sicherheitsmechanismen bereits von Anfang an gefordert. Diese Mechanismen werden als IPSec bezeichnet. Anwendungsgebiete sind z.b.:

- Sichere Verbindung von Zweigstellen übers Internet
- Sicherer remote Access
- Verbesserung von e-commerce Sicherheit
- ....

Klassisches Szenario von IPSec Benutzung ist eine Organisation, die mehrere LANs an verschiedenen Orten betreibt. Nicht-abgesicherter IP Verkehr wird innerhalb der LANs eingesetzt, zur Kommunikation zwischen den LANs über ein WAN oder Internet werden IPSec Protokolle verwendet, die in den entsprechenden Routern oder Firewalls laufen.

Der große Vorteil von Sicherheit die über IPSec bereitgestellt wird ist daß diese Art von Sicherheit für den User völlig transparent ist, da sie unterhalb des Transport Layer (TCP,UDP) angesiedelt ist. Keinerlei Anwendungssoftware muß angepasst werden. Nachteil ist die höhere Verarbeitungskomplexität für jede Kommunikation (also auch für Urlaubsgrüße).

Sowohl für IPv4 und IPv6 sind die Sicherheitsfeatures als Extension Headers hinter dem Haupt IP Header implementiert. Es gibt den Authentication Header (AH - Authentifizierungsprotokoll: hier werden die Daten und die invarianten Felder des äusseren IP Headers mit einem MAC - wegen der Geschwindigkeit statt einer Signatur - authentifiziert) und den Encapsulating Security Payload (ESP - kombiniertes Verschlüsselungs/Authentifizierungsprotokoll, bei ESP Authentifizierung wird der Header nicht authentifiziert sondern nur die Payload) Header. Ein Schlüsselkonzept ist das der Security Association (SA) – das ist eine Einwegbeziehung zwischen Sender und Empfänger und wird durch die IP-Empfangsadresse und den Security Parameter Index (SPI) festgelegt. In jeder IPSec Implementierung gibt es eine Datenbank, die die Parameter festlegt, die zu einer SA gehören (AH und ESP Information bezüglich Algorithmen, Schlüssel, Gültigkeitsbereich von Schlüsseln, IPSec Protocol Mode - Tunnel vs. Transport u.s.w.).

Die beiden Modi Transport und Tunnel unterscheiden sich bezüglich der Anwendungsszenarien: während der Transport Mode v.a. für sichere End-to-End Kommunikation verwendet wird, den Header unverändert lässt und die IP Payload schützt, wird beim Tunnel Mode das gesamte IP Paket geschützt und mit einem neuen IP Haeder versehen und für das Tunnelling zwischen Routern verwendet.

Der Standard Verschlüsselungsalgorithmus ist DES in CBC Modus, aber auch IDEA, triple DES und weitere Algorithmen können verwendet werden. Zur Authentifizierung werden MD5 und SHA1 verwendet.

Für spezielle Anwendungsszenarien können verschiedene SAs ineinander geschachtelt werden, das Key Management verwendet eine Verbesserung des Diffie-Hellman Key Exchange, das auch ellipische Systeme zuläßt. IPSec nimmt an, daß bereits eine SA etabliert worden ist stellt aber keinen Mechanismus zur Verfügung eine SA zu kreieren. IPSec führt die paketorientierte Verarbeitung durch, während IKE das Etablieren der

SAs übernimmt. IKE schafft einen authentifizierten, sicheren Tunnel und "verhandelt" die SAs zwischen den Teilnehmern.

- Authentifizierung:
  - Pre-shared keys: identische Keys sind auf den Hosts vorinstalliert; IKE Authentifizierung geschieht durch Berechnung und Senden eines verschlüsselten Hash Wertes von Daten die den vorinstallierten Key enthalten, wenn beide das gleiche generieren können, müssen beide im Besitz des gleichen Keys sein.
  - Public-Key Methode: jeder Teilnehmer generiert Zufallszahl und verschlüsselt diese mit dem public Key des Anderen (RSA). Jeder Teilnehmer generiert dann mit seinem private Key einen verschlüsselten Hash der Zufallszahl des Anderen und sendet diese. Kann auch direkt mit digitalen Signaturen gemacht werden (DSS).
- Key Exchange: Diffie-Hellman

IPSec wird stark kritisiert. Die hohe Komplexität des Verfahrens, Resultat des Entstehungsprozesses (ein Kommittee wählte nicht aus Submissions aus sondern bastelte IPSec selbst was viele Kompromisse notwendig machte), bedingt eine hohe Anfälligkeit gegenüber Fehlkonfiguration. Neben der Kritik an der "Unlesbarkeit" der Dokumentation wird eine Elimination des Transport Modes und des AH vorgeschlagen (der Overhead im Tunnel Mode könnte leicht durch einen speziellen Headerflag ausgeglichen werden, der angibt, wenn beide IP Header gleich sind und ESP müsste nur um Header Authentifizierung erweitert werden um volle AH Funktionalität zu erreichen).

4.1.2 Authentifizierung im Domain Name System: DNSSEC

Das Domain-Name-System kommt immer dann zum Einsatz, wenn IP-Adressen in Rechnernamen umgesetzt werden müssen oder eine umgekehrte Umsetzung erforderlich ist. Durch einen Angriff kann die Adreßumsetzung für ganze Domains übernommen werden, so daß weitreichender Einfluß auf die Funktion des gesamten Netzes genommen werden kann. Mit Hilfe solcher Manipulationen können Pakete auch um- und fehlgeleitet werden, so daß sich ein Angreifer in den Besitz der übertragenen Informationen bringen können oder sich sogar als fremde Rechner ausgeben können. In diesem Bereich sind bereits viele Angriffe bekannt geworden (DNS Spoofing) und die Schwachstellen wurden auch dokumentiert. Die grundlegenden Probleme sind jedoch ohne eine Änderung des gesamten DNS nicht lösbar da das System auf der Öffentlichkeit der Daten basiert und auf das korrekte Arbeiten aller Teilnehmer vertraut. Auch zum Schutz des DNS werden darum kryptographische Methoden, die auf Public-Keys basieren, entwickelt. Ziel ist jedoch im Gegensatz zu IPSec nicht Verschlüsselung sondern die Gewährleistung der Integrität (keine Veränderung) und Authentizität (Daten stammen wirklich aus der vorgegebenen Quelle) der DNS Daten. Diese Ziele werden durch 3 Services erreicht:

- Key distribution: erlaubt nicht nur das Beziehen des public Key eines DNS Namens um die Authenzität der DNS Zonen Daten zu prüfen sondern mit diesem

Schema können grundsätzlich alle Keys die mit einem DNS Namen verbuden sind administriert werden, auch für ganz andere Zwecke.

- Data origin authentication: das Resource Record (RR) Set einer DNS Zone wird kryptographisch signiert (verschlüsselter Hash des RRSet). Der Hash wird mit dem private Key des primären DNS Server der Zone verschlüsselt. Verifikation geschieht einerseits durch Hashen des RRSet und andererseits durch Entschlüsselung des verschlüsselten Hash mit dem public Key des primären DNS Servers der Zone (der mit Hilfe der Key distribution Funktionalität verfügbar ist).
- DNS Transaction and Request Authentifizierung: hier geht es darum ob eine Antwort eines DNS Servers tatsächlich die ursprüngliche Frage beantwortet und ob die Antwort von dem Server kam der auch befragt wurde. Das wird durch eine Signatur einer Concatenation von Frage und Anwort erreicht die alle notwendigen Daten für Verifikation enthält. Zusätzlich gibt es Mechanismen für starke Authentifizierung des dynamischen Updates sekundärer DNS Server.

Um diese Services realisiern zu können, wurden neue RRs definiert:

- Key RR: hier ist der (die) public Key(s) für einen DNS Namen abgelegt. Zusätzlich wird der zugehörige Algorithmus (z.B. RSA, MD5, Diffie-Hellman, DSA, elliptic curve) und der Protokolltyp (TLS, email, DNSSEC, IPSec) definiert. Nicht inkludiert sind Key Zertifikate, die im CERT RR abgelegt sind.
- SIG RR: enthält die Signatur des RRSet und die notwendigen algorithmischen Informationen sowie den Gültigkeitszeitraum der Signatur.
- NXT RR: dient zur Authentifizierung nicht existierender RRSets, falls ein DNS Name nicht existiert oder der RR Typ in einer Anfrage für einen Namen nicht existiert. Dies wird durch eine vordefinierte Anordnung der RRSets erreicht.

In Security Aware DNS-Servern gibt es zwei Formen von Authentifizierung:

- Off-line: der private Key der Zone darf keinesfalls komprimittiert werden, er ist daher nicht on-line verfügbar und wird jedesmal zur Erstellung der SIG RR generiert/retrieved.
- On-line: Authemtifizierung von DNS Transaktionen geschieht häufiger - dies wird mit anderen Keys, den Server-Keys gemacht, welche daher on-line verfügbar sein müssen.

Es bleibt die Frage, wie die die public Keys verteilt werden. Wird DNS selbst durch Abfrage des key RR verwendet stellt sich die Frage wie die Authenzität der public Keys im Key RR gewährleistet wird. Diese müssen dann von der Masterzone signiert sein, wobei die Frage entsteht, wie die Authentizität deren public Keys gewährleistet ist. Man braucht daher irgendwann eine “trusted zone”. Statische Konfiguration der Keys skaliert wiederum nicht gut - wird ein Key geändert, müssen alle Konfigurationen geändert werden.

Weitere Informationen sind im Internet verfügbar:

- IETF WG DNS Security (DNSSEC)
  `http://www.ietf.cnri.reston.va.us/html.charters/dnssec-charter.html`
- Internet Draft http://www.ietf.org/ids.by.wg/dnssec.html
- RFC2065: Domain Name System Security Extensions / D. Eastlake; C. Kaufman. - Jan. 97.

4.2 Sicherheit auf der Anwendungsebene

4.2.1 Authentifizierung in verteilten Systemen mit Kerberos

Kerberos (der Wächter des Eingangs zur Unterwelt in der griechischen Mythologie) wurde am MIT entwickelt und dient zur Authentifizierung in offenen verteilten Systemen, wie z.B. ein Netzwerk von Workstations. In so einer Umgebung kann man nicht darauf vertrauen, daß eine Workstation ihre Benutzer geg. Netzwerk Services korrekt identifiziert:

- Ein fremder Benutzer kann sich Zugang zur einer Workstation verschaffen und vortäuschen ein anderer zu sein
- Die Netzwerkadressen einer Workstation kann gefälscht sein
- Replay Attacken geg. Server

Folgende drei Sicherheits-Systemarchitekturen wären denkbar:

- Jede individuelle Client Workstation und jeder Server sorgt für korrekte user identification
- Clients müssen sich geg. Server authentifizieren, aber den Clients wird bezüglich der user ID vertraut
- Die User müssen sich für jeden Servise authentifizieren, ebenso die Server geg. den Clients

Kerberos verwendet einen zentralen Ansatz mit einem Autentifizierungs Server, der User und Server gegenseitig authentifiziert. Kerberos verwendet auch keine public-key Algorithmen sondern beruht ausschließlich auf symmetrischer Kryptographie.

Kerberos Version 4 verwendet DES in einem relativ komplizierten Protokoll zur Authentifizierung. Der Authentifizierungsserver (AS) kennt die Passwörter aller User und speichert diese in einer zentralen Datenbank. Zusätzlich teilt der AS einen geheimen Key mit jedem Server im Netzwerk. Um von einem Client aus einen Request an einen Server schicken zu können, muß vom AS ein sogennantes Ticket geholt werden. Mit dem Protokoll in erster Variante gibt es zwei Hauptprobleme:

- Wenn jedes Ticket nur einmal benutzt werden kann muß der User sehr oft sein Passwort eingeben.

- Das Passwort wird als Plaintext übertragen

Um diese Probleme zu lösen wird ein neuer Server eingeführt, der sog. "Ticket Granting Server" (TGS), der für jeden Service ein Ticket pro Session ausstellt, wenn sich der user beim AS authentifiziert hat. Das Ticket granting Ticket wird vom Client Modul während der gesamten Session aufbewahrt und benutzt wenn immer ein neuer Service benötigt wird. Diese Tickets für einen bestimmten Service werden ebenfalls die ganze Session aufbewahrt. Hier wird nun kein Passwort mehr übertragen, denn der AS verschlüsselt das Ticket mit einem aus dem Passwort abgeleiteten Key, der User muß nur auf seinem Client zur Entschlüsselung des keys sein Paassswort eingeben. Um Replay Attacken zu erschweren hat das Ticket noch eine Timestamp und eine Lebensdauer. Die Tickets sind noch zusätzlich Serverseitig verschlüsselt. Es bleiben jedoch noch zwei Probleme:

- Wenn die Lebensdauer eines Tickets lang ist (was bequem ist), ist die Gefahr von Replay größer - man will also beweisen können, daß die person die das Ticket benutzt auch die "richtige" person ist
- Es soll auch die Möglichkeit geben, daß sich Server bei Usern authentifizieren müssen (es könnte jemand vortäuschen, ein Server zu "sein").

Um das erste Problem zu lösen wird vom AS an TGS und User ein session Key verteilt, mit dem der user immer seine Zugehörigkeit zu seinem Ticket beweisen kann. Auch die Authentifizierung von Servern wird über einen Key geregelt.

Ein volles Kerberos System besteht aus einem Kerberos Server, mehreren Clients und mehreren Applikations-Servern und den folgenden Daten:

- Der Kerberos Server muß alle User IDs und einen hash der jeweiligen Passwörter in seiner Datenbank haben. Alle User müssen beim Server angemeldet sein.
- Der Kerberos Server muß mit jedem Applikations-Server einen Key teilen. Alle solchen Server müssen beim K-Server angemeldet sein.

Eine solche Umgebung nennt man "Realm". Kerberos unterstützt aber auch inter-Realm Authentifizierung. In so einem Fall muß jeder K-Server mit dem Kollegen im anderen Realm einen Schlüssel teilen und die jeweiligen K-server müssen sich gegenseitig trauen, daß der jeweilig andere die User korrekt authentifiziert und die Apllikations-Server im fremden Realm müssen natürlich den fremden K-server trauen. Die Abwicklung funktioniert dann über vom TGS augestellte Tickets für den fremden TGS.

In Kerberos Version 5 wird die Einschränkung auf DES und IP aufgehoben, effizientere Inter-Realm Kommunikation ermöglicht, Lifetimes können länger sein.

Probleme bei Kerberos sind

- Schlechte Skalierbarkeit
- Kerberos ist eine Komplettlösung, das Einbinden in andere Applikationen ist aufwendig

- Der TGS muß immer verfügbar sein

Alternative Systeme sind Distributed Computing Environment (DCE – von der Open Software Foundation, ähnlich wie Kerberos), SESAME (Europäisches Projekt) und CORBA.

4.2.2 Sicherheit von GSM Netzen

In GSM Netzen wird mit dem A5 Algorithmus verschlüsselt, der als eher schwacher Algorithmus gilt (politische Diskussion bei der Entscheidung und der Kryptoanalyse von A5). Das ist ein Schieberegistergenerator mit linearer Rückkopplung (alle Crays haben einen Spezialbefehl zur LFSR Auswertung zur "staatlichen" kryptoanalyse). A5 benutzt tatsächlich 3 LFSR mit unterschiedlichen Registerlängen.

In jeder SIM Karte ist ein Chip mit einer Seriennummer und einer geheimen Zahl Ki, die nicht ausgelesen werden kann (von Amateuren :-). Im Chip existieren auch zwei Algorithmen A3 und A8 die jeder Netzbetreiber geheim hält. Ki haben auch die Netzbetreiber in ihren Rechnern gespeichert. Startet ein Teilnehmer einen Anruf, so sendet der Chip die Seriennummer - das Netzwerk sucht die entsprechende Zahl Ki und schickt eine Zufallszahl RAND als Antwort. Mittels A3 berechnet der Chip aus RAND und Ki eine 32-Bit Antwort SRES und schickt diese an die Basisstation. Diese verifiziert das, ist alles korrekt, geht der Anruf o.k. Aus RAND und Ki berechnen nun der Chip und die Basisstation einen 64-Bit Sessionkey K_c unter Verwendung von A8. Dieser Schlüssel wird für A5 Ent- und Verschlüsselung benutzt. Durch die korrekte Authentifizierung ist der Schlüssel natürlich identisch. Durch die Verwendung von verschiedenen Schlüsseln für jedes Gespräch relativiert sich die Unsicherheit von A5, da die Entschlüsselung sehr schnell sein müsste (1997 hätte ein Rechner mit 50 Spezialchips ca. 3 Stunden zum Knacken gebraucht - also auch nur eine Frage des Geldes !).

In ausländischen Netzen werden natürlich andere Algorithmen A3 und A8 verwendet - hier muß RAND im Heimatnetz geholt werden und auch SRES und K_c werden dort berechnet. Daher werden auch im Ausland Gespräche ins dortige Festnetz zum Auslandstarif berechnet, da ja die Parameter für das Gespräch im Heimatnetz geholt werden müssen.

Insgesamt erscheint das Gesamtkonzept ziemlich sicher, mit der Ausnahme, daß die Verschlüsselung an der Basisstation endet und beginnt, d.h. es gibt keine verschlüsselte End-to-End Kommunikation. An den Basisstationen horcht dann auch "Vater Staat". Ein weiteres Problem ist die mögliche Positionsbestimmung mit den momentanen Protokollen.

Im UMTS Bereich wird es weiterhin keine uneingeschränkte End-to-End Verschüsselung geben, nicht zuletzt wegen polizeilicher Bedürfnisse, die bekannt schwachen Elemente wie A5 werden durch stärkere Verfahren ersetzt.

4.2.3 WWW-Sicherheit

Durch das WWW bewegen sich auch Leute in Netzwerken die das ohne graphische Oberfläche sicher nicht getan hätten – die Mehrzahl der User hat keine Ahnung von möglichen Sicherheitsproblemen.

4.2.3.1 SSL und Transport Layer Security

SSL (Secure Sockets Layer) ist ein Protokoll, das zur sicheren Datenübertragung und zur Authenifizierung zwischen Server und Client von der Firma Netscape entwickelt wurde. Dabei wird zwischen dem Transport Layer (TCP) und dem Anwendungsprotokoll (HTTP) eine zusätzliche Schicht eingetragen. Dies würde im OSI Modell dem Presentation Layer entsprechen.

Dieser Layer bietet das selbe Interface wie TCP und kann daher von allen TCP/IP basierten Protokollen genutzt und dadurch gesichert werden. Für den Einsatz von SSL für HTTP wurde der Port 443 reserviert.

SSL bietet:

Kryptographische Sicherheit

Offenheit: Die Spezifikation des Protokolls ist bekannt un ermöglicht daher die Zusammenarbeit unterschiedlicher Applikationen.

Erweiterbarkeit: Die Einbindung neuer Kompressions- und Kryptographiemethoden ist ohne Protokolländerung möglich.

Effizienz: Das Caching von Verbindungen ist möglich.

SSL verwendet zum Verbindungsaufbau asymmetrische Verschlüsselung. Die Datenübertragung selber erfolgt über ein symmetrisches Verfahren (DES, RC4). Die hierfür benötigten Schlüssel werden beim Verbindungsaufbau ausgetauscht (RSA, DSS). Zur Überprüfung der Datenintegrität eines Dokumentes wird ein Hashing Verfahren (SHA, MD-5) verwendet.

Das Problem der Zuordnung des Public Key zu einem bestimmten Server wird über Certification Authorities (CA) gelöst. Diese stellen für einen Server ein Zertifikat aus, das die ID des Servers sowie dessen Public Key enthält. Das Zertifikat wird mit dem Private Key der CA unterschrieben und ist somit genauso vertrauenswürdig wie diese Stelle selbst. Der Client kann nun mit dem Public Key der CA die Echtheit des Zertifikates überprüfen.

SSL besteht aus zwei Schichen:

Steuerprotokollschicht: Dies Schicht ist verantwortlich für

- Verbindungsaushandlung: Vereinbarung der Kompressions- und Verschlüsselungsverfahren.
- Zertifikataustausch.
- Schlüsselaustausch für die symmetrische Übertragung.
- Überprüfung der Verbindung.

Im Moment ist das sogenannte Hand Shake Protokoll in dieser Ebene in Verwendung. RSA und Diffie-Hellman werden in diesem Protokoll verwendet. Die drei übrigen Protokolle in dieser Schicht sind das Change Cipher Spec Protocol, das Alert Protocol und das Application Data Protocol.

Record-Layer-Schicht: Dies ist das Interface zum TCP-Protokoll und ist für das Sichern, Versenden, Empfangen und Überprüfen von Daten zuständig. Heißt auch SSL Record Protocol. Die Daten werden beim Versenden zuerst fragmentiert und komprimiert, jedenfalls wird der Datenstrom für SSL völlig transparent übertragen. Danach erhält jeder Block eine Nummer, es wird mittels Hash-Verfahren eine Prüfsumme erzeugt und der Block symmetrisch verschlüsselt. Beim Empfangen werden die selben Tätigkeiten in umgekehrter Reihenfolge durchgeführt. MD-5, SHA-1, IDEA, DES und andere bekannte Algorithmen werden verwendet.

Es gibt sowohl kommerzielle als auch Freeware Implementierungen. Letzter sind auch im Quelltext erhältlich. Da starke Kryptographie in Amerika unter das Exportverbot von Waffen fällt, sind die in Europa erhältlichen aber in Amerika erstellten Produkte mit einer schwächeren Verschlüsselung ausgestattet. (z.B. 40-Bit Schlüssel statt 128),

Transport Layer Security (TLS) ist praktisch identisch mit SSLv3 außer einigen Erweiterungen - diese Erweiturungen verhindern ein direktes Kommunizieren zwischen SSL und TLS daher wurde TLS eigenständig standardisiert, mit Rckwärtskompatibilität zu SSLv3.

4.2.3.2 SHTTP

SHTTP (Secure Hyptertext Transport Protokol) wurde von der Terisa Systems/NCSA entwickelt und bietet die Möglichkeit individuelle Transaktionen abzusichern. Dieses Protokoll ist im Application Layer angesiedelt und kann daher von anderen Anwendungen nicht benutzt werdne. SHTTP ermöglicht also den Zugriff und Transfer einzelner Dokumente oder etwa die Übermittlung von Daten zum Abschluß eines Online-Geschäftes. Die Sicherheit ist immer an das Dokument bzw. Übertragung gebunden und es können alle erforderlichen Sicherheitsanforderungen erfüllt werden.

SHTTP ist "nur" eine Ergänzung (Privacy Enhancements) des HTTP Protokolls, wobei die http Nachricht verschlüsselt wird. Durch die Ergänzung wird es im Rahmen der Anwendung möglich, individuell über die zu gewährleistenden Sicherheitsanforderungen zu entscheiden. So können weiterhin bestimmte Dokumente ohne weitere Absicherungen übertragen werden, während andere Dokumente nur von zwei oder drei Personen gelesen werden können.

4.2.3.3 SET

Im Zusammenhang mit dem über elektronische Medien abgewickelten Handel (Electronic Commerce) werden auch Protokolle entwickelt, die die Sicherheit finanzieller Transaktionen gewährleisten sollen. Zunächst wurden zwei unterschiedliche Ansätze verfolgt:

- Secure Transaction Technology Protocol (STT) -Zusammenarbeit von VISA und Microsoft
- Secure Electronic Payment Protocol (SEPP) - Beteiligung von Mastercard, Netscape, u. a.

Durch eine Einigung der Beteiligten im Januar 1996 wurden beide Konzepte zu einem Protokoll zusammengefaßt: Secure Encrypted Transaction (SET). Hauptgrund für die Einigung war die angestrebte Interoperabilität, die eine wesentliche Voraussetzung für den breiten Einsatz eines solchen Verfahrens.

Anforderungen an dieses System:

- Vertraulichkeit von Zahlungs- und Bestellungsinformation (durch Verschlüsslung)
- Integrität der übertragenen Daten (Digitale Unterschriften)
- Authentifizierung daß der Kreditkartenbenutzer auch der Berechtigte Nutzer ist
- Authentifizierung daß der Händler Zahlungen mit kreditkarte annehmen kann
- Ein Protokoll das nicht von von TCP/IP Sicherheit abhängt oder damit interferiert.
- Interoperabilität, Portierbarkeit

Im Gegensatz zu IPSec oder SSL gibt es bei SET keine Auswahlmöglichkeit bei den Algorithmen – das macht auch Sinn, weil es sich ja um eine bestimmte Anwendung handelt. Bei SET kommt das RSA-Verfahren beim Signieren und für Key Exchange zum Einsatz, DES zum Verschlüsseln und SHA-1 als Hashfunktion. Ein wichtiges Feature von SET ist, daß der Händler die Kreditkartennummer des Kunden nicht erfährt sondern nur die Bank.

SET Teilnehmer:

- Karteninhaber
- Händler
- Kartenaussteller
- Kapitalnehmer (nimmt dem Händer das Abwickeln der Kreditkartenbezahlungen ab)
- Payment Gateway (Interface zwischen SET und den existierenden Kartennetzwerken
- Certification Authority (Zertifikationsstelle für Karteninhaber, Händler und Payment Gateways, hierarchisch !)

Abfolge der Events bei einer SET-Transaktion:

1. Kunde eröffnet ein kreditkartenfähiges Konto bei einer SET unterstützenden Bank
2. Kunde erhält ein digitales Zertifikat (bestätigt den public key und das Ablaufdatum)
3. Händler haben auch Zertifikate (X.509v3 digitale Zertifikate) - für jede Karte die er akzeptiert für zwei Schlüssel: zum Signieren und zum Key Exchange.
4. Der Kunde tätigt eine Bestellung
5. Zusätzlich zum Bestellformular schickt der Händler eine Kopie seiner Zertifikats, daß er ein echter Händler ist.
6. Der Kunde schickt die Bestellung und Zahlungsinformation an den Händler zusammen mit seinem Zertifikat.
7. Der Händler verlangt Zahlungsautorisierung (das Payment Gateway muß bestätigen, daß die Kreditkarte o.k. ist)
8. Der Händler bestätigt die Bestellung
9. Die Ware oder Dienstleistung wird vom Händler bereitgestellt
10. Der Händler verlangt Bezahlung (vom payment Gateway).

Ein wichtiges Detail von SET ist die sogenannte "duale Signatur" – Zwei Nachrichten werden verbunden die für verschiedene Empfänger bestimmt sind. In diesem Fall geht es um die Bestellungsinformation die an den Händler geht und die Bezahlungsinformation die an die Bank geht – sie müssen verbunden sein um bei Streifällen die Aktion rekonstruieren zu können (diese Bezahlung ist für diese Ware und nicht für eine andere), die Bank und der Händler sollen aber andererseits die jeweils anderen Informationen nicht lesen können.

Das wird wie folgt realisiert: der Kunde berechnet einen Hash der OI und PI, fügt diese zusammen und berechnet vom Ergebnis wieder einen Hash. Das Ergebnis wird mit dem privaten Signatur-Key signiert. Hat der Händler nun diese duale Signatur DS, die OI, und den Hash der PI kann er $H(PIMD||H(OI))$ und $D_{KU_C}(DS)$ berechnen. Sind diese Ausdrücke gleich, hat der Händler die Unterschrift verifiziert. Die Bank kann das anaolg machen mit vertauschten Rollen von PI und OI.

4.2.4 Sichere E-mail

Es gibt diverse Protokolle, die für die Verschlüsselung und Authentifizierung verwendet werden. Die bekanntesten davon sind:

- PGP,
- OpenPGP,
- S/MIME,
- PEM,
- MOSS.

4.2.4.1 Schlüsselmanagement

Schwieriger als die Bereitstellung öffentlicher Schlüssel ist die Frage der Authentizität. Soll die Echtheit eines öffentlichen Schlüssels (Public Keys) einwandfrei nachgewiesen werden, werden digitale Zertifikate benötigt, welche eine eindeutige Verbindung zwischen kryptographischen Schlüssel und Besitzer herstellen. Technisch gesehen handelt es sich bei Zertifikaten um eine besondere Form der Digitalen Signatur, so daß die Überprüfung von jedem Benutzer nachvollzogen werden kann. Es gibt zwei grundsätzliche Lösungen, durch wen ein Zertifikat vergeben wird:

- Erzeugung durch Benutzer: Aufbau eines ”Web of Trust”
- Erzeugung durch Instanzen: Aufbau einer Zertifizierungshierarchie

Web of Trust Grundsätzlich wird bei PGP ein Zertifikat durch einen anderen Benutzer erzeugt. Damit bestätigt dieser, daß er die Person als diejenige anerkennt, die er für den Besitzer des Schlüssels hält. Im Zuge von PGP Key-Signing Parties treffen sich auf Konferenzen oft PGP Benutzer und tauschen ihre Schlüssel aus, wobei sie sich durch Personalausweis oder Führerschein identifizieren. Die Zertifikate sind sehr unterschiedlich in ihrer Qualität, die von einem anderen Benutzer daher nicht oder nur schwer beurteilt werden kann. Nur wenn ein Benutzer einen anderen Benutzer kennt und weiß, nach welchen Grundsätzen er andere zertifiziert, kann er sich auf die Qualität verlassen. Das Web-of-trust wird dann aufgebaut indem die unterschiedlichen Qualitäten der Zertifikate gemittelt und beim Zustandekommen neuer Informationen neu bewertet werden.

Das ”Web of Trust” ist damit integraler Bestandteil von PGP, das ohne den Gedanken an eine wie immer auch geartete Hierarchie entwickelt wurde. Die Schlüsselverteilung erfolgt dabei über Netzwerk-Dienste wie Finger, oder aber durch Server, die mittels Electronic Mail oder WWW abgefragt werden können.

Zertifizierungsinstanzen (Certification Authorities / CAs) Zertifizierungsinstanzen übernehmen im Rahmen einer Hierarchie die Rolle eines vertrauenswürdigen Dritten. Sofern beide Kommunikationspartner der Instanz vertrauen und jeweils über einen durch diese zertifizierten öffentlichen Schlüssel verfügen, ist eine sofortige Aufnahme einer gesicherten (sowohl unter dem Aspekt der Vertraulichkeit als auch der Authentizität) Kommunikation möglich.

Die Regeln, die für eine Zertifizierungsinstanz bindend sind, werden durch sogenannte Policy Certification Authorities (PCAs) festgelegt. Damit soll verhindert werden, das ein unterschiedlicher Maßstab für die Zertifizierung den Nutzen des gesamten Verfahrens einschränkt. Außerdem wird Transparenz für den Benutzer geschaffen. Er kann sich nun anhand öffentlicher Richtlinien informieren, unter welchen Umständen und mit welchen Verfahren die Zertifikate erteilt werden. Bedingt durch die Entwicklung von CAs und PCAs steht auch die Verwendung des X.509 Formats für solche Zertifikate in einem engen inhaltlichen Zusammenhang, da X.500 ein ebenfalls hierarchisch gegliedertes Directory System aufbaut, in dem öffentliche Schlüssel sehr gut bereitgehalten werden können.

4.2.4.2 PGP

PGP wurde von Phil Zimmerman entwickelt und ist gerade durch die Tatsache daß es lange von nur einer Person entwickelt wurde ein sehr bemerkenswertes System.

Folgende Vorgangsweise wurde eingeschlagen:

- Die besten kryptographischen Algorithmen werden als Elemente verwendet (IDEA, SHA-1, RSA, El-Gamal, ...)
- Diese Algorithmen wurden in eine von Prozessor und OS unabhängige Applikation integriert die sehr einfach zu bedienen ist.
- Das Paket inklusive sourcecode und Doku ist frei am Netz erhältlich
- Eine Firma stellt eine billige komerzielle Version zur Verfügung (damit Leute die Support wollen diesen auch kriegen)

Zusätzlich zum Schlüsselmanagement stellt PGP folgende 5 Elemente zur Verfügung:

1. Authentifizierung: Die Nachricht wird mit SHA-1 gehashed, der Hash mit dem private Key des Senders signiert und vor die Nachricht eingefügt. Der Empfänger nimmt den RSA public Key des Senders um den Hash Code zu rekonstruieren, wendet selbst SHA-1 auf die Nachricht an und vergleicht die Hash Werte.
2. Vertraulichkeit: Der Sender generiert eine zufällige 128 Bit Zahl die als Session Key verwendet wird und verschlüsselt die Nachricht mit diesem Key. Der Session Key wird mit dem Public Key der Empfängers verschlüsselt und der Nachricht vorangestellt. Der Empfänger entschlüsselt mit seinem Private Key und dann den Text mit dem Session Key.
3. Kombination von 1) und 2): Die Nachricht wird zusammen zusammen mit der Signatur verschlüsselt.
4. Kompression: Die Nachricht wird nach dem Signieren aber vor der Verschlüsselung komprimiert (für die spätere Speicherung ist es praktischer wenn man nicht den komprimierten Text sondern das Original aufheben muß; die verwendeten Algorithmen zur Kompression sind nicht deterministisch und erzeugen verschiedene Bitstreams; die komprimierte Nachricht hat weniger Redundanz was eine Cryptoanalysis schwieriger macht)
5. E-mail Kompatibilität: Umwandlung des Bitstreams in ASCII character durch radix-64 Konversion (d.h. auch nur signierte Texte werden nicht in Plaintext übertragen)
6. Segmentierung und Zusammensetzung: Zu große Nachrichten werden aufgeteilt, getrennt verschickt und beim Empfänger wieder zusammengesetzt.

Auf jedem PGP Teilnehmerknoten gibt es ein Paar von sog. “Key-Rings”, einen private Key ring und einen public Key ring. Im private Key ring enthält alle public-Key private-Key Paare des Benutzers. Die private Keys sind natürlich verschlüsselt (User wählt Passwort, System berechnet Hash und verwendet Ergebnis als Schlüssel einer symmetrischen Verschlüsselung). Zusätzlich werden noch Erzeugungsdatum, Key-ID (least significant Bits des Keys), user ID gespeichert. Der public Key ring enthält die verschiedenen public Keys mit Erzeugungsdatum und den Eigentümern. Auch die Trust Werte sind hier abgelegt.

4.2.4.3 S/MIME

Während sich PGP bei der Verwendung im privaten Bereich durchgesetzt hat, wird sich wohl S/MIME als der Industriestandard heraubilden.

MIME ist die Lösung für diejenigen Datentypen die mit SMTP nicht korrekt übertragen werden können. S/MIME erweitert nun diese Funktionalität um Authentifizierung und Sicherheits Features von PGP. S/MIME stellt folgende Funktionen zur Verfügung:

- Enveloped Data: verschlüsselter Content und die entsprechneden Keys für einen oder mehrere Empfänger
- Signed Data: Hash der Daten und Verschlüsselung mit dem private Key. Daten und Signatur werden base64 kodiert. Kann nur von einem Empfänger mit S/MIME Fähigkeit gelesen werden.
- Clear-signed Data: wie vorher, aber hier wird nur die digitale Signatur base64 kodiert. Deshalb können die daten auch von einem Empfänger ohne S/MIME Fähigkeit gelesen werden.
- Signed and enveloped Data: Kombinationen der obigen Optionen.

Bezüglich der Unterstützung von verschiedenen kryptographischen Algorithmen gibt es die MUST/SHOULD Struktur. Um Spezifikationskonform zu sein, müssen die MUST Algorithmen inkludiert sein. SHOULD Algorithmen werden nur empfohlen.

- Hashing: MUST: SHA-1, MD5; SHOULD use SHA-1
- Verschlüsselung für Signatur: MUST: DSS; SHOULD: RSA
- Verschlüsselung für Key Exchange: MUST: El Gamal; SHOULD: RSA
- Symmetrische Verschlüsselung: MUST: tripleDES; SHOULD: RC2/40

Die public Key Zertifizierung folgt wieder X.509v3, ist also zentral/hierarchisch organisiert (z.B. von VeriSign kommerziell ausgestellt).

Zusätzliche Features sind noch:

- Signierte Receipts - es können empfangsbestätigungen geordert werden.
- Security Beschreibung - für welche Usergruppe sind die Daten bestimmt, ...
- Sicher Mailing Listen

4.2.4.4 PEM

Im Gegensatz zu PGP verwendet PEM CAs für das Schlüsselmanagement. Die Zertifizierung folgt X.509. Das Grundprinzip der Funktionsweise folgt PGP, ist aber im Gegensatz zu PGP ein Internet Standard. Unwichtig.

4.2.4.5 MOSS

MOSS wurde entworfen um die PEM Einschränkungen bzgl. MIME Daten zu entschärfen. Außerdem ist die Zertifizierungshierarchie entschärft, die Benutzerfreundlichkeit wurde von PGP übernommen. Es gibt jedoch so viele Implementierungsoptionen daß Kommunikation zwischen verschiedenen Implementierungen nicht mehr gewährleistet ist. Siehe `ftp://ftp.tis.com/pub/MOSS/FAQ`. Hat sich nicht durchgesetzt.