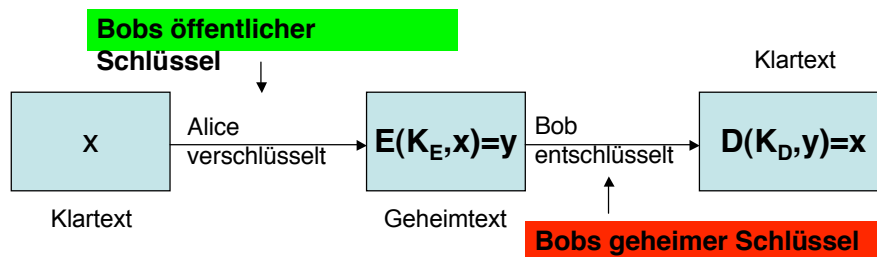


## 3.2 Asymmetrische Verfahren

Idee: Schlüsselpaar  $(K_E, K_D)$  mit einem *öffentlichen* und einem *privaten* Schlüssel



Bei der asymmetrischen Verschlüsselung hat jeder Teilnehmer ein persönliches Schlüsselpaar, das aus einem geheimen und einem öffentlichen Schlüssel besteht. Möchte Alice mit Bob kommunizieren, so benutzt sie Bobs öffentlichen Schlüssel, um ihre Nachricht an ihn zu verschlüsseln. Diesen verschlüsselten Text schickt sie dann an Bob, der mit Hilfe seines geheimen Schlüssels den Text wieder entschlüsseln kann. Da einzig Bob Kenntnis von seinem geheimen Schlüssel hat, ist auch nur er in der Lage, an ihn adressierte Nachrichten zu entschlüsseln. Es muss sichergestellt sein, dass man aus dem öffentlichen Schlüssel nicht auf den geheimen Schlüssel schließen kann. Veranschaulichen kann man sich ein solches Verfahren mit einer Reihe von einbruchssicheren Briefkästen. Wenn ich eine Nachricht verfasst habe, so suche ich den Briefkasten mit dem Namensschild des Empfängers und werfe den Brief dort ein. Danach kann ich die Nachricht selbst nicht mehr lesen oder verändern, da nur der legitime Empfänger im Besitz des Schlüssels für den Briefkasten ist.

## 3.2 Asymmetrische Verfahren

Beispiele: RSA, ElGamal, ...

einfaches Schlüsselmanagement

- Keine Übertragung geheimer Schlüssel im Vorfeld
- Problem: Unverfälschtheit des öffentlichen Schlüssels

Nachteil: langsamer als symmetrische Verfahren

Der Vorteil von asymmetrischen Verfahren ist das einfache Schlüsselmanagement. Betrachten wir wieder ein Netz mit  $n$  Teilnehmern. Um sicherzustellen, dass jeder Teilnehmer jederzeit eine verschlüsselte Verbindung zu jedem anderen Teilnehmer aufbauen kann, muss jeder Teilnehmer ein Schlüsselpaar besitzen. Man braucht also  $2n$  Schlüssel oder  $n$  Schlüsselpaare. Ferner ist im Vorfeld einer Übertragung kein sicherer Kanal notwendig, da alle Informationen, die zur Aufnahme einer vertraulichen Kommunikation notwendig sind, offen übertragen werden können. Hier ist lediglich auf die Unverfälschtheit (Integrität und Authentizität) des öffentlichen Schlüssels zu achten. Nachteil: Im Vergleich zu symmetrischen Verfahren sind reine asymmetrische Verfahren jedoch um ein Vielfaches langsamer.

## **3.2.1 RSA**

RSA-Algorithmus (Ronald Rivest, Adi Shamir, Leonard Adleman), 1978

Basiert auf dem Faktorisierungsproblem großer, natürlicher Zahlen

Dient zum Verschlüsseln, aber auch zum sicheren Austausch von Kommunikationsschlüsseln für symmetrische Verfahren (z.B. DES-Schlüssel)

Das bekannteste asymmetrische Verfahren ist der RSA-Algorithmus, der nach seinen Entwicklern Ronald Rivest, Adi Shamir und Leonard Adleman benannt wurde. Der RSA-Algorithmus wurde 1978 veröffentlicht. Er basiert auf dem Faktorisierungsproblem großer, natürlicher Zahlen. In der Praxis kommt das Verfahren sowohl zum Verschlüsseln als auch zum sicheren Austausch von Kommunikationsschlüsseln (z.B. DES-Schlüssel) sowie zur Erstellung und Überprüfung digitaler Unterschriften zum Einsatz. RSA ist beispielsweise Bestandteil von SSL-fähigen Browsern wie Netscape oder dem Microsoft Explorer.

## 3.2.1 RSA

### Funktionsweise des RSA

#### Schlüsselerzeugung

1. Wähle zufällig zwei große ( $>10^{100}$ ) verschiedene Primzahlen  $p$  und  $q$  und berechne  $n = p * q$ . Den Wert  $n$  bezeichnet man als RSA-Modul.
2. Wähle zufällig  $e$  aus  $\{0, \dots, n - 1\}$ , so dass gilt:  $e$  ist teilerfremd (relativ prim) zu  $J(n) = (p - 1) * (q - 1)$ , d.h.  $\text{ggT}(J(n), e) = 1$

Man kann die Einzelschritte zur Durchführung des RSA-Verfahrens folgendermaßen beschreiben. Schritt 1 bis 3 sind die Schlüsselerzeugung, Schritt 4 und 5 sind die Verschlüsselung, 6 und 7 die Entschlüsselung.

1. Wähle zufällig 2 verschiedene Primzahlen  $p$  und  $q$  und berechne  $n = p * q$ . Der Wert  $n$  wird als RSA-Modul bezeichnet.
2. Wähle zufällig  $e$  aus der Menge  $\{0, \dots, n - 1\}$ , so dass gilt:  $e$  ist teilerfremd zu  $J(n) = (p - 1) * (q - 1)$ . Danach kann man  $p$  und  $q$  "wegwerfen".

### 3.2.1 RSA

3. Wähle  $d$  aus  $\{0, \dots, n - 1\}$  mit  $e * d = 1 \text{ mod } J(n)$

Dann ist

$(n, e)$

der *öffentliche* Schlüssel und

$(n, d)$

der *private* Schlüssel

3. Wähle  $d$  aus der Menge  $\{0, \dots, n - 1\}$  mit  $e * d = 1 \text{ mod } J(n)$ , d.h. das Produkt  $e*d$  geteilt durch  $J(n)$  ergibt einen Rest von 1. Dann ist  $e$  die multiplikative Inverse zu  $d$  modulo  $J(n)$ . Danach kann man  $J(n)$  "wegwerfen". Dann ist  $(n, e)$  der öffentliche Schlüssel  $P$  und  $(n, d)$  ist der geheime Schlüssel  $S$ .

## 3.2.1 RSA

### Verschlüsselung

4. Zum Verschlüsseln wird die als (binäre) Zahl dargestellte Nachricht  $N$  in Teile  $M$  aufgebrochen, so dass jede Teilzahl  $M$  kleiner als  $n$  ist.
5. Verschlüsselung des Klartextes  $M$  aus  $\{0, \dots, n - 1\}$ :  
$$C = E((n, e), M) := M^e \bmod n.$$

4. Zum Verschlüsseln wird die als (binäre) Zahl dargestellte Nachricht in Teile aufgebrochen, so dass jede Teilzahl kleiner als  $n$  ist.

5. Verschlüsselung des Klartextes (bzw. seiner Teilstücke)  $M$  aus der Menge  $\{1, \dots, n - 1\}$ :

$$E((n, e); M) := M^e \bmod n.$$

## 3.2.1 RSA

### Entschlüsselung

6. Zum Entschlüsseln wird das binär als Zahl dargestellte Chiffretext in Teile  $C$  aufgebrochen, so dass jede Teilzahl  $C$  kleiner als  $n$  ist.

7. Entschlüsselung des Chiffretextes  $C$  aus  $\{0, \dots, n - 1\}$ :  
$$D((n, d), C) := C^d \bmod n.$$

6. Zum Entschlüsseln wird das binär als Zahl dargestellte Chiffretext in Teile aufgebrochen, so dass jede Teilzahl kleiner als  $n$  ist.

7. Entschlüsselung des Chiffretextes (bzw. seiner Teilstücke)  $C$  aus  $\{1, \dots, n - 1\}$ :

$$M = D((n, d); C) := C^d \bmod n.$$

Die Sicherheit des RSA-Verfahrens basiert auf der Geheimhaltung der Werte  $p, q$  und  $J(n)$ .

## 3.2.1 RSA

### Beispiel 1:

RSA mit kleinen Primzahlen und mit einer Zahl ( $M=2$ ) als Nachricht

1. Die gewählten Primzahlen seien  $p = 5$  und  $q = 11$ .  
Also ist das RSA-Modul  $n = 55$ .
2. Es ist  $J(n) = (p - 1) (q - 1) = 40$ . Wähle dann  $e = 7$ ,  
denn  $\text{ggT}(40,7)=1$ .
3. Wähle nun  $d = 23$ , denn  $23 * 7 = 161 = 1 \pmod{40}$   
Öffentlicher Schlüssel: (55, 7)  
Privater Schlüssel: (55, 23)

### 3.2.1 RSA

4. Nachricht sei nur die Zahl  $M = 2$  (also ist kein Aufbrechen in Blöcke nötig, da  $M < n = 55$ ).
5. Verschlüsseln:  
 $C = 2^7 \bmod 55 = 128 \bmod 55 = 18$
6. Chiffre ist nur die Zahl  $C = 18$  (also kein Aufbrechen in Blöcke nötig, da  $18 < n = 55$ ).
7. Entschlüsseln:  
 $M = 18^{23} \bmod 55 = 2$

## 3.2.1 RSA

### Beispiel 2:

RSA mit etwas größeren Primzahlen und einem Text aus Großbuchstaben "ATTACK AT DAWN"

1. Die gewählten Primzahlen seien  $p = 47$ ,  $q = 79$ .  
Dann ist das RSA-Modul  $n = 3713$ .
2. Es ist  $J(n) = (p - 1) (q - 1) = 3588$ . Wähle dann  $e = 37$  (denn  $\text{ggT}(3588, 37) = 1$ ).
3. Wähle  $d = 97$ , denn  $e * d = 37 * 97 = 3.589 = 1 \text{ mod } 3588 = 1 \text{ mod } J(n)$ .

Öffentlicher Schlüssel: (3713, 37)

Privater Schlüssel: (3713, 97)

## 3.2.1 RSA

### 4. Nachricht in Blöcke aufteilen

Die Zeichen werden auf folgende einfache Weise codiert:

Leerzeichen = 0, A = 1, ..., Z = 26

Text: A T T A C K    A T    D A W N  
Zahl: 01 20 20 01 03 11 00 01 20 00 04 01 23 14

Aufteilung dieser 28-stelligen Zahl in 4-stellige Teile  
(denn  $26^2$  ist noch kleiner als  $n = 3713$ ), d.h. dass die  
Blocklänge 2 beträgt.

0120 2001 0311 0001 2000 0401 2314

## 3.2.1 RSA

### 5. Verschlüsselung

alle 7 Teile werden jeweils per  $C = M^{37} \pmod{3713}$  verschlüsselt.

Es ergibt sich folgender Chiffretext

1404 2932 3536 0001 3284 2280 2235

### 3.2.1 RSA

6. Chiffre in Blöcke aufteilen:

Chiffre: 1404 2932 3536 0001 3284 2280 2235

Aufteilung dieser 28-stelligen Zahl in 4-stellige Teile.

7. Entschlüsselung aller 7 Teile jeweils per:

$M = C^{97} \pmod{3.713}$ :

0120 2001 0311 0001 2000 0401 2314

Umwandeln von 2-stelligen Zahlen in Großbuchstaben und Blanks ergibt Klartext.

## 3.2.1 RSA

**Frage:** Ist die Entschlüsselungsfunktion invers zur Verschlüsselungsfunktion?

**Satz von Euler:** Ist  $n=pq$ , so gilt für alle Zahlen  $m$  mit  $m < n$  und alle natürlichen Zahlen  $k$ :

$$m^{k \cdot J(n)+1} \bmod n = m$$

Zur Entschlüsselung ergibt sich damit

$$\begin{aligned} C^d \bmod n &= M^{e \cdot d} \bmod n = M^{1 \bmod J(n)} \bmod n \\ &= M^{k \cdot J(n)+1} \bmod n = M \end{aligned}$$

Es ist noch mathematisch zu zeigen, dass die Entschlüsselungsfunktion tatsächlich die "Inverse" der Verschlüsselungsfunktion ist, d.h. wenn eine Klartextnachricht  $M$  nach RSA-Algorithmus mit öffentlichem Schlüssel erst verschlüsselt und dann mit geheimen Schlüssel entschlüsselt wird, bekommt man die Nachricht  $M$  zurück. Das dies funktioniert, basiert auf dem Satz von Euler.

## 3.2.1 RSA

### Theoretische Sicherheit des RSA-Verfahrens

Beruhet darauf, dass die Faktorisierung großer Zahlen schwer ist

- Effizienter Algorithmus zur Faktorisierung  $n = pq$  großer Zahlen würden den RSA-Algorithmus beeinträchtigen
- Fermat'sche Faktorisierung: schnelle Lösung wenn  $p$  und  $q$  nah an  $\sqrt{n}$   
Beruht auf dem Satz

$$n = a^2 - b^2 = (a + b)(a - b) = pq$$

Die Sicherheit des RSA-Algorithmus basiert auf der empirischen Beobachtung, dass die Faktorisierung großer ganzer Zahlen ein schwieriges Problem ist. Besteht wie beim RSA-Algorithmus das zugrunde liegende RSA-Modul  $n$  aus dem Produkt zweier großer Primzahlen  $p$ ,  $q$  (typische Längen:  $p$ ,  $q$  500 - 600 bit,  $n$  1024 bit), so lässt sich  $n = pq$  aus  $p$  und  $q$  leicht bestimmen, jedoch ist es mit den bisher bekannten Faktorisierungsalgorithmen nicht möglich,  $p$  und  $q$  aus  $n$  zu gewinnen. Nur mit Kenntnis von  $p$  und  $q$  lässt sich jedoch der private aus dem öffentlichen Schlüssel ermitteln. Die Entdeckung eines Algorithmus zur effizienten Faktorisierung von Produkten  $n = pq$  großer Primzahlen würde daher den RSA-Algorithmus wesentlich beeinträchtigen.

Unterscheiden sich die beiden Primzahlen  $p$  und  $q$  nur gering von der Wurzel von  $n$ , kann man die Fermat'sche Faktorisierung anwenden. Dieser Algorithmus beruht auf dem Satz, dass es für eine natürliche Zahl  $n$ , die Produkt zweier Zahlen  $p$  und  $q$  ist, zwei natürliche Zahlen  $a$  und  $b$  gibt, so dass  $n = a^2 - b^2 = (a + b)(a - b) = pq$ . Dann ist also  $p = (a+b)$  und  $q = (a - b)$ . Die Idee des Algorithmus ist, nach Zahlen  $a$  und  $b$  zu suchen, die diese Gleichung erfüllen. Man beginnt die Suche bei einem Wert von Wurzel  $n$  für  $a$ . Dann wird  $a$  immer um eins erhöht, bis  $a^2 - n$  eine Quadratzahl ist.

## 3.2.1 RSA

### Praktische Sicherheit des RSA-Verfahrens

#### *RSA-Modul-Größe*

je größer, desto schwieriger ist der Schlüssel zu brechen, aber desto länger dauert auch die Verschlüsselung.

Modulgröße sollte über 512 Bit liegen, möglichst 1024 Bit, besser 2048 Bit

#### *Größe der Primzahlen*

mindestens 100-stellige Dezimalzahlen

Länge von  $p$  und  $q$  sollte sich um einige Ziffern unterscheiden

Die Sicherheit des RSA-Verfahrens ist durch die Schwierigkeit des Faktorisierungsproblems sichergestellt. Für die praktische Sicherheit ist es aber auch wichtig, auf die Modul-Größe und die gewählten Primzahlen zu achten.

Die Größe des RSA-Moduls hat Auswirkungen auf die Sicherheit des RSA-Verfahrens. Je größer das Modul, desto schwieriger wird es, den Schlüssel zu brechen. Jedoch führt eine höhere Modul-Größe auch zu einer höheren Verschlüsselungs- und Entschlüsselungszeit. Eine Verdopplung führt im Durchschnitt dazu, dass die Zeit für die Verschlüsselung um einen Faktor vier und die Entschlüsselung um einen Faktor acht wächst. Ein 512-Bit langes Modul gewährleistet beim heutigen Stand der Technik keine ausreichende Sicherheit. Verschlüsselungen mit dieser Modul-Größe konnten durch einzelne Forschergruppen bereits geknackt werden. Deshalb sollte man eine Modul-Größe von 1024 Bit, in sicherheitskritischen Anwendungen von 2048 Bit benutzen.

Die Primzahlen  $p$  und  $q$  sollten mindestens eine Größe im 100-stelligen Dezimalbereich haben, so dass das Modul eine mindestens 200-stellige Dezimalzahl ist. Die Länge der Primzahlen sollte sich um einige Ziffern unterscheiden, da gleich lange Primzahlen Faktorisierungsangriffe erleichtern.

## 3.2.1 RSA

RSA ist bei der Ver- und Entschlüsselung wesentlich langsamer als symmetrische Verfahren.

### Vergleich DES

Hardware: DES etwa 1000 mal schneller

Software: DES etwa 100 mal schneller

RSA nicht zur Verschlüsselung großer Datenmengen benutzen

RSA kann aber gut zum Austausch von Schlüsseln in symmetrischen Verfahren (wie DES) verwendet werden.

Die Ver- und Entschlüsselung beim RSA ist wesentlich langsamer als bei symmetrischen Verfahren. Eine in Hardware realisierte DES-Verschlüsselung ist ca. 1000 mal schneller als eine in Hardware realisierte RSA-Verschlüsselung. Auch eine in Software realisierte DES-Verschlüsselung ist noch etwa 100 mal schneller als eine RSA-Softwarelösung. Aus diesem Grund wird die RSA-Verschlüsselung nicht zur Verschlüsselung großer Datenmengen eingesetzt. Dagegen verwendet man den RSA häufig zur Verschlüsselung und zum Austausch eines symmetrischen Schlüssels.

## 3.2.2 Elliptische Kurven

- Kubische Gleichungen
- Basieren auf dem diskreten Logarithmus-Problem
- Vorteile gegenüber RSA:
  - Geringere Rechenleistung bei der Ver- und Entschlüsselung
  - Kürzere Schlüssellänge (160-Bit Schlüssel entspricht etwa einem 1024-Bit RSA-Schlüssel, ein 320-Bit Schlüssel schon einem 5120 RSA-Schlüssel)
  - Geringerer Speicherbedarf für Schlüssel (günstig z.B. für Chipkarten)

Elliptische Kurven, kurz EC (Elliptic Curves), sind keine Ellipsen, sondern werden so genannt, weil sie mit kubischen Gleichungen dargestellt werden. Sie basieren auf dem diskreten Logarithmus Problem. Bereits im Jahre 1985 wurden diese von Victor Miller und Neal Koblitz unabhängig voneinander zur Konstruktion von asymmetrischen Kryptoalgorithmen vorgeschlagen. Die Vorteile von ECCs, die zukünftig für Chipkartenrealisierungen bestimmt von großem Interesse sein werden, liegen einerseits in der weit geringeren notwendigen Rechenleistung (es wird kein eigener Krypto-Coprozessor benötigt) und andererseits in den kürzeren Schlüssellängen. Ein 160Bit Schlüssel eines ECCs entspricht in etwa dem eines 1024Bit RSA-Schlüssel, ein 320Bit Schlüssel bereits einem vergleichbaren 5120Bit RSA-Schlüssel. Daraus resultiert auch ein geringerer EEPROM Speicherbedarf für die Schlüssel. Ein Nachteil ist, dass ECCs der Wissenschaft wohl bekannt, in der Kryptographie und bei Chipkarten aber eher unerprobt und mathematisch komplex sind.

## 3.3 Hybridverfahren

Kombination symmetrischer und asymmetrischer Verfahren (z.B. Secure Socket Layer, SSL)

- Absender generiert Sitzungsschlüssel
- Sitzungsschlüssel wird mit Hilfe des asymmetrischen Verfahrens verschlüsselt und an Empfänger übertragen
- Empfänger bestimmt Sitzungsschlüssel mit Hilfe seines geheimen Schlüssels
- Nachfolgend werden Daten symmetrisch verschlüsselt

Um die Vorteile von symmetrischen und asymmetrischen Techniken gemeinsam nutzen zu können, werden (zur Verschlüsselung) in der Praxis meist Hybridverfahren verwendet. Hier werden die Daten mittels symmetrischer Verfahren verschlüsselt: der Schlüssel ist ein vom Absender zufällig generierter Sitzungsschlüssel (session key), der nur für diese Nachricht verwendet wird. Anschließend wird dieser Sitzungsschlüssel mit Hilfe des asymmetrischen Verfahrens verschlüsselt und zusammen mit der Nachricht an den Empfänger übertragen. Der Empfänger kann den Sitzungsschlüssel mit Hilfe seines geheimen Schlüssels bestimmen und mit diesem dann die Nachricht entschlüsseln. Auf diese Weise nutzt man das bequeme Schlüsselmanagement asymmetrischer Verfahren und kann trotzdem große Datenmengen schnell und effektiv mit symmetrischen Verfahren verschlüsseln.

### 3.4 Geheimhaltung, Integrität und Authentizität

Zur Erinnerung:

**Geheimhaltung:** nur der Empfänger kann die Nachricht lesen

**Integrität:** Nachricht erreicht den Empfänger so, wie sie abgeschickt wurde

**Authentizität:** es ist sichergestellt, dass eine Nachricht, die angeblich von A stammt, tatsächlich von A ist.

Wir haben Verschlüsselungsverfahren (asymmetrische und symmetrische) vorgestellt. Welche Schutzziele werden nun durch den Sicherheitsmechanismus der Verschlüsselung erreicht? Die bisher vorgestellten Verschlüsselungsverfahren sichern die Vertraulichkeit bzw. die Geheimhaltung von Daten, d.h. kein unautorisierter Benutzer kann die Daten einsehen und nur der Empfänger einer Nachricht kann diese lesen. Neben der Vertraulichkeit der Daten sind jedoch auch die Integrität der Daten und die Authentizität des Datenursprungs von Interesse. Die Integrität fordert, dass eine Nachricht den Empfänger so erreicht, wie sie abgeschickt wurde. Authentizität soll sicherstellen, dass eine Nachricht, die angeblich von A stammt, auch wirklich von A ist.

### 3.4 Geheimhaltung, Integrität und Authentizität

**Symmetrische Verschlüsselung** gewährleistet

**Geheimhaltung** – weil eine verschlüsselte Nachricht nicht ohne Kenntnis des geheimen Schlüssels *dechiffriert* werden kann

**Integrität** – weil eine verschlüsselte Nachricht nicht ohne Kenntnis des geheimen Schlüssels *so modifiziert* werden kann, dass die Dechiffrierung normalen Klartext ergibt

**Authentizität** – weil eine verschlüsselte Nachricht, deren Dechiffrierung normalen Klartext liefert, nur von jemandem, der den geheimen Schlüssel kennt, stammen kann

Betrachten wir symmetrische Verschlüsselungsverfahren, so wird die Geheimhaltung der Daten durch die Verschlüsselung des Klartextes in einen Geheimtext mit dem gemeinsamen geheimen Schlüssel erreicht. Nur mit dem geheimen Schlüssel ist es möglich, eine verschlüsselte Nachricht zu dechiffrieren und zu lesen. Der geheime Schlüssel sorgt auch dafür, dass eine verschlüsselte Nachricht nicht so modifiziert werden kann, dass die Entschlüsselung einen normalen und sinnvollen Klartext ergibt. Wenn die verschlüsselte Nachricht bei der Entschlüsselung einen sinnvollen und normalen Klartext ergibt, so muss die Nachricht von jemanden verschlüsselt worden sein, der den geheimen Schlüssel kennt. Wenn ich mir sicher bin, dass nur mein Kommunikationspartner über den geheimen Schlüssel Bescheid weiß, kann ich auch sicher sein, dass die Nachricht von ihm stammt. Es wird somit die Authentizität des Ursprungs der Nachricht sichergestellt.

### 3.4 Geheimhaltung, Integrität und Authentizität

*Achtung:* Integrität und Authentizität sind eventuell bedroht durch **Wiedereinspielen** (replay) zuvor abgehörter (verschlüsselter) Nachrichten oder Nachrichtenteile.

*Abhilfe:* Nachrichten durchnummerieren  
Rückkoppelung ( gemäß Cipher Block Chaining, Output Feedback, Cipher Feedback)  
Nachricht mit Zeitangabe versehen

Durch das Wiedereinspielen (replay) zuvor abgehörter verschlüsselter Nachrichten kann die Integrität und die Authentizität eventuell bedroht sein. Angreifer können abgefangene Nachrichten ohne Änderung zu einem späteren Zeitpunkt wieder an denselben Adressaten senden. Dann kommt diese Nachricht nicht mehr vom eigentlichen Sender, sondern vom Angreifer (d.h. Schutzziel der Authentizität ist verletzt). Außerdem kann durch das Wiedereinspielen von nur Teilen der Nachricht die Integrität verletzt werden, da die Nachricht in diesem Fall modifiziert wurde. Abhilfe gegen das Wiedereinspielen von abgefangenen Nachrichten kann die Nummerierung von Nachrichten sein (die abgefangene und wieder versendete Nachricht hätte dann dieselbe Nummer, wie die Originalnachricht), das Einbauen einer Rückkopplung in die Nachrichten (die aktuelle Nachricht hängt von der vorigen Nachricht ab) oder auch durch einen Zeitstempel.

### 3.4 Geheimhaltung, Integrität und Authentizität

*Verbleibende Schwäche:*

Geheimer Schlüssel ist **Gruppenschlüssel** von 2 (oder mehr) Partnern

**Verbindlichkeit** ist daher *nicht* gewährleistet:

A: „Ich habe von B diese Nachricht erhalten“  
ist weder beweisbar noch widerlegbar.

Das Schutzziel der Verbindlichkeit, d.h. die Sicherstellung, dass Aktionen (z.B. Senden und Empfangen von Nachrichten) später von den Beteiligten nicht abgestritten werden können, ist mit einfacher symmetrischer Verschlüsselung nicht möglich, da alle Partner den geheimen Schlüssel besitzen. So kann jeder Partner eine Nachricht erstellen und verschlüsseln, und behaupten, diese von einem anderen Partner bekommen zu haben. Diese Aussage ist weder beweisbar noch widerlegbar.

### 3.4 Geheimhaltung, Integrität und Authentizität

**Asymmetrische Verschlüsselung** ermöglicht Entscheidung für Geheimhaltung *oder* Authentizität *oder* beides.

Chiffrierung mit öffentlichem Schlüssel garantiert Geheimhaltung und Integrität, *aber nicht Authentizität*:

Wenn Anna „Herzliche Grüße von Deinem Bert“ dechiffriert, ist nicht gesichert, dass die Nachricht tatsächlich von Bert kommt.

Die asymmetrische Verschlüsselung ermöglicht durch die Verschlüsselung einer Nachricht mit dem öffentlichen Schlüssel die Vertraulichkeit bzw. Geheimhaltung der Nachricht, da der verschlüsselte Text nur mit dem privaten Schlüssel gelesen werden kann. Die Verschlüsselung mit dem öffentlichen Schlüssel garantiert jedoch nicht die Authentizität des Ursprungs der Nachricht, da der öffentliche Schlüssel von allen benutzt werden kann und allein an der Verschlüsselung nicht gesehen werden kann, von wem die Nachricht stammt.

### 3.4 Geheimhaltung, Integrität und Authentizität

Beim RSA garantiert die Chiffrierung mit privatem Schlüssel (!)  
*Authentizität* und Integrität, *aber nicht Geheimhaltung*:

Jeder kann die Nachricht mit dem öffentlichen Schlüssel dechiffrieren. Ergibt sich normaler Text, ist die Nachricht authentisch.

*Bemerkung:* „authentisch“ bedeutet hier „von demjenigen, der den geheimen Schlüssel kennt“ (sofern nur einer ihn kennt).

„Herzliche Grüße von Deinem Bert“ stammt nur dann wirklich von Bert, wenn der für die Dechiffrierung verwendete *öffentliche Schlüssel wirklich von Bert ist !*

Für das RSA-Verfahren gilt wegen der Vertauschbarkeit von öffentlichem und privatem Schlüssel, dass die Authentizität mit dem privaten Schlüssel garantiert werden kann. Kann ich eine mit einem privaten Schlüssel chiffrierte Nachricht mit dem dazugehörigen öffentlichen Schlüssel dechiffrieren, d.h. ergibt sich normaler Text, so ist die Nachricht authentisch, d.h. die Nachricht kommt vom Besitzer des öffentlichen Schlüssels. Hier ist dann allerdings noch zu klären, dass der öffentliche Schlüssel wirklich der Person gehört, die der öffentliche Schlüssel vorgibt (darauf kommen wir später zurück). Das heißt eine Nachricht M stammt nur dann wirklich von Sender A, wenn der zur Dechiffrierung verwendete öffentliche Schlüssel zu A gehört.

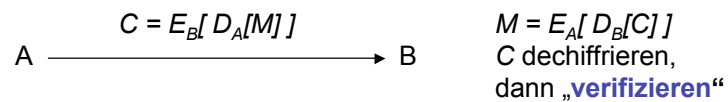
### 3.4 Geheimhaltung, Integrität und Authentizität

RSA:

$E_A[M]$  = mit dem öffentlichen Schlüssel von A chiffriertes M

$D_A[M]$  = mit dem geheimen Schlüssel von A „signiertes“ M

Geheime **und** authentische Nachricht von A an B – zwei Möglichkeiten:



entspricht „unterschiedenes Dokument in verschlossenem Kuvert“

Durch die kombinierte Anwendung von öffentlichem und privatem Schlüssel lässt sich beim RSA sowohl die Vertraulichkeit als auch die Authentizität der Nachricht sicherstellen. Das Verschlüsseln mit dem öffentlichen Schlüssel sichert die Vertraulichkeit und Integrität, die Anwendung des privaten Schlüssels sichert die Authentizität. Um geheime und authentische Nachrichten zu erzeugen gibt es zwei Möglichkeiten: Man kann eine Nachricht M zunächst mit dem privaten Schlüssel  $D_A$  vom Sender A signieren und das Ergebnis mit dem öffentlichen Schlüssel  $E_B$  des Empfängers B verschlüsseln. Dies entspricht einem unterschriebenen Dokument in einem verschlossenem Kuvert.

### 3.4 Geheimhaltung, Integrität und Authentizität

$$A \xrightarrow{C = D_A[E_B[M]]} B \quad M = D_B[E_A[C]]$$

entspricht „Dokument in unterschriebenem, verschlossenem Kuvert“

Andererseits kann man die Nachricht jedoch auch erst mit dem öffentlichem Schlüssel verschlüsseln und auf dieses Ergebnis den privaten Schlüssel anwenden. Dies entspricht dann einem Dokument in einem unterschriebenen und verschlossenem Kuvert.

## 3.5 Digitale Unterschriften

### **Digitale Unterschriften** (*digital signature*)

sollen juristisch gleichwertig sein zu konventionellen Unterschriften

#### **Identifikation**

Auskunft über die Person des Unterzeichners

#### **Echtheit**

Die Unterschrift bezeugt die Anerkennung des Dokuments

#### **Abschluss**

Die Unterschrift erklärt den Inhalt für richtig und vollständig

#### **Warnung**

Der Unterzeichnende wird auf die juristische Bedeutung aufmerksam

Mit digitalen Unterschriften (digital signature) soll die Beweiskraft gegeben werden, um die Urheberschaft einer Nachricht bzw. eines Dokumentes juristisch zweifelsfrei zu bestimmen. Digitale Unterschriften sollen juristisch gleichwertig zu konventionellen Unterschriften sein, d.h. sie müssen wie die herkömmliche Unterschrift folgende Funktionen erfüllen: Identifikation der Person, Echtheit, d.h. die Unterschrift bezeugt, dass das Dokument dem Aussteller vorgelegen hat und von ihm anerkannt wurde, Abschluss, d.h. die Unterschrift erklärt den Text für inhaltlich richtig und vollständig und Warnung, d.h. durch die Notwendigkeit, dass eine Unterschrift geleistet werden muss, wird dem Verfasser die rechtliche Bedeutung des Dokumentes aufgezeigt.

## 3.5 Digitale Unterschriften

Anforderungen an die digitale Unterschrift

**Zweifelsfreie Identität**

- Die Unterschrift muss die Person eindeutig identifizieren

**Keine Wiederverwendbarkeit**

- Die Unterschrift darf nicht von dem Dokument gelöst und anderweitig verwendet werden können

**Unveränderbarkeit**

- Wenn die Unterschrift geleistet wurde, darf das Dokument nicht mehr verändert werden können

**Verbindlichkeit**

- Es darf nicht abstreitbar sein, dass die Unterschrift geleistet

Rechtliche Rahmenbedingungen sind im Deutschen Signaturgesetz (SigG 2001) festgelegt

*<http://www.bsi.de/esig/index.htm>*

Aus diesen Funktionen lassen sich die Anforderungen an digitale Unterschriften ableiten. Die digitale Unterschrift muss die Identität des Unterzeichners zweifelsfrei bestätigen. Sie darf nicht wiederverwendbar und nur mit dem Originaldokument gültig sein. Ein signiertes Dokument darf nicht mehr veränderbar sein. Der Unterzeichner eines Dokuments darf das Unterzeichnen des Dokuments nicht im Nachhinein erfolgreich abstreiten können.

Die rechtlichen Rahmenbedingungen für den Umgang mit digitalen Signaturen in Deutschland sind im Signaturgesetz (SigG) festgelegt.

## **3.5 Digitale Unterschriften**

Signaturerstellung durch spezielle Signierverfahren oder (geeignete) asymmetrische Verschlüsselungsverfahren:

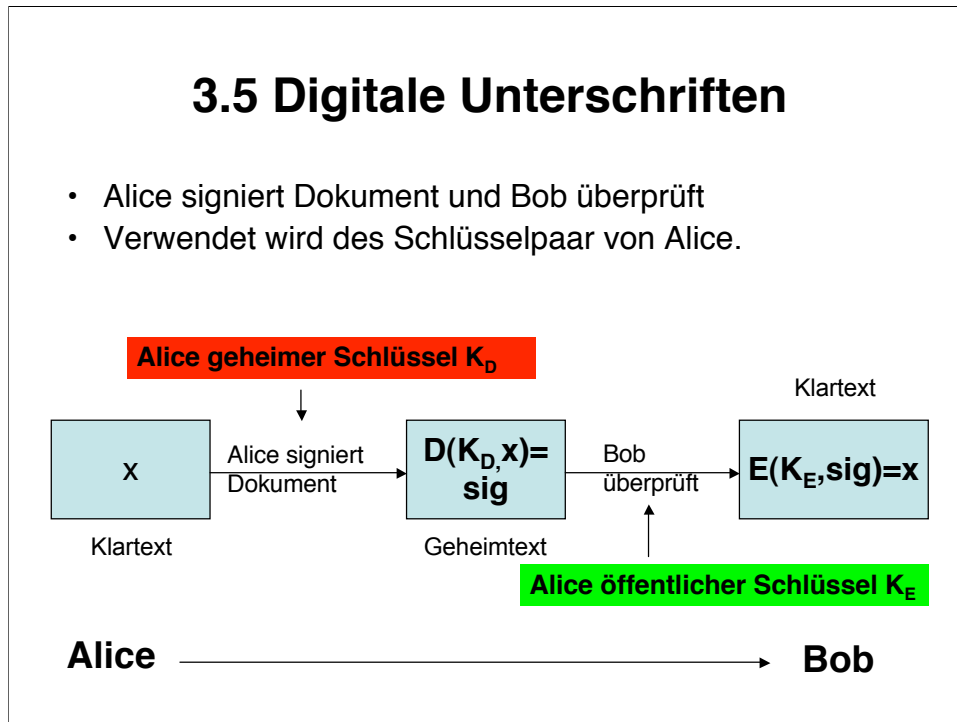
Das RSA-Verfahren kann sowohl zum Verschlüsseln und auch zum Signieren verwendet werden

DSA (Digital Signature Algorithm) ist spezielles Signierverfahren

In der Praxis werden spezielle Signaturverfahren oder asymmetrische Verschlüsselungsverfahren zur Signaturerstellung und –verifikation verwendet. Das RSA-Verfahren kann beispielsweise sowohl zum Verschlüsseln als auch zum Signieren eingesetzt werden. Der DAS (Digital Signature Algorithm) ist ein Beispiel eines speziellen Signieralgorithmuses.

## 3.5 Digitale Unterschriften

- Alice signiert Dokument und Bob überprüft
- Verwendet wird des Schlüsselpaar von Alice.

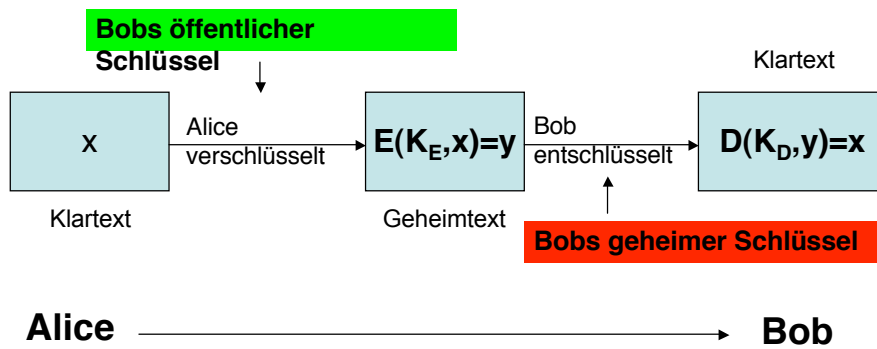


Das Protokoll zur Signierung mit asymmetrischen Verfahren ist wie folgt:

1.  $(K_E, K_D)$  ist das Schlüsselpaar von Alice, bestehend aus einem privaten Signierschlüssel  $K_D$  und dem öffentlichen Verifikationsschlüssel  $K_E$ .
2. Alice macht den Verifikationsschlüssel  $K_E$  öffentlich verfügbar.
3. Sie signiert ein Dokument  $M$  durch Verschlüsseln mit ihrem privaten Schlüssel  $K_D$ ,  $D(M, K_D) = \text{sig}$ , und sendet sig an Bob.
4. Bob holt sich den öffentlichen Signierschlüssel  $K_E$ .
5. Bob verifiziert die Signatur sig durch  $M = E(\text{sig}, K_E)$ . Da nur der Sender Alice selbst ihren geheimen Schlüssel kennt, kann eine Nachricht, die mit dem öffentlichen Schlüssel von Alice entschlüsselt werden kann, nur vom Sender Alice stammen.

### 3.5 Digitale Unterschriften

- Alice sendet verschlüsseltes Dokument an Bob
- Verwendet wird des Schlüsselpaar von Bob.



Vergleich zur Verschlüsselung: Bei der asymmetrischen Verschlüsselung wird das Dokument mit dem geheimen Schlüssel des Empfängers (hier Bob) verschlüsselt. Bei der Signierung wird das Signierschlüsselpaar vom Sender (hier Alice) verwendet und das Dokument mit dem privaten Schlüssel des Senders (hier Alice) signiert. Die Signierung ist also umgekehrt zur asymmetrischen Verschlüsselung.

## 3.5 Digitale Unterschriften

Die asymmetrischer Verschlüsselung (RSA) mit dem öffentlichen Schlüssel des Empfängers  
garantiert die *Vertraulichkeit der Nachricht*,  
aber weder ihre Echtheit noch die des Senders.

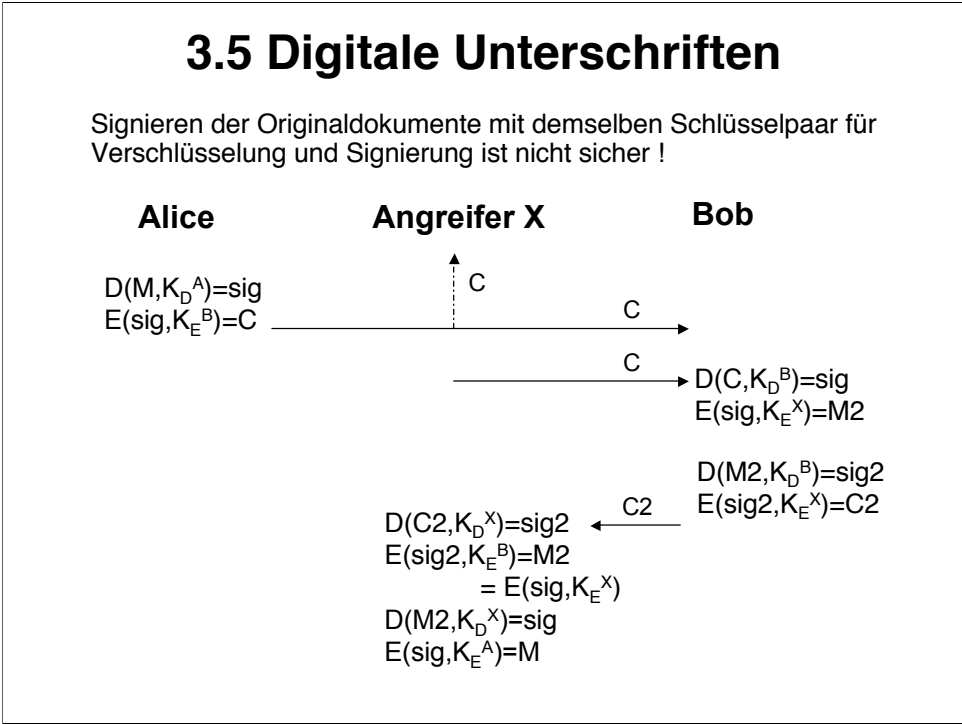
Die digitale Signatur sichert  
die *Echtheit des Senders* (kein anderer besitzt den geheimen Schlüssel),  
sichert aber nicht die Vertraulichkeit der Nachricht (jeder kann mit A's öffentlichem Schlüssel die Nachricht lesen)

Daher kann man zusätzlich mit B's öffentlichem Schlüssel codieren, um sowohl Vertraulichkeit, als auch Authentizität zu garantieren.

Die asymmetrische Verschlüsselung dient dazu, die Vertraulichkeit einer Nachricht oder eines Dokumentes zu gewährleisten. Verschlüsselung garantiert jedoch weder die Echtheit des Senders noch der Nachricht/des Dokuments. Im Gegensatz dazu, sichert die digitale Signatur die Echtheit des Senders, da kein anderer den geheimen Schlüssel zum Signieren besitzt. Die digitale Signatur sichert jedoch nicht die Vertraulichkeit der Nachricht, da jeder den öffentlichen Schlüssel des Signierenden bekommen kann, um damit die Nachricht zu lesen. Daher kann man eine signierte Nachricht noch zusätzlich mit dem öffentlichen Schlüssel des Empfängers verschlüsseln, um sowohl Vertraulichkeit als auch Authentizität zu erreichen. (Dies hatten wir schon auf einer vorigen Folie gezeigt.)

### 3.5 Digitale Unterschriften

Signieren der Originaldokumente mit demselben Schlüsselpaar für Verschlüsselung und Signierung ist nicht sicher !



Es ist jedoch zu beachten, dass die Schlüsselpaare zum Signieren und zum Verschlüsseln unterschiedlich sein sollten, da sonst folgender Angriff erfolgreich durchgeführt werden kann:

Alice sendet eine signierte und verschlüsselte Nachricht C an Bob. Ein Angreifer X fängt die Nachricht C ab und sendet sie zu einem späteren Zeitpunkt seinerseits an Bob. Bob entschlüsselt das von X vorgelegte C mit seinem privaten Schlüssel und verifiziert sig mit dem Verifikationsschlüssel  $K_E^X$  des Angreifers X. Bob erhält eine unsinnige Nachricht M2. Bob sendet M2 verschlüsselt und signiert an X zurück (z.B. durch automatische Empfangsbestätigung der Mail-Software). X entschlüsselt die gesendete Nachricht C2 mit seinem privaten Schlüssel, verifiziert die Signatur sig2 mit dem öffentlichen Schlüssel von Bob und wendet erneut seinen eigenen privaten Schlüssel an, und verifiziert schließlich sig mit dem öffentlichen Schlüssel von Alice. X erhält dadurch die ursprüngliche Nachricht M.

## 3.5 Digitale Unterschriften

Digitale Unterschrift mit **Hashcode** (*digest, hash code*)

Sender berechnet Hashwert  $H[M]$  einer Nachricht  $M$  nicht  $M$ , sondern nur  $H[M]$  wird mit privatem Schlüssel  $D_A$  signiert:

$D_A[H[M]]$  bildet die *digitale Unterschrift von A unter M*

$H$  ist eine *Hash-Funktion*.

Den vorgestellten Angriff kann man auf einfache Weise verhindern, indem man anstelle der Originaldokumente nur deren Hash-Werte signiert. Kenntnisse des Hash-Wertes liefern dem Angreifer aufgrund der Einweg-Eigenschaft keine Auskunft über das Originaldokument. Bei einer digitalen Unterschrift mit Hashcode berechnet der Sender für eine Nachricht  $M$  zunächst einen Hashwert  $H(M)$  mittels einer Hash-Funktion  $H$ . Um Rechenzeit zu sparen, wird dann nur der Hashwert mit dem privaten Schlüssel signiert und an den Empfänger geschickt.

## 3.5.1 Hash-Funktionen

Bilden eindeutigen digitalen Fingerabdruck von Daten.

Eine Hash-Funktion  $H$  ist eine Funktion die eine (variabel lange) Nachricht  $M$  auf einen Hash-Wert  $H(M)$  konstanter Länge abbildet.

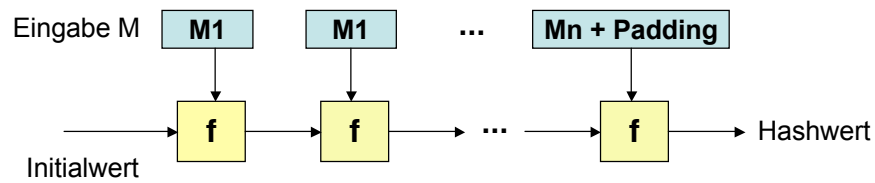
Hash-Funktionen haben die folgenden Eigenschaften

- *Einweg-Funktion*:  $H$  ist nicht invertierbar, d.h. bei gegebenem Hash-Wert  $H(M)$  kann  $M$  nicht ermittelt werden (d.h nicht mit realistischem Rechenaufwand leistbar).
- *Kollisionsresistent*: Es ist ebenfalls (praktisch) nicht möglich, ein paar verschiedener Nachrichten  $M$  und  $M'$  zu finden, für die gilt :  $H(M) = H(M')$ .

Mittels Hash-Funktionen werden eindeutige, sogenannte digitale Fingerabdrücke von Datenobjekten berechnet, mit der die Integrität des Objekts geprüft werden kann. Eine Hash-Funktion ist eine nicht injektive Abbildung, die eine beliebig lange Nachricht  $M$  einem Hash-Wert  $H(M)$  fester Länge zuordnet. Dabei muss die Funktion folgende Eigenschaften haben: Sie muss eine Einweg-Funktion sein, d.h. die Funktionswerte sind effizient berechenbar, es gibt jedoch kein effizientes Verfahren, um aus einem Bild  $f(x)$  das Urbild  $x$  zu berechnen. Es muss ebenfalls praktisch unmöglich sein, ein paar verschiedener Eingabewerte  $M$  und  $M'$  zu finden, die auf denselben Hash-Wert abgebildet werden, d.h.  $H(M)=H(M')$ .

## 3.5.1 Hash-Funktionen

Realisiert durch wiederholte Anwendung einer Kompressionsfunktion  $f$ .



Hash-Funktionen werden in der Regel durch eine Folge gleichartiger Kompressionsfunktionen realisiert, durch welche die Eingabe  $M$  blockweise zu einem Hash-Wert verarbeitet wird. Um Eingaben variabler Länge zu verarbeiten, wendet man die Komprimierungsfunktion iterierend an. Die Berechnung startet mit einem festgelegten Initialwert. Das Padding sorgt dafür, den letzten Block aufzufüllen, falls die Nachricht  $M$  nicht mehr dazu reicht, einen ganzen Block zu füllen.

## 3.5.1 Hash-Funktionen

### **SHA-1 (Secure Hash Algorithm 1)**

Anerkannter Standard (NIST) seit 1993  
Sequentielle Verarbeitung der Eingabe in 512-Bit-Blöcken,  
80 Verarbeitungsschritte pro Block  
erzeugt 160 Bit langen Hashwert

### **MD-5 (message digest 5)**

Vorgeschlagen von R. Rivest  
Sequentielle Verarbeitung der Eingabe in 512-Bit-Blöcken,  
64 Verarbeitungsschritte pro Block  
erzeugt 128 bitlangen Hashwert  
nicht ganz so sicher wie SHA-1

Der SHA-1 wurde 1993 vom amerikanischen National Institute of Standards and Technologies (NIST) veröffentlicht. Der SHA-1 verarbeitet eine Eingabe M blockweise zu einem 160-Bit Hash-Wert, wobei eine Blockgröße von 512 Bit verwendet wird. Der SHA-1 benötigt insgesamt 80 Verarbeitungsschritte pro 512 Bit Nachrichtenblock.

Die MD-Verfahren (Message Digest) wurden 1990 von R. Rivest mit dem MD-4 eingeführt und mit dem MD-5 fortgesetzt. Der MD-5 liefert einen 128-Bit Hash-Wert und arbeitet mit 512-Bit Eingabeblöcken. Im Gegensatz zum SHA-1 benötigt der MD-5 jedoch nur 64 Verarbeitungsschritte pro Block. Der MD-5 ist nicht ganz so sicher wie der SHA-1. Es wurde in Arbeiten gezeigt, wie ein Angriff auf den MD-5 durchzuführen wäre. Der dort vorgestellte Angriff erfordert zwar einigen analytischen Aufwand, ist aber prinzipiell möglich.

## 3.5.2 Message Authentication Code

Ein Hash-Wert sichert die Integrität einer Nachricht

### Integritäts-Kontrolle

1. Bilde Hash-Wert  $H(M)$  von Dokument  $M$  und füge  $H(M)$  dem Dokument  $M$  hinzu.
2. Zur Kontrolle der Integrität eines Dokumentes  $M'$  bilde dessen Hash-Wert  $H(M')$  und vergleiche diesen mit dem Hash-Wert  $H(M)$ , der dem Dokument beigefügt ist.
3. Falls  $H(M)=H(M')$  ist das Dokument unmodifiziert.

Aber schützt nicht davor, dass jemand die Nachricht abfängt, ändert und mit einem neuen Hash-Wert versieht.

Hash-Funktionen werden zur Kontrolle der Integrität gespeicherter oder gesendeter Daten verwendet. Der allgemeine Ablauf der Integritäts-Kontrolle ist wie folgt:

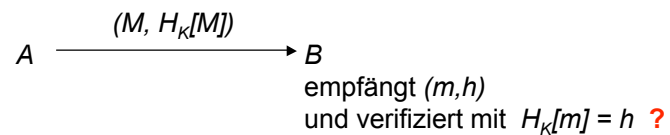
1. Der Urheber einer Nachricht oder eines Dokumentes  $M$  berechnet den Hash-Wert  $H(M)$  und hinterlegt diesen zusammen mit  $M$ . Im Falle einer Nachricht  $M$  werden also  $M$  und  $H(M)$  versendet.
2. Zur Kontrolle der Integrität eines Dokumentes  $M'$  wird zunächst dessen Hash-Wert  $h'=H(M')$  berechnet und dieses Ergebnis mit dem Hashwert verglichen, der dem Dokument beigefügt ist.
3. Falls  $h=h'$ , wird davon ausgegangen, dass  $M=M'$ .

Hash-Werte sichern die Integrität einer Nachricht oder eines Dokumentes. Sie schützen jedoch nicht davor, dass eine Nachricht abgefangen, vom Angreifer verändert und mit einem neuen Hash-Wert versehen wird.

## 3.5.2 Message Authentication Code

### Message Authentication Code (MAC):

Hash-Funktion parametrisiert mit geheimen Schlüssel  
(z.B. MD-5, HMAC)



Der Empfänger trennt Nachricht und MAC und berechnet ebenfalls aus Nachricht  $M$  und Schlüssel  $K$  einen  $MAC'$ , den er mit dem empfangenen MAC vergleicht.

Um einen authentischen Ursprung eines Dokumentes bzw. einer Nachricht zu sichern, kann man Message Authentication Codes (MAC) verwenden. Ein MAC ist eine Hashfunktion, die zusätzlich noch einen geheimen Schlüssel verwendet. Der Schlüssel ist nur den beiden Kommunikationspartnern bekannt. Der Empfänger einer Nachricht oder eines Dokumentes überprüft den MAC, indem er ebenfalls aus dem gerade empfangenen Dokument und dem ihm bekannten geheimen Schlüssel den MAC-Wert berechnet.

## **3.5.2 Message Authentication Code**

Bei Übereinstimmung weiß der Empfänger

- dass der Sender echt ist, da nur mit dem Schlüssel K derselbe Hash-Wert herauskommt,
- dass die Nachricht nicht modifiziert wurde, sonst würde ein anderer Hash-Wert resultieren

Bei Wertgleichheit ist das Dokument authentisch. Außerdem weiß er, dass die Nachricht/das Dokument nicht modifiziert wurde, da sonst der Hash-Wert unterschiedlich wäre.

## 3.5.2 Message Authentication Code

*Beispiel MD5-MAC:*

Kombination von Originaldokument und Schlüssel zu neuem Dokument, auf das die Hash-Funktion ausgeführt wird. Schlüssel wird nicht zum Verschlüsseln verwendet, sondern nur zur Modifikation des Originaldokumentes.

*Beispiel HMAC:*

Gegeben sei eine Hash-Funktion  $H$  und Schlüssel  $K$ . Der HMAC für eine Nachricht  $M$  ergibt sich dann aus:

$$HMAC(M) = H(K \parallel p1 \parallel H(K \parallel p2 \parallel M))$$

wobei  $p1$  und  $p2$  spezielle Bit-Strings sind. Ein Beispiel ist der HMAC-MD5.

Ein Beispiel ist der MD5-MAC. Hier ist die übliche Vorgehensweise, den Schlüssel als Bestandteil der zu hashenden Daten zu verwenden. Dies geschieht durch eine Kombination der Originaldaten und des Schlüssels. Dieses neu erzeugte Dokument wird dann als Eingabe für den Hash-Wert genommen. Der Schlüssel wird also nicht zum Verschlüsseln verwendet, sondern nur zur Modifikation des ursprünglichen Dokumentes.

Die Idee des HMAC-Verfahrens besteht darin, den Schlüssel zur Beeinflussung des Initialwertes der Kompressionsfunktion der verwendeten Hash-Funktion zu benutzen. Die verwendete Hash-Funktion wird hierbei als Black-Box betrachtet und nicht modifiziert. Der HMAC einer Nachricht  $M$  unter Verwendung des geheimen Schlüssels  $K$  ergibt sich als  $HMAC(M) = H(K \parallel p1 \parallel H(K \parallel p2 \parallel M))$ . Hierbei sind  $p1$  und  $p2$  spezielle Bit-Strings. Für die Hash-Funktion  $H$  könnte beispielsweise ein MD5 oder ein SHA-1 Hash verwendet werden.

## 3.5.3 Digital Signature Standard

### Digital Signature Standard (DSS)

1994 vom NIST als Standard festgelegt

arbeitet mit Hash-Funktion SHA-1 und mit  
asymmetrischen Chiffrier-Verfahren **DSA**  
(*Digital Signature Algorithm*)

**DSA**: Seine Sicherheit beruht auf der Schwierigkeit, den diskreten  
Logarithmus zu berechnen:

welches  $x$  löst  $a^x \bmod n = b$  ?

Ein Beispiel für ein spezielles Signierverfahren ist der Digital Signature Standard (DSS). Der DSS wurde vom National Institute of Standards (NIST) der USA als Standard festgelegt. Zentrales Element des DSS ist der Digital Signature Algorithm (DSA), der von der NSA entwickelt wurde. Der DSA basiert auf dem Problem des diskreten Logarithmus.

### 3.5.3 Digital Signature Standard

Alice möchte das Dokument M signieren. Konstruiere zunächst öffentlichen und privaten Signierschlüssel:

1. Wähle Primzahl  $q$  mit  $2^{159} < q < 2^{160}$
2. Wähle  $t$  aus  $\{0, \dots, 8\}$  und Primzahl  $p$  mit  $2^{511+64t} < p < 2^{512+64t}$ , so dass  $q$  ein Teiler von  $p-1$ .
3. Wähle  $j$ ,  $1 < j < p-1$  und berechne  $g = j^{(p-1)/q} \bmod p$ . Wenn  $g = 1$ , wähle neues  $j$ .
4. Wähle **geheimen Schlüssel**  $x$  mit  $1 \leq x \leq q-1$
5. Berechne **öffentlichen Schlüssel**  $y = g^x \bmod p$ .

Alice möchte das Dokument M signieren. Die Generierung ihres öffentlichen und privaten Signierschlüssels erfolgt in den fünf auf der Folie gezeigten Schritten.

### 3.5.3 Digital Signature Standard

Alice möchte das Dokument  $M$  signieren, ihr öffentlicher Signierschlüssel ist  $K_E$ , ihr privater Schlüssel  $K_D$ .

1. Alice generiert eine Zufallszahl  $0 < k < q$ .
2. Berechne  $r = (g^k \bmod p) \bmod q$
3. Berechne  $k^{-1} \bmod q$
4. Berechne  $s = k^{-1}(H(M) + K_D r) \bmod q$

Die Signatur für  $M$  ist dann das Paar  $(r,s)$ , die Alice an den Empfänger schickt.

Sind die Signierschlüssel generiert, kann eine Nachricht mit Hilfe des privaten Signierschlüssels signiert werden.

### 3.5.3 Digital Signature Standard

Der Empfänger (Bob) verifiziert die Signatur  $(r,s)$  mit dem öffentlichen Schlüssel  $K_E$  von Alice.

1. Prüfe, ob  $1 \leq r \leq q$  und  $1 \leq s \leq q$ .
2. Berechne  $w = s^{-1} \bmod q$
3. Berechne  $u_1 = (w H(M)) \bmod q$  und  
 $u_2 = (r w) \bmod q$
4. Berechne  $v = ((g^{u_1} K_E^{u_2}) \bmod p) \bmod q$
5. Falls  $v = r$ , dann ist die Signatur verifiziert.

Zur Verifikation der Signatur wird dann der öffentliche Signierschlüssel benutzt. Die einzelnen Berechnungsschritte zur Verifikation sind auf der Folie gezeigt.

## 3.6 Schlüsselverwaltung

<b>Probleme:</b>	Erzeugung
	Speicherung
	Verteilung/Austausch/Übermittlung
von	geheimen <b>Sitzungsschlüsseln</b> (symmetrisch) für effiziente Chiffrierung,
	<b>öffentlichen Schlüsseln</b> (asymmetrisch) für Authentisierung

Die Sicherheit von Verschlüsselungsverfahren beruht sehr stark auf der sicheren Verwaltung der geheimen Schlüssel. Sind die geheimen Schlüssel leicht zu ermitteln, so helfen auch die Verschlüsselungsverfahren nicht mehr, die Vertraulichkeit der Daten zu gewährleisten. Daher spielen die Schlüsselerzeugung und -verwaltung eine wichtige Rolle. Die Hauptaufgaben der Schlüsselverwaltung können in der sicheren Erzeugung, Speicherung bzw. Archivierung, und der Verteilung von geheimen Schlüsseln angesehen werden. Außerdem spielt auch die Zertifizierung von öffentlichen Schlüsseln bei asymmetrischen Verfahren eine wichtige Rolle.

## 3.6.1 Schlüsselerzeugung

### **Symmetrische Verfahren**

benötigen gute Zufallszahlengeneratoren zur Erzeugung des geheimen Schlüssels.

Häufig Pseudozufallsgeneratoren, so dass Zufallszahl zumindest für einen Angreifer nicht vorhersehbar, basieren auf einem Startwert, der geeignet gewählt werden muss.

Techniken zur Zufallszahlgenerierung sind natürliche Prozesse (atmosphärisches Rauschen, radioaktiver Zerfall, Entladung von Kapazitäten etc.)

Hardware-Komponenten (Zeit zwischen zwei Tastenschlägen, Analyse der Mausbewegungen, Kombination aus Uhrzeit der lokalen Systemuhr, der Systemidentifikation und dem aktuellen Datum etc.)

Für die Schlüsselerzeugung in symmetrischen Verschlüsselungssystemen benötigt man gute Zufallsgeneratoren. Da echte Zufallsgeneratoren aufwendig zu realisieren sind, werden in der Praxis häufig Pseudozufallsgeneratoren eingesetzt, weil es ausreicht, wenn die Schlüsselwahl für einen Angreifer nicht vorhersehbar ist. Ein Pseudozufallszahlengenerator erzeugt keine Zufallszahlen im eigentlichen Sinne, sondern berechnet aus einem Startwert eine Zahl, deren Entstehung für einen Außenstehenden nicht nachvollziehbar ist. Allerdings muss ein geeigneter Startwert zur Initialisierung des Generators gefunden werden.

Techniken zur Schlüsselerzeugung, d.h. einer Zufallszahl, können recht zuverlässig, aber aufwendig sein, wenn man natürliche Prozesse ausnutzt. Beispiele sind das atmosphärische Rauschen, radioaktiver Zerfall, Entladung von Kapazitäten etc. Hardware-Komponenten eignen sich zur Generierung von Pseudozufallszahlen. Diese lassen sich beispielsweise durch Messung der Zeit zwischen Tastaturanschlägen, durch die Analyse von Mausbewegungen, der Systemzeit, der Systemidentifikation etc. erstellen.

## **3.6.1 Schlüsselerzeugung**

### **Asymmetrische Verfahren**

höherer Berechnungsaufwand zur Schlüsselerzeugung  
als bei symmetrischen Verfahren

häufig werden große Primzahlen benötigt, für die  
aufwendige Primzahltests durchzuführen sind (z.B. RSA  
oder DSS)

Die Schlüsselerzeugung im asymmetrischen Fall erfordert in der Regel einen höheren Berechnungsaufwand. Für die meisten in der Praxis vorkommenden Verfahren (RSA, DSA) werden große Primzahlen benötigt, für die aufwendige Primzahltests durchzuführen sind.

## 3.6.2 Schlüsselspeicherung

Sichere Aufbewahrung kryptographischer Schlüssel

- Benutzer muss den Schlüssel selbst verwalten (z.B. Passwort oder PIN merken)
- Speicherung auf Chipkarten
- Aufteilen des Schlüssels (z.B. einen Teil auf der Chipkarte, der andere auf dem PC)
- Speichern der Schlüssel in Dateien durch das Betriebssystem (z.B. Schlüsselringe in PGP, Passwörter)

Die sichere Speicherung der Schlüssel ist eine weitere wichtige Aufgabe der Schlüsselverwaltung. In vielen Systemen ist der Benutzer selbst für die Speicherung zuständig. Im einfachsten Fall muss er sich den Schlüssel im Gedächtnis merken (z.B. Passwörter). Chipkarten bieten eine weitere Möglichkeit zur Speicherung von Schlüsseln. Hierbei werden die Schlüssel im ROM des Chip gespeichert und können mit speziellen Lesegeräten ausgelesen werden. Die Sicherheit der Schlüsselspeicherung kann weiter erhöht werden, wenn der Schlüssel auf mehrere Komponenten aufgeteilt wird. Zum Beispiel kann ein Teil auf dem Chip einer Chipkarte, der andere auf dem PC verwaltet werden. Nur wenn alle Schlüsselteile vorhanden sind, hat man den Gesamtschlüssel. Häufig werden Schlüssel in Dateien gespeichert, die vom Betriebssystem verwaltet werden (z.B. die Schlüsselringe beim Verschlüsselungsprogramm Pretty Good Privacy (PGP) oder bei Passwörtern). Der Zugriff auf diese Dateien muss besonders geschützt werden.

### 3.6.3 Schlüsselaustausch nach Diffie-Hellman

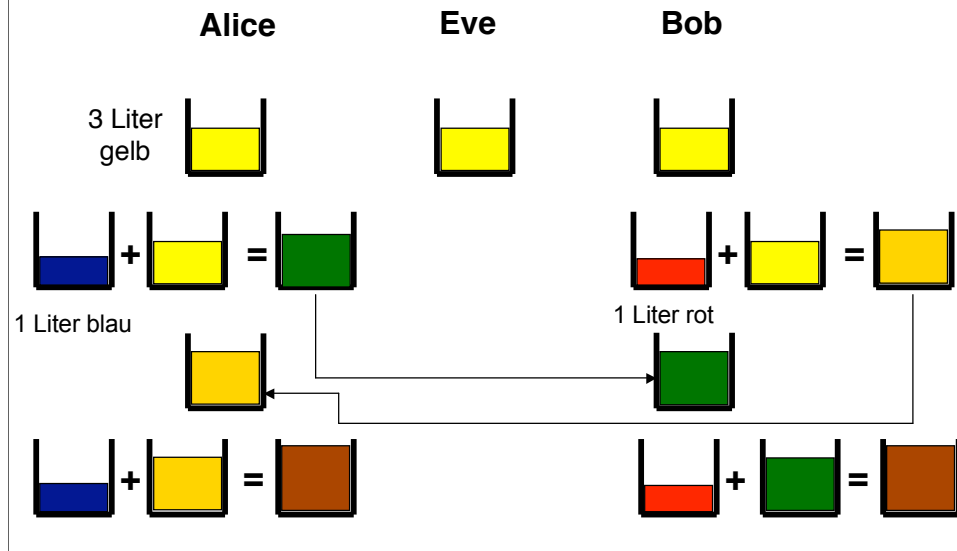
Diffie-Hellman Schlüsselaustausch 1976

**Ziel:** Vereinbarung eines neu erzeugten, geheimen Sitzungsschlüssel K zwischen zwei Partnern

Nicht zum Ver- und Entschlüsseln gedacht, d.h. kein asymmetrisches Verschlüsselungsverfahren, dient nur zur dezentralen Verwaltung von Schlüsseln

Das DH-Schlüsselaustauschprotokoll wurde 1976 in Stanford von Whitfield Diffie, Martin E. Hellman und Ralph Merkle erdacht. Eine Einwegfunktion dient Alice und Bob dazu, sich einen Schlüssel K, den Sitzungsschlüssel, für die nachfolgende Verständigung zu verschaffen. Dieser ist dann ein Geheimnis, das nur diesen beiden bekannt ist, und das nicht über ein unsicheres Medium transportiert werden muss. Das DH-Verfahren ist nicht zum Ver- und Entschlüsseln verwendbar, obwohl es auch öffentliche und private Schlüsselpaare verwendet. Die Funktionalität des DH-Verfahrens besteht allein in der dezentralen Vereinbarung eines gemeinsamen Schlüssels.

### 3.6.3 Schlüsselaustausch nach Diffie-Hellman



Nehmen wir an Bob möchte mit Alice einen geheimen Schlüssel über einen unsicheren Nachrichtenkanal aushandeln, der von Eve abgehört werden kann. Wir betrachten nun die Schlüssel als Farben (Beispiel aus S.Singh, Geheime Botschaften). Alice und Bob haben zunächst einen 3-Liter Kanister mit gelber Farbe. Dies kann auch Eve bekannt sein. Wenn Alice und Bob einen Geheimschlüssel erstellen wollen, so rühren sie jeweils für sich einen Liter frei gewählte, aber geheimgehaltene Farbe in den Kanister. Alice kippt beispielsweise 1-Liter blauer Farbe hinzu, Bob einen Liter roter Farbe. Alice schickt Bob ihren Kanister mit der neuen Farbmischung, und Bob schickt Alice seine neue Farbe. Alice fügt nun zu dem von Bob geschickten Kanister einen Liter ihrer geheimen Farbe hinzu, und Bob kippt zu Alice Mischung einen Liter seiner geheimen Farbe. Beide Mischungen haben dann dieselbe Farbe, da sie zu gleichen Anteilen Gelb, Blau und Rot enthalten. Diese Farbe dient dann als gemeinsamer Schlüssel. Alice hat keine Ahnung, welche Farbe Bob zugefügt hat und Bob weiß nicht die geheime Farbe von Alice. Eve hingegen kann, selbst wenn sie beide Kanister abfängt, die endgültige Farbe nicht herausfinden. Sie muss die geheimen Farben von Bob und Alice wissen, um die Schlüsselfarbe herzustellen.

### 3.6.3 Schlüsselaustausch nach Diffie-Hellman

#### DH-Verfahren:

Alice und Bob wollen einen geheimen Sitzungsschlüssel  $K$  über einen abhörbaren Kanal aushandeln.

Gemeinsame, nicht geheime Informationen:

1. Sie wählen eine große Primzahl  $p$
2. Sie wählen eine Zahl  $g$  mit  $0 < g < p$   
*Empfohlen:*  $g$  ist *Generator von  $p$* , d.h. für jedes  $i$  aus  $[1, p-1]$  gibt es ein  $k$  aus  $[0, p-1]$  mit  $i = g^k \bmod p$

Formaler gesehen, funktioniert das DH-Verfahren wie folgt: Alice und Bob wählen eine große Primzahl  $p$  und eine Zahl  $g$ , die größer gleich 0 und kleiner  $p$  ist. Die Zahl  $g$  ist beliebig und darf öffentlich bekannt sein. Es ist jedoch empfohlen,  $g$  als einen Generator von  $p$  zu wählen.

### 3.6.3 Schlüsselaustausch nach Diffie-Hellman

#### Dezentrale Berechnung des Schlüssels

3. Alice wählt nun  $a$ , eine Zufallszahl aus  $[0, p-1]$  und hält diese geheim.  
Bob wählt  $b$ , eine Zufallszahl aus  $[0, p-1]$  und hält diese geheim.
4. Alice berechnet nun  $A=g^a \pmod{p}$ . Bob berechnet  $B=g^b \pmod{p}$ .
5. Alice sendet das Ergebnis  $A$  an Bob. Bob sendet das Ergebnis  $B$  an Alice.
6. Um den nun gemeinsam zu benutzenden Sitzungsschlüssel zu bestimmen, potenzieren sie beide jeweils für sich das jeweils empfangene Ergebnis mit ihrer geheimen Zufallszahl modulo  $p$ .  
Das heißt: Alice berechnet  $K=B^a \pmod{p}$ , und Bob berechnet  $K=A^b \pmod{p}$ .

Alice wählt sich eine Zufallszahl  $a$  und hält sie geheim. Aus  $a$  berechnet sie mit der Einwegfunktion die Zahl  $A = g^a$  und schickt sie an Bob. Der verfährt ebenso, indem er eine geheime Zufallszahl  $b$  wählt, daraus  $B = g^b$  berechnet und an Alice schickt. Alice wendet die Einwegfunktion mit ihrer Geheimzahl  $a$  auf  $B$  an, Bob tut gleiches mit seiner Geheimzahl  $b$  und der empfangenen Zahl  $A$ . Das Ergebnis  $K$  ist in beiden Fällen dasselbe, weil die Einwegfunktion kommutativ ist:  $g^{a*b} = g^{b*a}$ . Aber selbst Bob kann Alices Geheimnis  $a$  nicht aus den ihm vorliegenden Daten rekonstruieren, Alice wiederum Bobs Geheimnis  $b$  nicht ermitteln, und ein Lauscher, der  $g$  kennt und sowohl  $A$  als auch  $B$  mitgelesen hat, vermag daraus weder  $a$  noch  $b$  noch  $K$  zu berechnen.

### 3.6.3 Schlüsselaustausch nach Diffie-Hellman

Beispiel 1 in Zahlen:

1. Alice und Bob wählen  $g = 2$ ,  $p = 11$ .
2. Alice wählt  $a = 9$ , Bob wählt  $b = 5$  und behalten  $a$  und  $b$  geheim.
3. Alice berechnet  $A = g^a = 2^9 \pmod{11} = 512 \pmod{11} = 6$   
Bob berechnet  $B = g^b = 2^5 \pmod{11} = 32 \pmod{11} = 10$
4. Alice sendet Bob:  $A = 6$ ,  
Bob sendet Alice:  $B = 10$ .
5. Alice berechnet  $B^a \pmod{11} = 10^9 \pmod{11} = 10$   
Bob berechnet  $A^b \pmod{11} = 6^5 \pmod{11} = 10$

Diese Folie zeigt ein Beispiel mit einfachen Zahlen.

### 3.6.3 Schlüsselaustausch nach Diffie-Hellman

Beispiel 2 in Zahlen:

1. Alice und Bob wählen  $g = 11$ ,  $p = 347$ .
2. Alice wählt  $a = 240$ , Bob wählt  $b = 39$  und behalten  $a$  und  $b$  geheim.
3. Alice berechnet  $A = g^a = 11^{240} \pmod{347} = 49$   
Bob berechnet  $B = g^b = 11^{39} \pmod{347} = 285$
4. Alice sendet Bob:  $A = 49$ ,  
Bob sendet Alice:  $B = 285$ .
5. Alice berechnet  $B^a = 285^{240} \pmod{347} = 268$   
Bob berechnet  $A^b = 49^{39} \pmod{347} = 268$

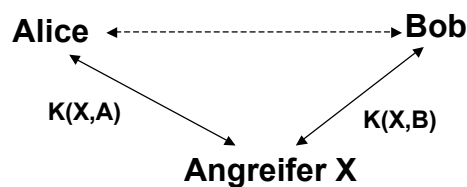
Ein weiteres Beispiel mit etwas größeren Zahlen.

### 3.6.3 Schlüsselaustausch nach Diffie-Hellman

**Achtung:** Diffie-Hellman authentifiziert die Kommunikationspartner nicht

#### Man-in-the-Middle-Angriff

Angreifer X gibt sich als Bob bzw. Alice aus und vereinbart gemeinsame Schlüssel  $K(X,A)$  bzw.  $K(X,b)$ .



Beim DH-Verfahren werden die Kommunikationspartner jedoch nicht authentifiziert. Dadurch ist das Verfahren Bedrohungen in Form von Man-in-the-Middle-Angriffen ausgesetzt. Nehmen wir an, ein Angreifer X hat es geschafft, sich gegenüber Alice als Bob und gegenüber Bob als Alice auszugeben. Der Angreifer handelt dann mit Alice einen Schlüssel  $K(X,A)$  und mit Bob einen Schlüssel  $K(X,B)$  aus. Dann ergibt sich die auf der Folie gezeigte Situation: Bob glaubt mit Alice einen gemeinsamen Schlüssel ausgetauscht zu haben und verschlüsselt Nachrichten an Alice mit dem Schlüssel  $K(X,B)$ . Angreifer X kann diese Nachricht an Alice abfangen und mit dem Schlüssel  $K(X,B)$  entschlüsseln. Er kann die Nachricht lesen und gegebenenfalls modifizieren und sie dann mit dem Schlüssel  $K(X,A)$  verschlüsseln und an Alice schicken. Alice kann nicht erkennen, dass sie mit X und nicht mit Bob kommuniziert hat.

### 3.6.3 Schlüsselaustausch nach Diffie-Hellman

*Sicherheit:*

$f(x) = g^x \bmod p$  wirkt als *Einbahn-Funktion*

Berechnung des *diskreten Logarithmus* erfordert exponentiellen Aufwand (vgl. DSA)

Das DH-Verfahren beruht auf dem diskreten Logarithmus-Problem. Das heißt es erfordert exponentiellen Aufwand, den Wert  $x$  bei gegebener Gleichung  $y = g^x \bmod p$  zu berechnen.

## 3.6.5 Zertifizierung

Asymmetrische Chiffrierung:

jeder erzeugt sein eigenes Schlüsselpaar und publiziert den öffentlichen Schlüssel

**Aber:** wie erfährt B auf sichere Weise den öffentlichen Schlüssel von A?

! Verhindern, dass X seinen eigenen öffentlichen Schlüssel als „Schlüssel von A“ präsentieren kann !

*Öffentlicher Schlüssel muss authentisch sein*

### **Zertifizierung**

Bei asymmetrischer Verschlüsselung erzeugt jeder sein eigenes Schlüsselpaar und publiziert den öffentlichen Schlüssel, der von anderen zur Verschlüsselung benutzt werden kann. Aber wie kann eine Person B sicher sein oder auf sichere Weise erfahren, dass der für A publizierte öffentliche Schlüssel wirklich zu A gehört? Person B muss sich sicher sein können, dass kein Angreifer X seinen selbst erzeugten öffentlichen Schlüssel als den öffentlichen Schlüssel von A ausgeben kann. Es muss also die Authentizität der öffentlichen Schlüssel gewährleistet werden. Dies geschieht durch die Zertifizierung von öffentlichen Schlüsseln.

## 3.6.5 Zertifizierung

Zertifikat sichert die Zuordnung von öffentlichem Schlüssel und Person.

Zertifikat ist Beglaubigung (wie Ausweis, Versicherungskarte, Geburtsurkunde etc.) durch eine Behörde, die die Identität bestätigt.

Zertifikat besteht aus Daten, anhand derer die Gültigkeit des öffentlichen Schlüssels festgestellt werden kann.

Zertifikat besteht aus

- öffentlichem Schlüssel
- Zertifikatsdaten (Daten zur Identität des Benutzers, wie Name, ID etc.)
- digitaler Unterschrift (bestätigt das Daten zum öffentlichen Schlüssel gehören)

Mit einem Zertifikat wird die zweifelsfreie Zuordnung eines öffentlichen Schlüssels zu einer natürlichen oder juristischen Person sichergestellt. Ein Zertifikat ist eine Beglaubigung, wie beispielsweise der Ausweis, eine Versicherungskarte, die Geburtsurkunde etc., die von einer Behörde ausgestellt wird, die die Identität einer Person bestätigen kann. Ein Zertifikat besteht aus Daten, anhand der die Gültigkeit des öffentlichen Schlüssels festgestellt werden kann. Bestandteil der Daten ist der öffentliche Schlüssel, Zertifikatsdaten wie die der Name des Benutzers, seine Benutzer-ID usw. Außerdem hat eine Zertifikat immer (mindestens eine) digitale Unterschrift, die die Zertifikatsdaten durch eine dritte Person oder Behörde beglaubigt. Mit der digitalen Unterschrift wird nicht die Echtheit des Zertifikats bestätigt, sondern dass die unterschriebenen Daten zur Identität an den öffentlichen Schlüssel gebunden sind.

## 3.6.5 Zertifizierung

**Zertifikatformat X.509** besteht aus

**Versionsnummer:**

verwendetes Zertifikatformat bestimmt enthaltene Informationen

**Öffentlicher Schlüssel:**

Öffentlicher Schlüssel des Zertifikatinhabers,  
Algorithmuskennung des Verschlüsselungssystems,  
Schlüsselparameter

**Seriennummer:**

eindeutige Identifizierung des Zertifikats

Die im deutschen Signaturgesetz festgelegten Inhalte eines Zertifikats entsprechen im Wesentlichen den im X.509 Standard festgelegten. Der X.509-Standard legt fest, welche Informationen in das Zertifikat aufgenommen werden. Alle X.509-Zertifikate enthalten die folgenden Daten:

-X.509-Versionsnummer: Beschreibt verwendetes Zertifikatformat. Davon hängt ab, welche Informationen im Zertifikat angegeben werden können.

-Öffentlicher Schlüssel: Der öffentliche Schlüssel des Zertifikatinhabers zusammen mit einer Algorithmuskennung, die das Verschlüsselungssystem bestimmt, dem der Schlüssel angehört, und alle Schlüsselparameter.

-Seriennummer des Zertifikats: Dem Zertifikat ist eine eindeutige Seriennummer zugeordnet, die es von allen anderen ausgestellten Zertifikaten unterscheidet.

## 3.6.5 Zertifizierung

### **Eindeutige Kennung des Zertifikatinhabers:**

Eindeutiger Name DN (distinguished name), besteht aus mehreren Unterabschnitten, z.B.

CN (common name)= Bob,

OU (organizational unit) =Total Network Security  
Division

O (organization) = Network Associatex Inc.

C (country) = US

### **Gültigkeitsdauer:**

Anfangs- und Ablaufdatum

-Eindeutige Kennung des Zertifikatinhabers: Der Name muss eindeutig sein und besteht aus mehreren Unterabschnitten.

-Gültigkeitsdauer des Zertifikats: Beinhaltet das Anfangs- und Ablaufdatum des Zertifikats und gibt somit an, wann ein Zertifikat abläuft.

## 3.6.5 Zertifizierung

**Eindeutiger Name des Zertifikatausstellers:**

Name des Zertifikatausstellers (normalerweise eine CA)

**Digitale Unterschrift des Ausstellers:**

digitale Unterschrift desjenigen, der das Zertifikat ausgestellt hat (benutzt wird der private Schlüssel des Ausstellers)

**Kennung für den Unterschriftenalgorithmus:**

Kennzeichnet den von für das Unterschreiben des Zertifikats verwendeten Algorithmus.

-Eindeutiger Name des Zertifikatausstellers: Eindeutiger Name dessen, der das Zertifikat unterschrieben hat. Hierbei handelt es sich normalerweise um eine Zertifizierungsstelle.

-Die digitale Unterschrift des Ausstellers: Die Unterschrift, für die der private Schlüssel des Dritten verwendet wird, der das Zertifikat ausgestellt hat.

-Die Kennung für den Unterschriftenalgorithmus: Kennzeichnet den für das Unterschreiben des Zertifikats verwendeten Algorithmus.

## 3.6.5 Zertifizierung

### Zertifizierungsstelle

(*Trust Center / Certification Authority, CA*)

ist zuständig für

- Ausstellung* von Zertifikaten, z.B. gemäß X.509, und Signierung mit privatem Schlüssel der Zertifizierungsstelle
- Publizieren* der Zertifikate in öffentlichem Verzeichnis, z.B. X.500
- Speicherung* der Zertifikate z.B. auf Chipkarte, zusammen mit eigenem öffentlichen Schlüssel und evtl. dem privaten Schlüssel des Teilnehmers

Eine Zertifizierungsstelle (*Trust Center* bzw. *Certification Authority*) bietet Dienste zur Ausstellung von Zertifikaten an. Für jeden Teilnehmer wird von der Zertifizierungsstelle ein Zertifikat (z.B. X.509) erzeugt und mit dem privaten Schlüssel der Zertifizierungsstelle signiert. Das Zertifikat und ggf. öffentlicher und privater Schlüssel werden auf eine Signaturkomponente (z.B. Chipkarte) gespeichert. Über einem öffentlichen Verzeichnis wie X.500 muss jedem Teilnehmer Auskunft darüber gegeben werden, ob ein bestimmtes, ihm zugeordnetes Zertifikat noch gültig ist.

## 3.6.5 Zertifizierung

### Öffentliche-Schlüssel-Infrastruktur

(*public-key infrastructure, PKI*)

ist hierarchisch strukturiertes Netz von Zertifizierungsstellen,  
z.B. zweistufige Hierarchie gemäß deutschem *Signaturgesetz*

Wurzel: Regulierungsbehörde für  
Telekommunikation und Post (genehmigt CAs)

Zweite Stufe: z.B. Deutsche Telekom / TeleSec,  
Signtrust / Deutsche Post eBusiness,  
Bundesnotarkammer, DATEV eG,  
Steuerberaterkammer Berlin, etc.

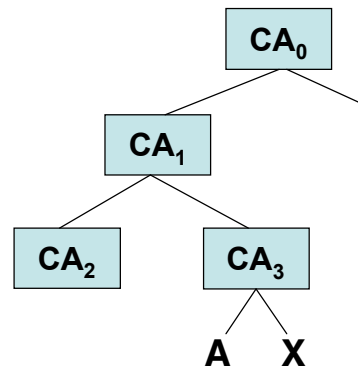
Eine Public-Key Infrastruktur (PKI) ist ein hierarchisch strukturiertes Netz aus Zertifizierungsstellen. Im deutschen Signaturgesetz wird eine zweistufige Zertifizierungsinfrastruktur festgelegt. Die Wurzel bildet die Regulierungsbehörde für Post und Telekommunikation, die den Betrieb von Zertifizierungsstellen genehmigt.

## 3.6.5 Zertifizierung

### Zertifikat-Überprüfung

Benötigt öffentlichen Schlüssel der Zertifizierungsstelle

X.509 Zertifikat von X	
Public Key von X	
Seriennummer des Zertifikates	
Gültigkeitszeitraum	
Eindeutiger Name von X	
Eindeutiger Name der Zertifizierungsstelle	
Digitale Unterschrift der Zertifizierungsstelle	

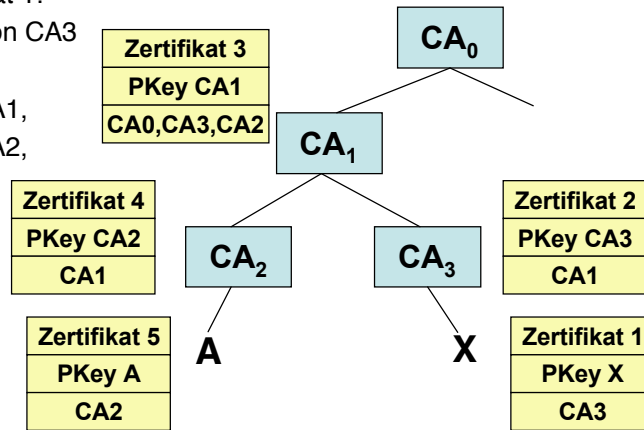


Um ein von einer bestimmten Zertifizierungsstelle ausgestelltes Zertifikat überprüfen zu können, benötigt man den öffentlichen Schlüssel der Zertifizierungsstelle. Möchten zwei Teilnehmer A und X, die derselben CA zugeordnet sind, ihre jeweiligen Zertifikate verifizieren, können sie die Zertifikate des Partners unmittelbar überprüfen, da beiden der öffentliche Schlüssel der CA bekannt ist und sie von dessen Authentizität überzeugt sind.

## 3.6.5 Zertifizierung

Voraussetzung: Wechselseitige Zertifizierung der CAs

A überprüft Zertifikat 1:  
 Public Key für X von CA3  
 signiert,  
 PK für CA3 von CA1,  
 PK für CA1 von CA2,  
 PK für CA2 ist A  
 bekannt



Die Zertifikate der CAs in der Hierarchie werden wie folgt beglaubigt: Jeder CA in der Hierarchie ist ein Zertifikat zugeordnet, das von seinem direkten Vorgängerknoten und auch von allen unmittelbaren Nachfolgern in der Hierarchie signiert ist.

Sind A und X unterschiedlichen CAs zugeordnet, ist die Überprüfung komplexer. Nehmen wir an, A möchte das Zertifikat von X überprüfen. Teilnehmer A kennt den öffentlichen Schlüssel von CA<sub>2</sub>, d.h. A ist von der Authentizität des öffentlichen Schlüssels von CA<sub>2</sub> überzeugt. Die Instanz CA<sub>2</sub> hat den öffentlichen Schlüssel von CA<sub>1</sub> und CA<sub>1</sub> hat den Schlüssel von CA<sub>3</sub> signiert. Somit kann A mit der Kenntnis des öffentlichen Schlüssels von CA<sub>2</sub> das Zertifikat, das CA<sub>2</sub> für CA<sub>1</sub> ausgestellt hat, prüfen. Mit der Kenntnis des öffentlichen Schlüssels von CA<sub>1</sub> kann im nächsten Schritt die Signatur des Zertifikats für CA<sub>3</sub> überprüft werden. Da CA<sub>3</sub> ihrerseits den Schlüssel von X signiert hat, ist A mit dem öffentlichen Schlüssel von CA<sub>2</sub> in der Lage, auch diese Signatur zu prüfen und den öffentlichen Schlüssel von X zu überprüfen.

## 3.6.5 Zertifizierung

Zertifikate werden beispielsweise in folgenden Anwendungen benutzt

**SSL** (Secure Socket Layer) verwendet digitale Zertifikate für den Schlüsselaustausch, die Serverauthentifizierung und die Clientauthentifizierung.

**Sichere E-Mail** verwenden digitale Zertifikate für digitale Unterschriften und für den Austausch von Schlüsseln zur Nachrichtenver- und -entschlüsselung (z.B. S/MIME)

**Secure Electronic Transaction ( SET )** Standard für sichere Kreditkartenzahlungen im Internet. Die Verwendung digitaler Zertifikate gestattet sichere, private Verbindungen zwischen Karteninhabern, Händlern und Banken.

Anwendungen, die Verschlüsselungssysteme mit öffentlichen Schlüsseln für den Schlüsselaustausch oder digitale Unterschriften verwenden, müssen die benötigten öffentlichen Schlüssel mit Hilfe von digitalen Zertifikaten erhalten. Es gibt zahlreiche Internetanwendungen dieser Art:

### **SSL**

Ein Protokoll, das Datenschutz und Integrität für die Datenübertragung zur Verfügung stellt. SSL verwendet digitale Zertifikate für den Schlüsselaustausch, die Serverauthentifizierung und wahlweise für die Clientauthentifizierung. Bei der Clientauthentifizierung muss ein Server das digitale Zertifikat eines Clients authentifizieren, bevor sich der Client anmelden oder auf bestimmte Ressourcen zugreifen darf. Der Server fordert das digitale Zertifikat des Clients während des SSL-Handshake an und authentifiziert es. Zu diesem Zeitpunkt kann der Server auch bestimmen, ob er der Zertifizierungsinstanz, die das digitale Zertifikat an den Client ausgegeben hat, vertraut.

### **Sichere E-Mail**

Viele E-Mail-Systeme, die Standards wie *S/MIME* (Secure/Multipurpose Internet Mail Extensions) für sichere E-Mail einsetzen, verwenden digitale Zertifikate für digitale Unterschriften und für den Austausch von Schlüsseln zur Nachrichtenver- und -entschlüsselung.

### **Secure Electronic Transaction ( SET )**

Der Standard SET ist für sichere Kreditkartenzahlungen in nicht sicheren Netzen (Internet) konzipiert. Digitale Zertifikate werden für Karteninhaber (elektronische Kreditkarten) und Händler verwendet. Die Verwendung digitaler Zertifikate in SET gestattet sichere, private Verbindungen zwischen Karteninhabern, Händlern und Banken. Die erstellten Transaktionen sind sicher und nicht anfechtbar und können nicht gefälscht werden. Die Händler erhalten keine Kreditkartendaten, die missbraucht oder gestohlen werden können.

### 3.6.5.1 Zertifikate in Java

#### **keytool**

Verwaltung von Schlüsseln und Zertifikaten

Schlüssel werden im **keystore** gespeichert

Zertifikate entsprechen dem X.509-Standard

keystore GUI Tool: ks

**keytool** ist ein Java-Tool, das zur Verwaltung von privaten und öffentlichen Schlüsseln sowie Zertifikaten in Java verwendet wird. Schlüssel und Zertifikate werden im sogenannten *keystore* gespeichert, welcher in der Default-Implementierung als Datei realisiert ist, die mit einem Passwort geschützt ist. Die erstellten Zertifikate entsprechen dem X.509 Standard. Es gibt ein GUI tool mit denen Zertifikate erstellt werden können.

### 3.6.5.1 Zertifikate in Java

**Ziel:** Zertifikate erstellen mit Keystore GUI Tool *ks*

1. neuen Keystore für Benutzer erzeugen (File/New)
2. neue Schlüssel für Benutzer erzeugen (Keys/New)
  - 2.1 key alias wird CN des distinguished name
3. Schlüssel auswählen und als self-signed-certificate exportieren (Certificates/Export)
  
4. neuen Keystore und Schlüssel für CA erzeugen
5. Schlüssel auswählen und als sel-signed-certificate exportieren

Die Folie zeigt den Ablauf einer Zertifikaterstellung mit dem keytool.

### **3.6.5.1 Zertifikate in Java**

5. Benutzer-Zertifikat als *trustee* in CA hinzufügen  
(Trustees/add)
6. Benutzer-Zertifikat auswählen und zertifizieren  
(Certificates/Create)
  - 6.1 CA-Passwort eingeben
7. Altes Benutzer-Zertifikat entfernen (Trustees/Delete)
  
8. Im Benutzer-Keystore Schlüsseintrag wählen
9. CA-zertifiziertes Benutzer-Zertifikat importieren  
(Certificates/Import)

### **3.6.5.1 Zertifikate in Java**

10. CA-Zertifikat importieren
11. CA-Zertifikat als Trustee hinzufügen
12. certificate chain prüfen (Keys/Verify Chain)

Zertifikate ansehen mit *keytool -printcert -v -file cert*

## 3.7 PGP und GnuPG

**PGP** („*pretty good privacy*“) = Software für  
Verschlüsselung,  
Schlüsselerzeugung,  
Schlüsselverwaltung (Bekanntmachung,  
Speicherung, ...)  
Erstellung und Verifikation digitaler Unterschriften  
<http://www.pgpi.org/>

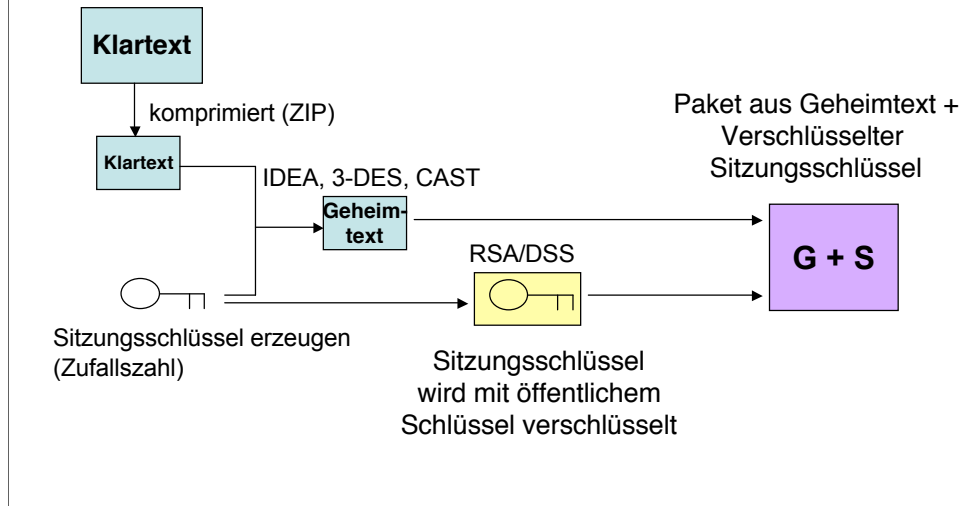
**GnuPG**  
Open-Source Version  
<http://www.gnupg.org/>

PGP (Pretty Good Privacy) wurde von Phil Zimmermann entwickelt um allen Personen die Möglichkeit zu geben ihre Privatsphäre zu schützen. Es handelt sich um ein Ver- und Entschlüsselungsprogramm. Eine [Open-Source](#) Version des Programms wurde unter dem Namen [GnuPG](#) von Werner Koch entwickelt.

Um die Interoperabilität zu gewährleisten, wurde das von PGP verwendete Dateiformat festgehalten und Erweiterungen definiert. Dieses Format nennt sich OpenPGP und wird von PGP und GnuPG größtenteils eingehalten.

## 3.7.1 PGP

### Funktionsweise der PGP-Verschlüsselung

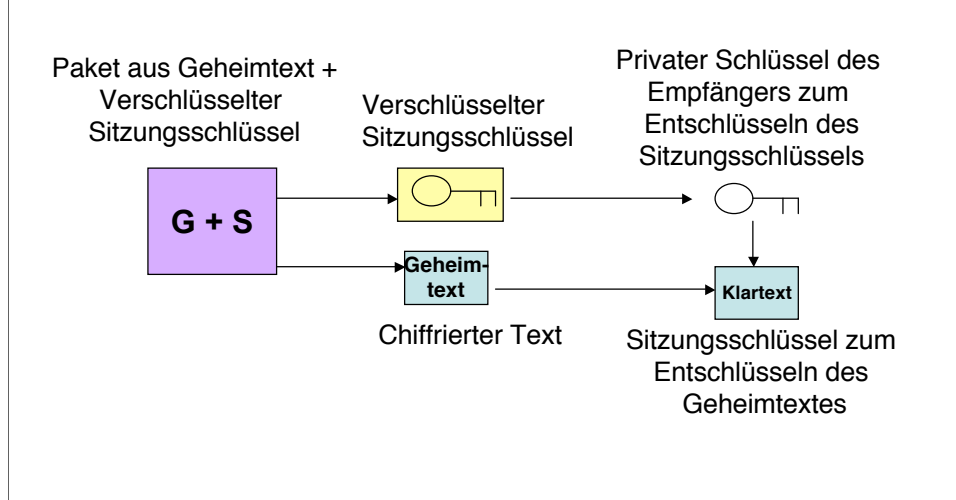


Beim Verschlüsseln von Klartext mit PGP wird dieser Text zuerst komprimiert. PGP komprimiert die Informationen vor der Verschlüsselung mit Hilfe des Freeware-ZIP-Verfahrens. PGP erstellt dann einen *Sitzungsschlüssel*, einen Geheimschlüssel zum einmaligen Gebrauch. Dieser Schlüssel ist eine Zufallszahl, die aus den willkürlichen Bewegungen, die mit der Maus ausgeführt werden, und den ausgeführten Tastenanschlägen erstellt wird. Mit diesem Sitzungsschlüssel und einem symmetrischen Verschlüsselungsverfahren (IDEA, 3-DES oder CAST) wird der Klartext zu einem chiffrierten Text verschlüsselt.

Nach der Verschlüsselung der Daten wird der Sitzungsschlüssel selbst mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Zur Generierung des asymmetrischen Schlüsselpaares kann der RSA oder DSS verwendet werden. Dieser mit einem öffentlichen Schlüssel verschlüsselte Sitzungsschlüssel wird zusammen mit dem chiffrierten Text an den Empfänger übertragen.

## 3.7.1 PGP

### Funktionsweise der PGP-Entschlüsselung



Die Entschlüsselung läuft in umgekehrter Reihenfolge ab. In der PGP-Kopie des Empfängers wird dessen privater Schlüssel verwendet, um den verschlüsselten Sitzungsschlüssel wiederherzustellen. Diesen verwendet PGP anschließend, um den symmetrisch verschlüsselten chiffrierten Text zu entschlüsseln.

## 3.7.1 PGP

In PGP gibt es zwei Schlüsselbünde, die in zwei Dateien auf der Festplatte gespeichert sind.

**öffentliches Schlüsselbund**

enthält alle öffentlichen Schlüssel

**privates Schlüsselbund**

enthält alle privaten Schlüssel

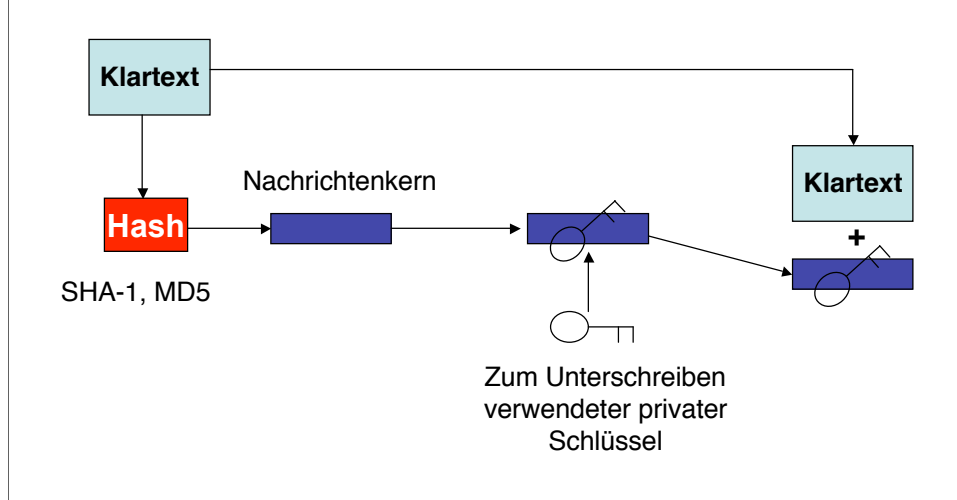
Private Schlüssel werden mit einem Passwort gesichert und verschlüsselt gespeichert.

1. privater Schlüssel D
2. Passwort P, Hashwert H(P) ist Schlüssel für symmetrische Verschlüsselung (secret key)
3. verschlüsseln von D mittels secret key

In PGP werden die Schlüssel auf der Festplatte in zwei Dateien gespeichert, einer Datei für öffentliche und einer für private Schlüssel. Diese Dateien werden als *Schlüsselbunde* bezeichnet. Private Schlüssel werden mit einem Passwort verschlüsselt gespeichert. Dazu bildet PGP einen Hashwert aus dem gewählten Passwort, den sogenannten "Secret Key". Mit dem Secret Key wird der private Schlüssel von PGP verschlüsselt und dann im privaten Schlüsselbund gespeichert. Somit wird der private Schlüssel direkt durch den Secret Key und indirekt durch das dem Secret Key zugrunde liegende Passwort geschützt. PGP fragt jedesmal, wenn man entschlüsselt oder signiert, das Passwort ab, da in beiden Fällen der Private Key, bzw. Secret Key zum Einsatz kommt.

## 3.7.1 PGP

### Digitale Unterschriften in PGP



PGP kann auch zum Signieren verwendet werden: Mit PGP wird der Klartext zunächst durch eine Hash-Funktion (SHA-1 (default) oder MD5) auf ein Datenelement fester Länge gebracht, das auch als *Nachrichtenkern* bezeichnet wird. Anhand des Nachrichtenkerne und des privaten Schlüssels zum Signieren wird dann die digitale Unterschrift erstellt. Die Unterschrift und der Klartext werden zusammen übertragen. Bei Erhalt der Nachricht berechnet der Empfänger mit Hilfe von PGP den Nachrichtenkern neu und überprüft damit auch die Unterschrift.

## 3.7.1 PGP

PGP benutzt keine offizielle PKI, sondern basiert auf privatem Vertrauensnetz (*web of trust*)

Jeder PGP-Benutzer kann Zertifikate für öffentliche Schlüssel anderer PGP-Benutzer erstellen.

Gültigkeit des Zertifikats hängt vom persönlichen Vertrauen zum Zertifikataussteller ab.

Key-Signing-Parties sind Treffen, auf denen öffentliche Schlüssel signiert werden.

PGP benutzt keine offizielle PKI, sondern basiert auf dem Vertrauensmodell des *Web of Trust*. In diesem Modell kann jeder PGP-Benutzer als Zertifizierungsinstanz agieren, d.h. jeder PGP-Benutzer kann die Gültigkeit des Zertifikats für den öffentlichen Schlüssel eines anderen PGP Benutzers bestätigen. Ein derartiges Zertifikat ist aber nur dann für den anderen Benutzer echt, wenn die überprüfende Person von der abhängigen Partei als ein autorisierter Schlüsselverwalter anerkannt wird. (Das heißt, man vertraut hinsichtlich der Gültigkeit von Benutzerschlüsseln nur auf die Meinung von PGP-Benutzern, denen man traut. Andernfalls ist diese Meinung über die Gültigkeit anderer Schlüssel nicht von Belang.) Wenn ein Benutzer den Schlüssel eines anderen unterzeichnet, wird er zum Schlüsselverwalter für diesen Schlüssel. Bei diesem Vorgang entsteht allmählich ein Netz bzw. ein *Web of Trust*.

## 3.7.1 PGP

öffentliche Schlüssel im Schlüsselbund eines Benutzers enthalten Informationen, ob der Benutzer

- 1) den Schlüssel als gültig betrachtet
- 2) die *Vertrauensstufe*, d.h. inwieweit dem Aussteller des Zertifikats für den öffentlichen Schlüssel getraut wird

Vertrauensstufen in PGP

*implizites Vertrauen*: höchste Vertrauensstufe, gilt für das eigene Schlüsselpaar

Auf jedem öffentlichen Schlüsselbund eines Benutzers werden folgende Informationen gespeichert:

- Ob der Benutzer einen bestimmten Schlüssel als gültig betrachtet
- Die Vertrauensstufe, die der Benutzer dem Schlüssel zuordnet, d. h. die Eignung des Schlüsseleigentümers als Zertifizierungsinstanz für andere Schlüssel.

Es handelt sich also um ein System, das auf der Grundlage des guten Rufes beruht: Manche Personen haben den Ruf, vertrauenswürdige Unterschriften zu geben, daher wird ihnen vertraut, wenn sie die Gültigkeit anderer Schlüssel bestätigen.

Die unterschiedlichen Vertrauensstufen in PGP sind:

Implizites Vertrauen: Die höchste Vertrauensstufe eines Schlüssels, *implizites* Vertrauen, ist das Vertrauen in das eigene Schlüsselpaar. Bei PGP wird davon ausgegangen, dass bei Besitz eines privaten Schlüssels auch den Aktionen der zugeordneten öffentlichen Schlüssel vertraut werden muss. Alle Schlüssel, die vom Schlüssel mit dem impliziten Vertrauen unterschrieben wurden, sind echt.

## 3.7.1 PGP

Vertrauensstufen in PGP für öffentliche Schlüssel anderer Benutzer

**volles Vertrauen:**

„Eine Signatur dieses Benutzers ist so gut wie die eigene“

**eingeschränktes Vertrauen:**

„Benutzer prüft Schlüssel bevor er sie signiert“

**kein Vertrauen (oder nicht vertrauenswürdig):**

„Der Benutzer ist bekannt, dass er bei der Signierung keine genügende Überprüfung durchführt“

Einem öffentlichen Schlüssel eines anderen Benutzers können drei Vertrauensstufen zugeordnet werden:

- *Volles Vertrauen*
- *Eingeschränktes Vertrauen*
- *Kein Vertrauen (oder Nicht vertrauenswürdig)*

## **3.7.1 PGP**

Öffentlicher Schlüssel ist gültig, wenn

    eine Unterschrift einer Person der vollen  
Vertrauensstufe vorliegt,

    oder

    zwei Unterschriften von zwei Personen der  
eingeschränkten Vertrauensstufe.

Um die Gültigkeit eines Schlüssels zu bestätigen, sind bei PGP eine Unterschrift der vollen Vertrauensstufe oder zwei Unterschriften der eingeschränkten Vertrauensstufe erforderlich. Zwei eingeschränkt vertrauenswürdige Unterschriften mit einer voll vertrauenswürdigen Unterschrift gleichzusetzen, ähnelt dem Vorgehen eines Händlers, der zwei unterschiedliche Identitätsnachweise fordert.

## 3.7.1 PGP

Von PGP werden zwei verschiedene Zertifikatsformate anerkannt:

- PGP-Zertifikate
- X.509-Zertifikate

Unterschiede zwischen den Zertifikatsformaten

PGP-Zertifikate kann man selbst ausstellen, X.509 Zertifikate benötigen eine Zertifizierungsstelle

X.509 Zertifikate unterstützen nur eine einzige digitale Unterschrift zur Bestätigung der Schlüsselgültigkeit

X.509 Zertifikate haben nur einen einzigen Namen für den Schlüsseleigentümer

Von PGP werden zwei verschiedene Zertifikatsformate anerkannt: PGP-Zertifikate und X.509-Zertifikate. Es bestehen einige Unterschiede zwischen einem X.509- und einem PGP-Zertifikat. Die wichtigsten sind jedoch die folgenden:

- PGP-Zertifikat kann man selbst erstellen, während die Ausstellung eine X.509-Zertifikats bei einer Zertifizierungsinstanz angefordert werden muss.
- X.509-Zertifikate unterstützen grundsätzlich nur einen einzigen Namen für einen Schlüsseleigentümer.
- X.509-Zertifikate unterstützen nur eine einzige digitale Unterschrift zur Bestätigung der Schlüsselgültigkeit. Bei PGP können mehrere verschiedene Personen das Schlüssel-/Identifikationspaar unterschreiben, und somit bestätigen, dass der öffentliche Schlüssel eindeutig zum angegebenen Eigentümer gehört.

## **3.7.2 GnuPG**

Open-Source Variante von PGP

GNU-Projekt und freie Software unter der GNU General Public License

Implementiert den OpenPGP-Standard

<http://www.gnupg.org/>

GnuPG ist eine von Werner Koch entwickelte Open-Source Verschlüsselungssoftware. Sie implementiert den OpenPGP-Standard, der als Erweiterung von PGP entwickelt wurde.

## 3.7.2 GnuPG

Die ersten Schritte mit GnuPG

Erzeugen von Schlüsseln

**gpg -gen-key**

Exportieren des öffentlichen Schlüssels

**gpg -a -o AusgabeDatei -export UID**

Das Ergebnis kann man dann in der Welt verbreiten.

Anzeigen der Schlüssel im Schlüsselbund

**gpg -list-keys**

Auf diesen Folien sind die grundlegenden ersten Befehle gegeben, mit denen man Schlüssel erzeugen, signieren etc. kann. Für eine genaue Übersicht, sollte man in die GnuPG-Dokumentation schauen.

## 3.7.2 GnuPG

Importieren öffentlicher Schlüssel

**gpg -import Schlüssel**

Nimmt öffentlichen Schlüssel **Schlüssel** ins Schlüsselbund auf

Signieren des öffentlichen Schlüssels

**gpg -edit-key UID** (Schlüssel auswählen)

**sign** (signiere Schlüssel)

**save**

Vertrauen von Schlüsseln anzeigen

**gpg -edit-key UID**

**trust**

## 3.7.2 GnuPG

Verschlüsseln von Dateien

**gpg - -encrypt Datei**

Entschlüsseln von Dateien

**gpg - - decrypt Datei**

GnuPG gibt es als Plugin für Mail-Clients, die die Ver- und Entschlüsselung übernehmen. Ebenso das Signieren.

z.B. KMail (KDE), Enigmail (Netscape, Mozilla) etc.

## 3.8 CrypTool

Open-Source-Projekt **CrypTool**

eLearning-Programm, mit dem kryptographische Verfahren angewendet und analysiert werden können.

Klassische (zum Beispiel das Caesar) und moderne (beispielsweise das RSA- und das AES-Verfahren, sowie auf Elliptischen Kurven basierende Verfahren) Verschlüsselungsverfahren.

<http://www.cryptool.de/>

Das Programm CrypTool ist ein eLearning-Programm für Windows, mit dem kryptographische Verfahren angewendet und analysiert werden können. Die umfangreiche Online-Hilfe wird durch Szenarien/Tutorials und ein Skript mit weiterführenden Informationen ergänzt.