

# 12. Foliensatz Computernetze

Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences  
(1971–2014: Fachhochschule Frankfurt am Main)  
Fachbereich Informatik und Ingenieurwissenschaften  
[christianbaun@fb2.fra-uas.de](mailto:christianbaun@fb2.fra-uas.de)

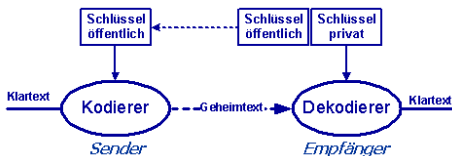
# Lernziele dieses Foliensatzes

- Einführung in die Kryptologie – Teil 2
  - Asymmetrische Verfahren
    - RSA
  - Schlüsselverteilung
    - Diffie-Hellmann
    - Elgamal
    - Public-Key-Infrastructure (PKI)
    - Web of Trust
  - Kryptografische Hashfunktionen
    - SHA-2
    - MD5

# Asymmetrische Verfahren

Bildquelle: <http://www.nordwest.net>

- Problem der Schlüsselverteilung bei symmetrischen Verfahren



- Asymmetrischen Verfahren arbeiten nicht mit nur einem Schlüssel
  - Es wird immer mit einem Paar aus öffentlichem und privatem Schlüssel gearbeitet
- Die populären asymmetrischen Verfahren basieren auf der Schwierigkeit folgender zahlentheoretischer Probleme:
  - Bestimmung des diskreten Logarithmus
  - Faktorisierung eines Produktes großer Primzahlen
- Vorteil: Kein sicherer Kanal für den Schlüsseltausch nötig
- Nachteil: Hoher Aufwand für Ver- und Entschlüsselung

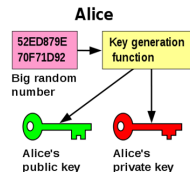
# Rivest-Shamir-Adleman (RSA)

- Das verbreitetste Public-Key-Verfahren
- Entwickelt 1977/78 von Ron Rivest, Adi Shamir und Len Adleman
- 1983-2000 für die USA patentiert
- Die Sicherheit basiert auf dem Problem, eine große ganze Zahl in ihre Primfaktoren zu zerlegen
  - Darum darf die Schlüssellänge nicht zu klein
- Wird heute u.a. genutzt, für:
  - Internet- und Telefonie-Infrastruktur (x.509-Zertifikate)
  - Übertragungsprotokolle (IPSec, TLS, SSH)
  - E-Mail-Verschlüsselung (PGP, S/MIME)
  - RFID Chip auf dem deutschen Reisepass
  - Austausch/Senden von geheimen symmetrischen Schlüsseln
- Schlüssellänge ist  $> 1024$  Bit

# Algorithmus zur Schlüsselerzeugung bei RSA



RSA-Modul  $N$ , Verschlüsselungsexponent  $e$ , Entschlüsselungsexponent  $d$



- 1 Alice wählt 2 große zufällige Primzahlen  $p$  und  $q$  mit  $p \neq q$ 
  - In der Praxis werden sehr große Primzahlen verwendet
  - Schlüsselgröße typischerweise  $\geq 1024$  Bit
- 2 Alice multipliziert die Primzahlen und erhält  $N = p * q$ 
  - Die Faktorisierung einer großen Zahl  $N$ , also die Zerlegung in ihre Primzahlen ist sehr rechenaufwendig
- 3 Berechne die Eulersche  $\varphi$ -Funktion von  $N \implies \varphi(N) = (p - 1) * (q - 1)$
- 4 Wähle eine zu  $\varphi(N)$  teilerfremde Zahl  $e$ , für die gilt  $1 < e < \varphi(N)$
- 5 Berechne den Entschlüsselungsexponenten  $d$  als Multiplikativ Inverses von  $e$  bezüglich des  $\varphi(N)$ 
  - Anwendung des erweiterten Euklidischen Algorithmus zur Findung des ggT zweier positiver Zahlen

# Algorithmus zur Schlüsselerzeugung bei RSA – Beispiel

- 1 Alice wählt die beiden Primzahlen  $p = 11$  und  $q = 13$
- 2 Multipliziert die Primzahlen und erhält das RSA-Modul  $N = p * q = 143$
- 3 Eulersche  $\varphi$ -Funktion von  $N$

$$\varphi(143) = (p - 1) * (q - 1) = 120$$

- 4 Alice wählt die zu  $\varphi(143)$  teilerfremde Zahl  $e = 23$  als Verschlüsselungsexponent  
 $\implies$  Der **öffentliche Schlüssel** ist  $(e, N) = (23, 143)$
- 5 Berechnung den Entschlüsselungsexponenten  $d$  als Multiplikativ Inverses von  $e$  bezüglich des  $\varphi(N)$ :

$$e * d + k * \varphi(N) = 1 \implies d = 47, k = -9$$

Das Berechnen von  $d$  und  $k$  geht mit dem erweiterten Euklidischen Algorithmus.  $d$  ist der geheime Exponent, während  $k$  nicht weiter benötigt wird.

$\implies$  Der **private Schlüssel** ist  $(d, N) = (47, 143)$

# Verschlüsselung und Entschlüsselung bei RSA

$K$  = Klartext,  $C$  = Geheimtext,  $N$  = RSA-Modul,  $e$  = Verschlüsselungsexponent,  $d$  = Entschlüsselungsexponent

- Nachrichten verschlüsseln

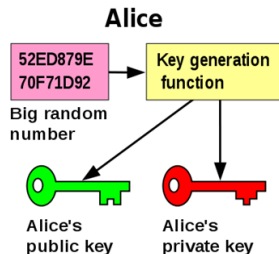
$$C = K^e \bmod N$$

- Nachrichten entschlüsseln

$$K = C^d \bmod N$$

- Beispiel

- Klartext  $K = 7$
- Öffentlicher Schlüssel ( $e = 23, N = 143$ )
- Privater Schlüssel ( $d = 47, N = 143$ )
- Verschlüsseln:  $C = 7^{23} \bmod 143 = 2$
- Entschlüsseln:  $K = 2^{47} \bmod 143 = 7$



# Schlüsselverteilung

- 2 wichtige Fragen:
  - ① **Wie erhalten zwei Teilnehmer bei den gemeinsam zu benutzenden Schlüssel bei symmetrischer Verschlüsselung?**  
⇒ Diffie-Hellmann-Algorithmus oder Elgamal-Algorithmus
  - ② **Wie wissen die Teilnehmer bei asymmetrischer Verschlüsselung welcher öffentliche Schlüssel einem bestimmten Teilnehmer gehört?**  
⇒ Public-Key-Infrastructure (PKI) oder Web of Trust



# Diffie-Hellmann

- Kein Verfahren zur Nachrichtenverschlüsselung, sondern ein Algorithmus zum sicheren **Schlüsselaustausch** über einen unsicheren Kanal
- Erster Public-Key-Algorithmus – 1976 vorgestellt
- Der Algorithmus erzeugt bei 2 oder mehr Partnern einen gemeinsamen Schlüssel, den die Partner zur symmetrischen Verschlüsselung verwenden
- Wir nennen die Kommunikationspartner (Parteien) **Alice** und **Bob**

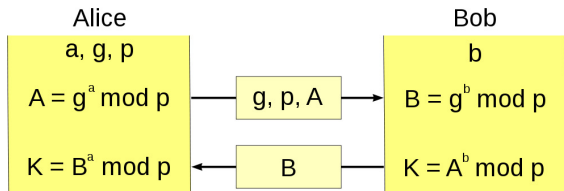
# Schlüsselerzeugung bei Diffie-Hellmann

- Alice und Bob einigen sich auf eine große Primzahl  $p$  und eine natürliche Zahl  $g$ , die kleiner  $p$  ist
  - $p$  und  $g$  können sie öffentlich bekannt geben ( $\implies$  öffentlicher Schlüssel), also über den unsicheren Kanal austauschen
- Alice wählt eine große, ganzzahlige Zufallszahl  $a$  und berechnet  $A := g^a \bmod p$ 
  - Alice sendet  $A$  an Bob
- Bob wählt eine große, ganzzahlige Zufallszahl  $b$  und berechnet  $B := g^b \bmod p$ 
  - Bob sendet  $B$  an Alice
- Alice berechnet den privaten (geheimen) Schlüssel  $K := B^a \bmod p$
- Bob berechnet den privaten (geheimen) Schlüssel  $K' := A^b \bmod p$
- Es gilt:  $K = K'$

$$\begin{aligned} K &= B^a \bmod p \\ &= (g^b)^a \bmod p \\ &= g^{a \cdot b} \bmod p \\ &= (g^a)^b \bmod p \\ &= A^b \bmod p \\ &= K' \end{aligned}$$

# Schlüsselerzeugung bei Diffie-Hellmann – Beispiel

- Alice schlägt  $p = 11$  und  $g = 4$  vor und wählt als Wert für  $a = 3$
- Bob akzeptiert  $p$  und  $g$  und wählt als Wert für  $b = 5$
- Alice berechnet  $A = g^a \bmod p = 4^3 \bmod 11 = 64 \bmod 11 = 9$
- Bob berechnet  $B = g^b \bmod p = 4^5 \bmod 11 = 1024 \bmod 11 = 1$
- Alice berechnet  $K = B^a \bmod p = 1^3 \bmod 11 = 1 \bmod 11 = 1$
- Bob berechnet  $K' = A^b \bmod p = 9^5 \bmod 11 = 59049 \bmod 11 = 1$
- $K = K'$



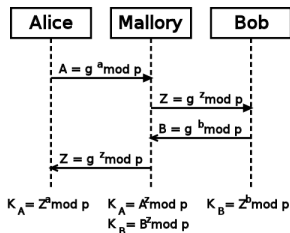
Bildquelle: Wikipedia

# Sicherheit bei Diffie-Hellmann

- Ein Angreifer kann  $p$  und  $g$  und  $A$  und  $B$  kennen
  - Ohne  $a$  oder  $b$  ist die Berechnung des Schlüssels  $K$  nicht möglich
  - Die Zahlen  $a$  und  $b$  kann man bestimmen, indem die Gleichungen
$$A := g^a \bmod p$$
und
$$B := g^b \bmod p$$
nach  $a$  bzw.  $b$  aufgelöst werden
- Das ist die einzige bekannte für beliebige Werte anwendbare Methode, um das Diffie-Hellmann-Verfahren zu brechen
- Die Lösungen der Gleichungen zu finden ist sehr aufwendig und wird als das **Problem des diskreten Logarithmus** bezeichnet
- Die Sicherheit hängt wesentlich von der Wahl von  $p$  und  $g$  ab
  - $p$  soll sehr groß (mindestens 1024 Bit) gewählt werden
  - $g$  soll so gewählt werden, dass die Potenzen
$$g^x \bmod p$$
nach Möglichkeit eine große Menge bilden

# Gefahr des Man-in-the-Middle-Angriff bei Diffie-Hellmann

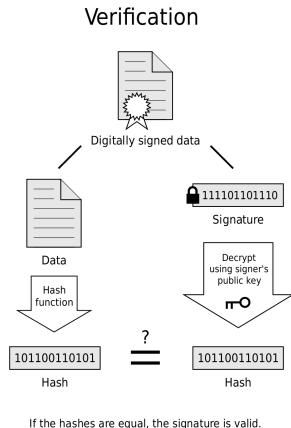
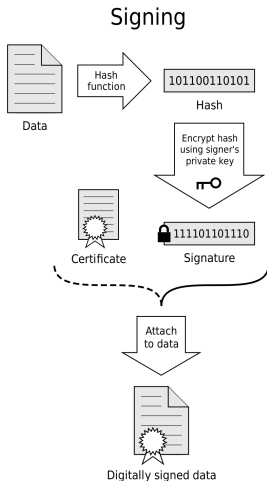
- Alice und Bob können nicht sicher sein, dass die Nachricht tatsächlich vom jeweils anderen kommt
- Der Angreifer (**Mallory**) tauscht sowohl mit Alice, als auch mit Bob einen geheimen Schlüssel aus
- Mallory fängt alle Nachrichten ab, entschlüsselt sie, verschlüsselt sie mit dem zweiten Schlüssel und sendet sie weiter
- Es wird zweimal ein Diffie-Hellmann-Schlüsselaustausch durchgeführt
- Alice und Bob gehen davon aus, mit dem jeweils Anderen zu kommunizieren
  - Mallory kann die symmetrisch verschlüsselte Kommunikation unbemerkt abhören und verändern
- Lösung: Signatur der Nachrichten z.B. mit Pretty good Privacy (PGP)



Bildquelle: <http://de.wikipedia.org>

# Digitale Signaturen (Quelle: Wikipedia)

- Verfahren zur Prüfung der Urheberschaft und Zugehörigkeit einer Nachricht
- Wurde der öffentliche Schlüssel mit einem elektronischen Zertifikat einer Person zugeordnet, kann über das öffentliche Verzeichnis des Zertifizierungsdienstes die Identität des Signaturerstellers überprüft werden
- Das ist möglich, weil es zum öffentlichen Schlüssel nur einen passenden privaten Schlüssel gibt



# Elgamal

- Abänderung des Diffie-Hellmann-Schlüsselaustauschs
- 1985 von Taher Elgamal veröffentlicht

- **Schlüsselerzeugung**

- Eine große Primzahl  $p$
- Primitivwurzel  $g$  mit  $1 < g < p$  bestimmen
  - $g$  ist eine natürliche Zahl
- Private Schlüssel ist eine beliebige natürliche Zahl  $d$ , die zu  $p - 1$  relativ prim (Teilerfremd) und kleiner  $p$  ist
  - Zwei natürliche Zahlen sind teilerfremd, wenn es keine natürliche Zahl außer 1 gibt, die beide Zahlen teilt
- Für den öffentlichen Schlüssel wird noch ein  $e$  berechnet:  $e = g^d \bmod p$
- Öffentliche Schlüssel: Tripel  $(p, g, e)$
- Der private (geheime) Schlüssel ist  $d$

3 ist eine Primitivwurzel modulo 7, da gilt

$$\begin{aligned} 3^1 &\equiv 3 \pmod{7} \\ 3^2 &\equiv 2 \pmod{7} \\ 3^3 &\equiv 6 \pmod{7} \\ 3^4 &\equiv 4 \pmod{7} \\ 3^5 &\equiv 5 \pmod{7} \\ 3^6 &\equiv 1 \pmod{7} \end{aligned}$$

Es lassen sich alle Elemente 1, 2, . . . , 6 der primen Restklassengruppe modulo 7 als Potenzen von 3 darstellen

m	Primitivwurzeln modulo m
2	1
3	2
5	2, 3
7	3, 5
11	2, 6, 7, 8
13	2, 6, 7, 11
17	3, 5, 6, 7, 10, 11, 12, 14
19	2, 3, 10, 13, 14, 15
23	5, 7, 10, 11, 14, 15, 17, 19, 20, 21
29	2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27

Die Tabelle zeigt die Primitivwurzeln modulo der Primzahlen bis 29

# Elgamal – Verschlüsselung und Entschlüsselung

## • Verschlüsselung

- Mit dem öffentlichen Schlüssel des Empfängers wählt der Sender zufällig eine natürliche Zahl  $x$  die kleiner ist als  $p - 1$
- Sender berechnet

$$Y = g^x \text{ mod } p$$

- $Y$  wird zusammen mit dem Chiffretext  $C$  an den Empfänger gesendet
- Chiffretext  $C$  wird aus dem Klartext  $M$  wie folgt berechnet

$$C = M * e^x \text{ mod } p$$

## • Entschlüsselung

- Empfänger kann den Klartext  $M$  aus der Nachricht  $(Y, C)$  berechnen

$$M = C * Y^{p-1-d} \text{ mod } p$$



# Elgamal – Beispiel

## ● Schlüsselerzeugung

- Primzahl  $p = 17$  und teilerfremde Zahl  $d = 6$ , die kleiner  $p$  ist
- Primitivwurzel  $g = 3$  die  $1 < g < p$
- Öffentlichen Schlüssel  $e$  berechnen:  $e = g^d \bmod p = 3^6 \bmod 17 = 15$
- Der öffentliche Schlüssel ist ( $p = 17, g = 3, e = 15$ )
- Der private (geheime) Schlüssel ist  $d = 6$

## ● Verschlüsselung

- In Beispiel ist der Klartext  $m = 9$
- Willkürlich sei  $x = 5$  gewählt und  $x$  ist kleiner als  $p - 1$
- Sender berechnet:  $Y = g^x \bmod p = 3^5 \bmod 17 = 5$
- Chiffretext  $C$  wird aus dem Klartext  $M$  wie folgt berechnet:  
 $C = M * e^x \bmod p = 9 * 15^5 \bmod 17 = 1$
- $Y$  wird zusammen mit dem Chiffretext  $C$  an den Empfänger gesendet

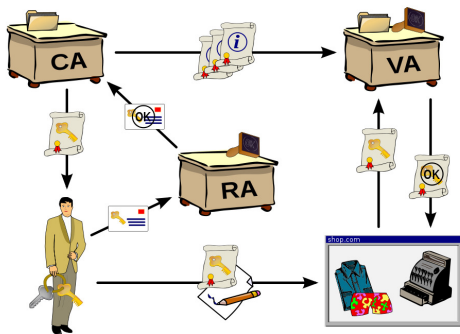
## ● Entschlüsselung

- Empfänger berechnet den Klartext  $M$  aus der Nachricht ( $Y, C$ )  
 $M = C * Y^{p-1-d} \bmod p = 1 * 5^{17-1-6} \bmod 17 = 9$

# Public-Key-Infrastructure (PKI)

Bildquelle: Wikipedia

- Benutzer erstellt Schlüsselpaar und meldet offline den öffentlichen Schlüssel bei einer RA (Registration Authority) an
- RA gibt der CA (Certificate Authority) das OK
- CA erstellt ein digitales Zertifikat bezüglich des öffentlichen Schlüssels und seinen Inhaber



- Dritte Partei kann das Zertifikat bei einer VA (Validation Authority) überprüfen und die Authentizität eines öffentlichen Schlüssels beglaubigen lassen
- Es muss Vertrauen in die involvierte CA herrschen
- Browser bringen vorab eine Liste vertrauenswürdiger CAs (z.B. VeriSign) mit

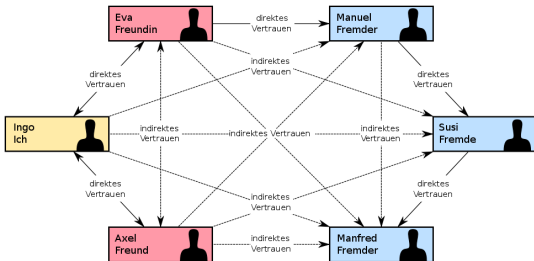
# Digitale Zertifikate – X-509-v3

- Standard für digitale Zertifikate in Public-Key-Infrastrukturen
- Struktur eines X-509-v3-Zertifikats:
  - Zertifikat
    - Version
    - Seriennummer
    - Algorithmen-ID
    - Aussteller
    - Gültigkeit (von)
    - Gültigkeit (bis)
    - Zertifikatinhaber
    - Zertifikatinhaber-Schlüsselinformationen
      - Public-Key-Algorithmus
      - Public Key des Zertifikatinhabers
    - Eindeutige ID des Ausstellers (optional)
    - Eindeutige ID des Inhabers (optional)
    - Erweiterungen
  - Zertifikat-Signaturalgorithmus
  - Zertifikat-Signatur

# Web of Trust

Bildquelle: Wikipedia

- Dezentrale Alternative zum hierarchischen PKI-System
- Jeder einzelne Teilnehmer stellt Zertifikate über Bekannte aus
  - Die Teilnehmer signieren gegenseitig ihre Schlüssel
- Verzicht auf zentrale CAs
- Schaffung eines Vertrauensgeflechts
- Geeignet für kleine Gruppen, deren Teilnehmer sich größtenteils persönlich kennen und vertrauen
- Verwendet u.a. GnuPG



# Kryptografische Hashfunktionen

- Eine **Hashfunktion** bzw. **Streuwerfunktion** bildet aus einer oft sehr großen Quellmenge eine Ausgabe aus einer typischerweise kleineren Zielmenge, den **Hashwert** bzw. **Hashcode**
- Gängige Verfahren: u.a. MD5 und SHA-2
- Einige Einsatzgebiete von Hashfunktionen:
  - Datenbanken
    - Identifikation eines Dokuments
    - Schnelles Auffinden von Daten in großen Datenspeichern
  - Prüfsummen
    - Überprüfung größerer Datentransfers
  - Kryptographie
    - Signierung
- Hashfunktionen sind Einwegfunktion und somit nicht umkehrbar
  - Anders als beim Chiffrieren, ist eine Wiederherstellung der ursprünglichen Eingabe nicht möglich

# Kriterien für eine gute Hashfunktion

Eingabe	⇒	Hashfunktion	⇒	Ausgabe
Fox	⇒	md5sum	⇒	2e1efc59375746270b4f5cc98ce31cc4
FOX	⇒	md5sum	⇒	e72505327e1f8fb10f0abb7466a66404
The red fox runs across the ice	⇒	md5sum	⇒	6b69b0ef2ac4446e1deee822f691edcf
The red fox walks across the ice	⇒	md5sum	⇒	aa9c14d467c3f7ea392fe24f7801b32f
The red Fox walks across the ice	⇒	md5sum	⇒	b0639f5a5fcd887858a4dc144b97d628

- Ausgabe der Hashfunktion ist immer gleich groß (lang)
- Qualität einer Hashfunktion ergibt sich aus ihrer Kollisionsfreiheit
  - Auch kleine Änderungen an der Eingabe führen zu großen Änderungen an der Ausgabe
  - Verschiedene Eingaben mit gleichem Hashwert sollen möglichst selten vorkommen

Kriterium	Beschreibung
<b>Chaos</b>	Ähnliche Eingabewerte sollen zu völlig unterschiedlichen Hashwerten führen
<b>Datenreduktion</b>	Speicherbedarf des Hashwertes soll deutlich kleiner sein als der des Eingabewertes
<b>Effizienz</b>	Die Funktion soll schnell berechenbar sein
<b>Surjektivität</b>	Kein Hashwert soll unmöglich sein, Jedes Ergebnis soll vorkommen können

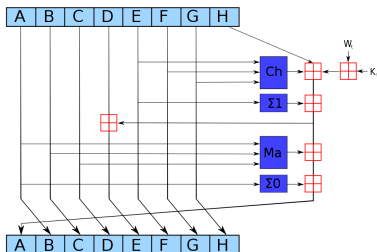
# Secure Hash Algorithm 2 (SHA-2)

- Entwickelt von der National Security Agency (NSA) und 2001 veröffentlicht
- Satz von 4 Hashfunktionen (SHA-224, SHA-256, SHA-384, SHA-512)
- Enthaltene Operationen: + (*append*), AND, OR, XOR, SHR (*rightrotate*), ROT (*rotate*)
- Es soll praktisch unmöglich sein, 2 verschiedene Nachrichten mit dem gleichen SHA-Wert zu finden
  - Bisher wurden keine Kollisionen gefunden

Algorithmus	Ausgabe	max. Nachrichtengröße	Blockgröße	Runden	Wortgröße
<b>SHA-224</b>	224 Bit	$2^{64} - 1 \approx 2$ Exabyte	512 Bit	64	32 Bit
<b>SHA-256</b>	256 Bit	$2^{64} - 1 \approx 2$ Exabyte	512 Bit	64	32 Bit
<b>SHA-384</b>	384 Bit	$2^{128} - 1$	1024 Bit	80	64 Bit
<b>SHA-512</b>	512 Bit	$2^{128} - 1$	1024 Bit	80	64 Bit

## Secure Hash Algorithm 2 – Arbeitsweise

Bildquelle: Wikipedia



$$\text{Ch}(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$$

$$\text{Ma}(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\sum_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$

$$\sum_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

Das rote Quadrat ist eine Addition modulo  $2^{32}$

$\oplus$  steht für die XOR-Operation

$\wedge$  steht für die AND-Operation

$\neg$  steht für die NOT-Operation

$\ggg$  steht für die SHR-Operation (*rightrotate*)

- 8 vordefinierte Variablen: A...H
- 64 (SHA-224/256) bzw. 80 (SHA-384/512) vordefinierte Runden-Konstanten:  $K_t$
- Jeder Durchlauf (64/80 Runden) verarbeitet 512 Bit/1024 Bit Nachrichtenblöcke
- Jeder Nachrichtenblock wird in 64 bzw. 80  $W_t$  Chunks aufgetrennt und dann expandiert
- Berechnung des Endwertes:
  - Aufsummierung der einzelnen neu erzeugten Variablen A...H aus allen Durchläufen
  - Hashwert Ergebnis:  $A + B + \dots + H$



# Secure Hash Algorithm 2 – Beispiele

## ● SHA-224

```
$ echo Franz jagt im komplett verwahrlosten Taxi quer durch Bayern | sha224sum  
f61b3c5a5555154001cda72b9f011444c5fc949ec976c297917b1adb
```

```
$ echo Frank jagt im komplett verwahrlosten Taxi quer durch Bayern | sha224sum  
6788f860277b243b24198918253fdc1762551f011af6a6c3e1fff485
```

## ● SHA-256

```
$ echo Franz jagt im komplett verwahrlosten Taxi quer durch Bayern | sha256sum  
c956e0cb0579a03c24d51e05cb3e20a45f3b5cc39e82a80dae77699840659afe
```

```
$ echo Frank jagt im komplett verwahrlosten Taxi quer durch Bayern | sha256sum  
3bbd3cc03155a4794460dbec7f8534ad773dde2dcb5750d448e13c12018033e1
```

## ● SHA-384

```
$ echo Franz jagt im komplett verwahrlosten Taxi quer durch Bayern | sha384sum  
bdbdcb27f42c1bf2fe026032b3dfa209b0685e81aba3318c1968264286cf7630d75213f2d30eed264ba39b40c3bec65b
```

```
$ echo Frank jagt im komplett verwahrlosten Taxi quer durch Bayern | sha384sum  
7e58e8d60e6d33ad8c7064ffadde95811589f13896249735a850b42de9cd4817a5871951d049918ebacf96e6e9751dc2
```

## ● SHA-512

```
$ echo Franz jagt im komplett verwahrlosten Taxi quer durch Bayern | sha512sum  
7009c4bc42268b782126d13d4f99148f332251dc2a888ca121d3158ce6565781
```

```
59e58a58aecb741b37bdd40589ba2e5b2eb05f579daaf4b7d52d952157c68c62
```

```
$ echo Frank jagt im komplett verwahrlosten Taxi quer durch Bayern | sha512sum  
7b7332c5ef2671461125026fe4b92f9f1486428cc50433ca914312183ad501d6
```

```
9fcf24517352327908e8e8bf14354c080f14366d8c4f2503ba74e90ab14cd720
```

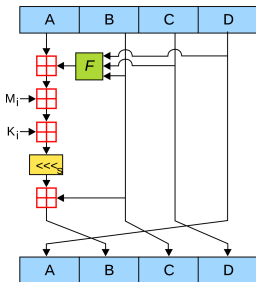
# Message-Digest Algorithm 5 (MD5)

- Entwickelt 1991 von Ronald L. Rivest vom MIT
- Gilt inzwischen nicht mehr als sicher, da es mit überschaubarem Aufwand möglich ist, unterschiedliche Nachrichten zu erzeugen, die dieselbe MD5-Prüfsumme aufweisen
  - 1996 wurde erstmals eine Kollision in MD5 gefunden
  - Es gibt  $2^{128} = 3,4 \cdot 10^{38}$  verschiedene Hash-Werte
  - Es ist dank der Kollisionsangriffe in vielen Fällen möglich, zwei Dokumente zu erstellen, die den gleichen MD5-Hash ergeben, dann das erste, legitime Dokument signieren zu lassen, und anschließend dieses durch das zweite, gefälschte Dokument auszutauschen
    - Darum ist von einer Weiterverwendung von MD5 abzuraten
- Die 128 Bit langen MD5-Hashs werden häufig als 32-stellige Hexadezimalzahl notiert

```
$ echo Franz jagt im komplett verahrlosten Taxi quer durch Bayern | md5sum  
4868ac39fdeb60e886791d6be8c0fcb3
```

```
$ echo Frank jagt im komplett verahrlosten Taxi quer durch Bayern | md5sum  
f98f6704a03727a9585e1ca62bc878e9
```

# Message-Digest Algorithm 5 – Arbeitsweise (1/2)



Quelle: Wikipedia

Initialisierungswerte

A: 01234567 B: 89abcdef

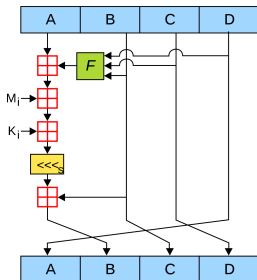
C: fedcba98 D: 76543210

- Ein Puffer (128 Bit) wird eingerichtet, der in 4 Wörter (sog. Kettenvariablen) A, B, C und D (je 32 Bit) unterteilt ist
  - Die Wörter werden mit vorgegeben (RFC 1321) Konstanten initialisiert
- Jeder Nachrichtenblock wird in 16 Wörter (je 32 Bit) unterteilt

- MD5 erzeugt aus einer Nachricht variabler Länge eine Ausgabe fester Länge (128 Bit)
  - 1 Eine 1 an die Ausgangsnachricht anhängen
  - 2 Die Ausgangsnachricht mit Nullen so auffüllen, dass ihre Länge 64 Bit davon entfernt ist, durch 512 teilbar zu sein
  - 3 Eine 64 Bit lange Zahl, die die Länge der Ausgangsnachricht codiert, anhängen
- Die Nachrichtenlänge ist jetzt durch 512 teilbar
- Die Nachricht wird in 512 Bit-Blocks unterteilt

# Message-Digest Algorithm 5 – Arbeitsweise (2/2)

Das Bild zeigt eine Operation. MD5 besteht aus 64 Operationen, gruppiert in 4 Runden mit je 16 Operationen.  $M_i$  ist ein 32 Bit-Block des Eingabestroms und  $K_i$  eine für jede Operation unterschiedliche 32 Bit-Konstante. <<< steht für bitweise Linksrotation um  $s$  (0 bis 63) Stellen, wobei  $s$  für jede Operation variiert. Das rote Quadrat ist eine Addition modulo  $2^{32}$  (stellt sicher, dass das Ergebnis nicht größer als 32 Bit ist)



Quelle: Wikipedia

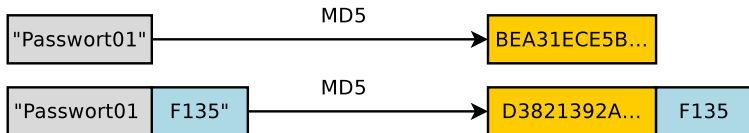
- Die Behandlung jedes 512 Bit-Nachrichtenblocks geschieht in 4 Runden
  - Jede Runde besteht aus 16 Operationen, basierend auf einer nichtlinearen Funktion  $F$ , modularer Addition und Linksrotation
    - In jeder der 16 Operationen wird ein anderer 32 Bit-Teil des Nachrichtenblocks berücksichtigt
  - Es gibt 4 mögliche  $F$ -Funktionen
    - In jeder Runde wird eine andere verwendet
      1. Runde:  $F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$
      2. Runde:  $G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$
      3. Runde:  $H(X, Y, Z) = X \oplus Y \oplus Z$
      4. Runde:  $I(X, Y, Z) = Y \oplus (X \vee \neg Z)$
- In jeder Runde wird auf das Ergebnis dieselbe Operationen mit dem nächsten Nachrichtenblock durchgeführt usw., bis zum letzten 512 Bit-Block
- Ergebnis: Die MD5-Summe (128-Bit) ist die Konkatenation der Werte von A, B, C und D

# Sicherheit von MD5

- 1996 fand der Kryptologe Hans Dobbertin am Bundesamt für Sicherheit in der Informationstechnik (BSI) die erste Kollision in MD5
  - Diese war ohne praktische Auswirkungen auf die Sicherheit, denn die beiden kollidierenden Nachrichten ergaben keinen Sinn
- 2004 fanden chinesischen Kryptologen einen Weg, Kollisionen auch für unterschiedliche Anfangs-Strings zu erzwingen
- Eine Angriffsmöglichkeit sind **Regenbogentabellen** (Rainbow Tables)
  - Die Tabellen enthalten Zeichenketten mit den zugehörigen MD5-Hashs
  - Der Angreifer durchsucht die Tabellen nach dem vorgegebenen Hashwert
  - Die Tabellen sind groß und es braucht viel Rechenaufwand ( $\implies$  GPUs), um sie zu erstellen
  - Angriffe mit Regenbogentabellen sind nur bei kurzen Passwörtern möglich
  - Eine Anwendung, die Windows-Passwörter mit Regenbogentabellen knackt ist Ophcrack
    - <http://ophcrack.sourceforge.net>

# Regenbogentabellen unwirksam machen (1/2)

- Mit **Salt** kann man Angriffe mit Regenbogentabellen unwirksam machen
- Ein zufälliger mitgespeicherter Anhang, das *Salz*, verändert das Ergebnis bei jedem Hash-Vorgang und vereitelt Angriffe mit Regenbogentabellen



- Ein Angreifer muss nicht mehr nur jedes mögliche Passwort, sondern jedes mögliche Passwort zu jedem möglichen Salzwert berücksichtigen
- Fast alle Unix-Derivate (inkl. Linux) setzen seit ca. drei Jahrzehnten Salt für Passwort-Hashs ein
  - Darum sind Angriffe mit Regenbogentabellen gegen UNIX aussichtslos

## Regenbogentabellen unwirksam machen (2/2)

- Unter Linux liegen die Passwort-Hashs in der Datei `/etc/shadow`
- Damit das System Passworteingaben mit dem Hash vergleichen kann, muss das Salt bekannt sein
- Deshalb wird es im Klartext dem gespeicherten Hash vorangestellt
  - Unter Linux sind 4.096 unterschiedliche Salt-Werte möglich
  - Jedes Passwort kann also auf 4.096 verschiedene Wege gespeichert werden
- Die Speicherung des Salts im Klartext klingt zwar widersprüchlich, aber es muss aber nicht geheim sondern nur zufällig sein
- Das Salt soll nur die Anzahl der Kombinationen für jedes einzelne mögliche Passwort aufblähen und so den Aufwand für das Anlegen der Regenbogentabellen immens vergrößern

# RIPEDM-160

- Gemeinsam mit Hans Dobbertin entwickelte Alternative zu SHA-1
  - 1996 erstmals publiziert
- Arbeitet mit 512 Bit-Nachrichtenblöcken
- Erzeugt 160 Bit-Hashwerte
- Zwei unabhängige Ablaufstränge, die am Ende verknüpft werden
- Jeder Ablaufstrang ist in 5 Runden gegliedert
  - Jeder der 5 Runden besteht aus 16 Teilrunden
- Patentfrei
- Aktuell eine der besten kryptografische Hashfunktionen
- Es existieren auch 128, 256 und 320 Bit-Versionen des Algorithmus
  - Die 256- und 320-Bit-Versionen reduzieren lediglich die Wahrscheinlichkeit von Hash-Wert-Kollisionen, bieten aber keine höhere Sicherheit als RIPEMD-128 oder RIPEMD-160
- RIPEMD-160 setzt u.a. das Verschlüsselungsprogramm TrueCrypt ein

Sehr gute Beschreibung von RIPEMD-160

<http://homes.esat.kuleuven.be/~bosselae/ripemd160.html>



# Weitere kryptografische Hashfunktionen

## ● Tiger

- 1995 von Ross Anderson und Eli Biham entwickelt
- Erzeugt 192 Bit-Hashwerte
- Nicht so weit verbreitet wie SHA oder RIPEMD-160
- Patentfrei
- <http://www.cs.technion.ac.il/~biham/Reports/Tiger/>

## ● WHIRLPOOL

- 2000 von Vincent Rijmen und Paulo S. L. M. Barreto entworfen
  - Endgültige Version 2003 veröffentlicht
- Erzeugt 512 Bit-Hashwerte
- 10 Verschlüsselungsrunden
- Der Algorithmus noch jung und bislang wenig untersucht
- Patentfrei
- <http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html>