Vorlesung

# *Sicherheit: Fragen und Lösungsansätze*

Dr. Thomas P. Ruhroth

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

# “Hashfunktion, Signatur und Schlüsselaustausch“

**[mit freundlicher Genehmigung basierend auf einem Foliensatz von Prof. Dr. Claudia Eckert (TU München)]**

**Literatur:**

Claudia Eckert: IT-Sicherheit: Konzept - Verfahren - Protokolle, 7., überarb. und erw. Aufl., Oldenbourg, 2012.

E-Book: http://www.ub.tu-dortmund.de/katalog/titel/1362263

# Agenda

- Hashfunktionen
- Schlüsselaustausch
- Signaturen

- Verfahren für Integrität und Authentizität von Nachrichten kennen.

# Hashfunktionen, Signaturen und Schlüsselaustausch

# Motivation

(Daten-) Integrität

(Informations-) Vertraulichkeit

Verfügbarkeit

Verbindlichkeit, Zurechenbarkeit

Authentizität

Privatheit

→ Hashfunktionen

→ Schlüsselaustausch

→ Verschlüsselung

→ Signaturen

→ Message-Authentication-Code

Basistechnologien werden in Protokollen/Diensten kombiniert z.B.

- IPSec, SSL, SSH, sichere eMail (PGP, S/MIME), DNSSec, ….

# 3.1 Kryptographische Hashfunktion

**Motivation:**

- Integritätsprüfung: Verfahren, um Manipulationen zu erkennen.
  - Prüfsummen (z.B. CRC) – Technische Manipulation
  - Kryptographische Hashfunktion – Intentionelle Manipulation
- Anforderung an Prüfwert (Message Digest, Fingerprint)?

# Hashfunktion

Eine kryptographische Hashfunktion ist eine nicht injektive Funktion

$H : A^*_1 \rightarrow A^k_2$,

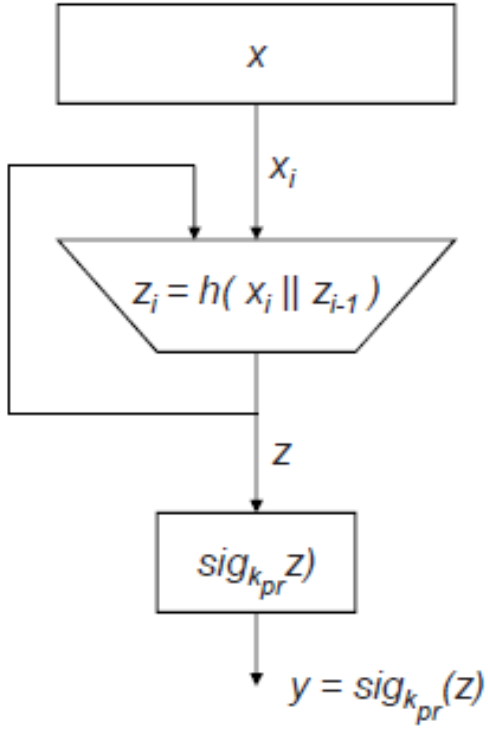
die folgende Eigenschaften erfüllt:

1. H besitzt die Eigenschaften einer Einweg-Funktion
2. Der Hashwert H(M ) = h, mit | h |= k, ist bei gegebener Eingabe M leicht zu berechnen.
3. Kollisionsresidenz
    - Schwache Kollisionsresistenz oder
        - Starke Kollisionsresistenz

# Beispiel Hash

**Lösungsansatz:**

- Nachricht x, beliebige Länge, $x_i$ Block
- Kompressionsfunktion h,
- kein Schlüsselmaterial notwendig
- h(x) kein vertraulicher Wert
- Effizientes Erzeugen eines
  Message Digest z fester Länge




**h ist eine Abbildung**

- $h : X^* \rightarrow X^n$ (Message-Digest, One-way-Hash), z.B. n = 128
  h ist eine Hashfunktion, wohlbekannt in der Informatik (wo noch?)

**Aber:**

- *h* ist nicht injektiv, deshalb sind prinzipiell Kollisionen möglich!

# Anforderungen

$\forall M \in X^*$ gilt: $H(M) = y$ ist einfach zu berechnen.

1. Einwegeigenschaft (Preimage resistance) von H:
   - Gegeben $y = H(M)$.
   - das Bestimmen des Wertes $M \in X^*$, mit $M = H^{-1}(y)$ ist nicht effizient möglich.
2. Schwache Kollisionsresistenz (second Preimage)
   - Gegeben sei $M \in X^*$, es ist nicht effizient möglich, ein $M' \in X^*$ zu finden, so dass gilt:
   - $M \neq M'$ und $H(M) = H(M')$
3. Starke Kollisionsresistenz:
   - das Finden von Paaren $M, M' \in X^*$, mit
   - $H(M) = H(M')$ ist nicht effizient möglich

# Beispiel: Geburtstagsattacke

**Bemerkung:**

- Starke Kollisionsresistenz ist schwerer zu garantieren als schwache

**Grund:**

- Angreifer hat zwei Freiheitsgrade: beide Nachrichten M, M‘ können geändert werden, um gleiche Hashwerte zu finden

**Aufwand** zum Finden einer Kollision: vgl. Geburtstagsattacke

# Beispiel: Geburtstagsattacke

**Geburtstagsattacke:**

- Wie viele Personen müssen versammelt sein, damit die Wahrscheinlichkeit, dass 2 Personen den gleichen Geburtstag haben, > 0.5 ist?
- Anzahl der möglichen Werte: 365
- Erste Vermutung: ca die Hälfte, also 183 Personen sind erforderlich
- Man kann zeigen: bei 23 Personen: Wahrscheinlichkeit > 0.5
- Bei 40 Personen: Wahrscheinlichkeit > 0.9

# Bedeutung des Geburtstagsangriffs für Hashfunktion

**Bedeutung des Geburtstagsangriffs für Hashfunktion:**

- Gegeben Hashfunktion H, Hashwert der Länge n, $2^n$ Hashwerte
- Man kann zeigen (siehe u.a. C. Paar Kapitel 11): die Anzahl der Nachrichten, die man konstruieren muss, um mit einer Wahrscheinlichkeit > 0.5 eine Kollision zu erzeugen liegt bei

$$\approx \sqrt{2^n}, \text{ also } \approx 2^{(n/2)}$$

D.h. sei H eine Hashfunktion mit Hashwerten der Länge $n = 80$

- Ein Kollisionsangriff erfordert die Berechnung von $\approx 2^{40}$ Hashwerten
- Das ist bei heutiger Rechenleistung einfach möglich

**Erkenntnis:**

- Länge n ist essentiell, Hashwert sollte mind. 160 Bit lang sein

# Beispiel für Hashfunktionen und deren Sicherheit

# Klassen von Hashfunktionen

**Klassen von Hashfunktionen:**

- Hashfunktionen basierend auf Block-Chiffren, z.B. DES
- dedizierte Hashfunktionen: u.a. MD4, MD5 mit 128-Bit Hash oder SHA (Secure Hash-Algorithm)

**Beispiel für Block-Chiffren-basierte Hashfunktion**

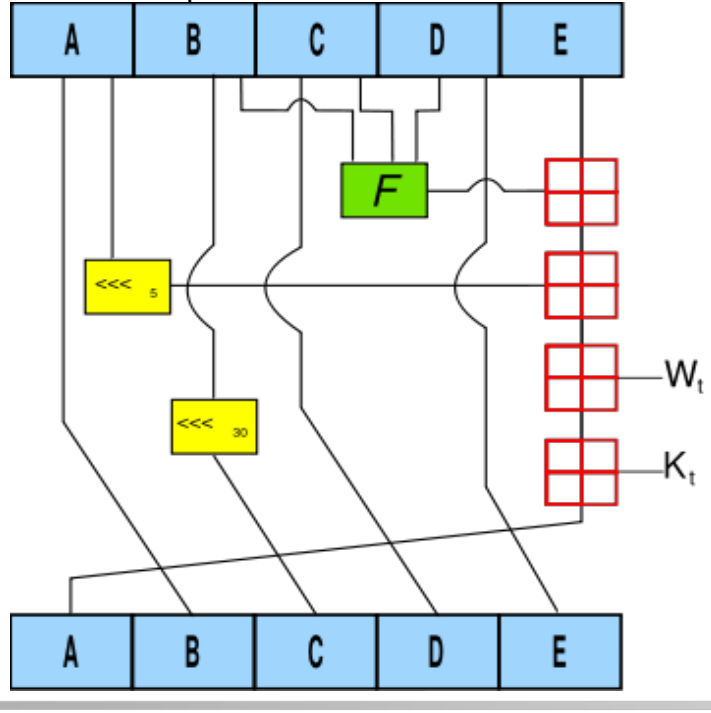
- DES-CBC: 64-Bit Hashwert, der letzte Block dient als Hash

# Beispiel: SHA-1

**Beispiel** für dedizierte Hashfunktion SHA-1 (Secure Hash-Algorithm)

- MD4-basiert, Eingabestrings: max $2^{64}$ Bit, Padding
- Verarbeitung von 512 Bit Blöcken, Hashwert von 160bit, A-E feste Startwerte
- Jeder Block:
  - Verarbeitung in 4 Stufen mit jeweils 20 Runden
  - Verarbeitung von auf 80-Bit erweiterten 32-Bit Worten $W_i$ in den 80 Runden, siehe Bild
  - $<<<_n$ Linksshift um n-Bits
- Effizient in Software implementierbar:
  AND, OR, XOR, Komplement und Shifts.

| Stage t | Round j | Constant $K_t$ | Function $f_t$ |
|---|---|---|---|
| 1 | 00…19 | K=5A827999 | f(B,C,D)=(B∧C)∨(¬B∧D) |
| 2 | 20…39 | K=6ED9EBA1 | f(B,C,D)=B⊕C⊕D |
| 3 | 40…59 | K=8F1BBCDC | f(B,C,D)=(B⊕C)∨(B⊕D)∨(C⊕D) |
| 4 | 60…79 | K=CA62C1D6 | f(B,C,D)=B⊕C⊕D |



# Sicherheit dedizierter Hashfunktionen

**Sicherheit dedizierter Hashfunktionen:**

- Das Finden von Kollisionen bei MD4 und dessen Erweiterung MD5 ist möglich! MD5 gilt als unsicher!
- Kollisionen auf SHA-1 (160 Bit) wurden im Jahr 2005 signifikant schneller als Brute Force $2^{80}$ in $2^{69}$ herbeigeführt
- Umstieg auf z.B. SHA-2, mit SHA-256 Bit oder SHA-512 Bit wird empfohlen, SHA-2 Verfahren gelten derzeit noch als sicher
- SHA 3 wurde am 2. Oktober 2012 ausgewählt. (siehe Übung) Die Suche nach SHA 3 wurde bereits 2007 durch einen Wettbewerb der NIST gestartet. Aus den ursprünglich 51 Kandidaten wurde Keccak nach 5 jähriger Prüfung und Analyse (unter anderem durch die NSA) zum neuen Standard erklärt.

# Message-Authentication-Code

# 3.2 Message-Authentication-Code

## 3.2 Message-Authentication-Code (MAC)

- Kryptographische Checksummen, Keyed-Hash
- **Ziel:** Authentizität des Datenursprungs und Datenintegrität

Hashfunktion mit Schlüssel: MAC: $X^* \times EK \rightarrow X^n$

- geheimer (Pre-shared) Schlüssel $K_{AB}$ zwischen Partnern $A$,$B$
- $K_{AB}$ symmetrischer Schlüssel (kein Non-Repudiation)
- MAC-Wert wird an Nachricht angefügt
- Empfänger prüft Authentizität und Integrität mit $K_{AB}$

# Beispiel: Keyed SHA-1

Beispiel: Keyed SHA-1: $M' = M \mid K_{AB}$ berechnen von SHA-1($M'$)

- Beispiel verwendet Secret-Suffix für Schlüssel
- Auch $M' = K_{AB} \mid M$ ist möglich, Secret-Prefix
- **Aber:** Attacken auf beide Varianten sind möglich
- **Lösung:** HMAC-Verfahren

**HMAC-Verfahren RFC 2104**

- 1996: Bellare, Canetti und Krawczyk
- Idee: HMAC als ‚Wrapper' um existierende Verfahren,
- HMAC nutzt Schlüssel, um Initialisierungsvektor zu variieren
- 2-Hash-Anwendungen: innerer und äußerer Hash erhöht die Kollissionsresistenz

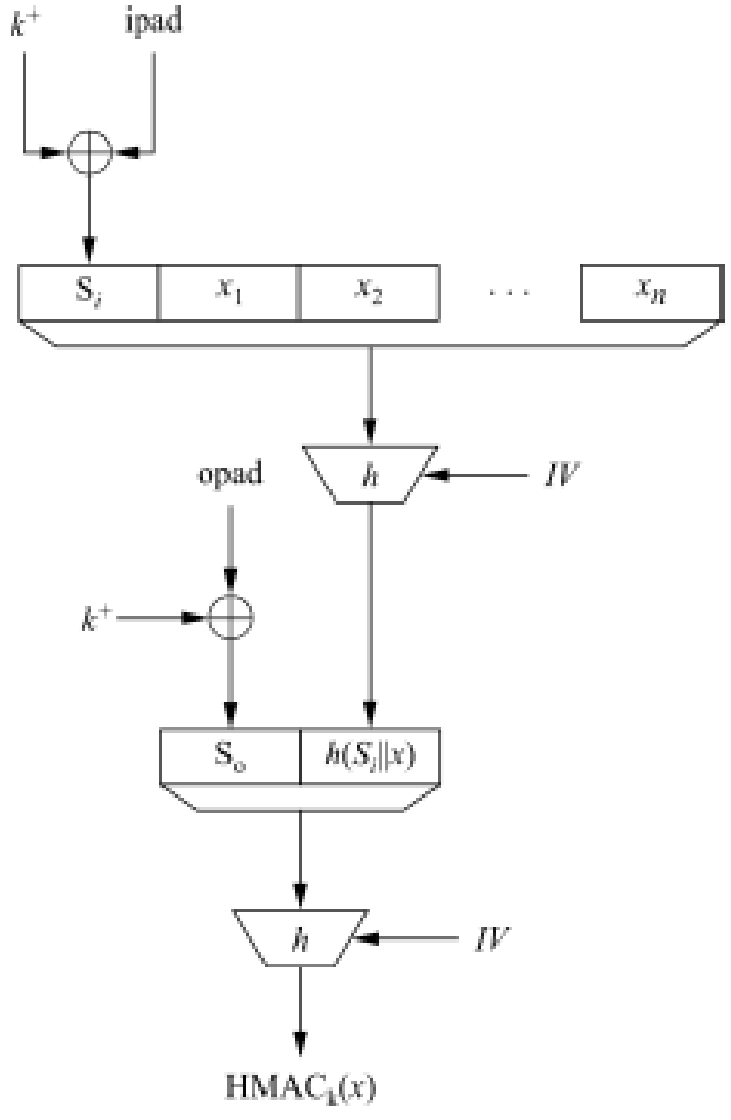
# HMAC

## HMAC: Allgemeine Arbeitsweise

- *h* ist hierbei eine beliebige kryptographische Hashfunktion, z.B. SHA
- *k* ist der gemeinsame MAC-Schlüssel
- x ist die Eingabe
- $k^+$ ist erweiterter Schlüssel k: mit 0 aufgefüllt

- *ipad* = 00110110,00110110, . . . , 00110110
- *opad* = 01011100,01011100, . . . , 01011100
- $HMAC_K(x) = h[(k^+ \text{ xor opad})||h[(k^+ \text{ xor ipad})||x]]$

  (|| ist die Konkatenation )




Konkatenation

# Digitale Signaturen

# 3.3 Digitale Signatur

## 3.3 Digitale Signatur

**Ziel:** Nachweis der Urheberschaft eines Dokuments

**Signaturverfahren:** analog zu Hashfunktionen

- Dedizierte Signaturverfahren**:** z.B. DSA, ECDSA (z.B. im nPA)
- Public Key Verschlüsselung**:** z.B. RSA (De-Facto Standard)

**Basis:** Schlüsselpaar ( $K_{sig}$ , $K_{veri}$ ) eines Public-Key-Verfahrens

- Privater Signaturschlüssel $K_{sig}$
- Öffentlicher Verifikationsschlüssel $K_{veri}$

**Vorgehen:** Signieren eines Klartextes M z.B. mit RSA

1. Hashen
2. Signieren
3. Prüfen

# Digital Signature Algorithm

**Digital Signature Algorithm DSA**

- Federal US Government Standard für digitale Signaturen
- DSA basiert auf dem ElGamal Signatur-Verfahren
- **Vorteil:** DSA-Signaturen sind nur 320 bits lang
- **Nachteil:** Signatur-Verifikation ist langsamer als bei RSA

**Schlüsselgenerierung bei DSA:**

1. Generiere Primzahl p mit $2^{1023} < p < 2^{1024}$
2. Finde Teiler $q$ von $p - 1$, mit $q$ ist Primzahl $I$ und $2^{159} < q < 2^{160}$
3. Finde Integer $\alpha$ mit $ord(\alpha) = q$
4. Wähle Random-Integer $d$ mit $0 < d < q$
5. Berechne $\beta \equiv \alpha^d \bmod p$

**Das erzeugte Schlüsselpaar** ( $K_{sig}$ , $K_{veri}$ ) **ist:**

$k_{pub} = (p, q, \alpha, \beta) = K_{veri}$

$k_{pr} = (d) = K_{sig}$

# Signatur-Erzeugung
# Signatur-Verifikation

- **Signatur-Erzeugung mit DSA**

Gegeben seien: Klartext *x*, Private Key *d* und Public Key *(p, q, α, β)*

1. Wähle Integer als Random Ephemeral Key $k_E$ mit $0 < k_E < q$
2. Berechne $r \equiv (\alpha^{kE} \bmod p) \bmod q$
3. Berechne $s \equiv (SHA-1(x) + d \cdot r)\, k_E^{-1} \bmod q$

Die Signatur ist gegeben durch: *(r, s)*

- **Signatur-Verifikation**

Gegeben seien: Klartext *x*, Signatur *(r, s)*, Public Key *(p, q, α, β)*

1. Berechne Hilfswert $w \equiv s^{-1} \bmod q$
2. Berechne Hilfswert $u_1 \equiv w \cdot SHA1(x) \bmod q$
3. Berechne Hilfswert $u_2 \equiv w \cdot r \bmod q$
4. Berechne $v \equiv (\alpha^{\mu 1} \cdot \beta^{u2} \bmod p) \bmod q$

Falls $v \equiv r \bmod q$ dann ist die Signatur gültig.

**Beispiel (Source: C. Paar)** Klartext x

| **Alice** | **Bob** |
|---|---|
| | **Key generation**:<br>1.choose p = 59 and q = 29<br>2.choose $\alpha = 3$<br>3.choose private key d = 7<br>4.$\beta = \alpha^{\beta} = 3^7 \equiv 4 \bmod 59$ |
| $(p, q, \alpha, \beta) = (59,29,3,4)$ ← | |
| | **Sign**:<br>Compute hash of message H(x) = 26<br>1.choose ephermal key $k_E = 10$<br>2.$r = (3^{10} \bmod 59) \equiv 20 \bmod 29$<br>3.$s = (26 + 7 \cdot 20) \cdot 3) \equiv 5 \bmod 29$ |
| $(x,(r,s)) = (x,20,5)$ ← | |
| **Verify**:<br>$w \equiv 5^{-1} \equiv 6 \bmod 29$<br>$u_1 \equiv 6 \cdot 26 \equiv 11 \bmod 29$<br>$u_2 \equiv 6 \cdot 20 \equiv 4 \bmod 29$<br>$v = (3^{11} \cdot 4^4 \bmod 59) \bmod 29 = 20$<br>$v \equiv r \bmod 29 \rightarrow$ valid signature | |

# Sicherheit von DSA

## Sicherheit von DSA

- basiert auf diskretem Logarithmus Problem für p, um den privaten Schlüssel d zu knacken:

$$d = \log_{\alpha} \beta \bmod p$$

- Es gilt:

  p muss mindestens 1024 bit groß sein, um Logarithmus-Berechnungs-Attacken abzuwehren
- NIST Empfehlungen für Primzahlen p,q

| p | q | hash output (min) | security levels |
|---|---|---|---|
| 1024 | 160 | 160 | 80 |
| 2048 | 224 | 224 | 112 |
| 3072 | 256 | 256 | 128 |

# Elliptic Curve Digital Signature Algorithm

## Elliptic Curve Digital Signature Algorithm (ECDSA)

- Basiert auf Elliptic Curve Cryptography (ECC)
- Bitlängen im Bereich von 160-256 bits bieten einen vergleichbaren Sicherheitslevel wie 1024-3072 bit RSA (das entspricht wiederum 80-128 bit bei symmetrischen Verfahren)
- Signature besteht aus 2 Punkten, d.h. 2-fache der Bitlänge

## Sicherheit von ECDSA

- basiert auf EC diskretem Log. Problem
- Komplexität der derzeit besten Analyse-Methode ist proportional zur Wurzel der Gruppengröße des DL Problems, d.h. √q
- Tabelle: Bit-Längen u. Sicherheitslevel

| q | Hash-Länge | Sicherheitslevel |
|---|---|---|
| 192 | 192 | 96 |
| 224 | 224 | 112 |
| 256 | 256 | 128 |
| 384 | 384 | 192 |
| 512 | 512 | 256 |

# Schlüsselvereinbarung

## 3.4 Schlüsselvereinbarung (Key Establishment)

Zwei Klassen von Ansätzen:

- Schlüsselaustausch,-verteilung (Key Distribution)
- Schlüsselabsprache (Key Agreement)

### 3.4.1 Schlüsselaustausch mit symmetrischen Verfahren

- Problem:
  - Schlüsselaustausch jeder mit jedem sehr aufwändig:

    $\frac{n*(n-1)}{2}$ Schlüssel
  - Jeder Partner muss n-1 Schlüssel speichern
  - Dynamik: neuer Partner muss mit jedem anderen einen Schlüssel austauschen!
- **Lösungs-Idee:** Schlüsselverteilungs-Center

# Lösungsansatz

**Lösungsansatz:** Key Distribution Center (KDC)

Häufig auch als TTP Trusted Third Party bezeichnet

- KDC besitzt Secret-Key (Masterkey) $K_A$ mit jedem Partner A (*n* Schlüssel)
- Dieser Schlüssel ist vorab, out-of-band bereits ausgetauscht
- Die Schlüssel sind idR. langlebig und dienen als Basis zum sicheren Austausch von kurzlebigen Schlüsseln $K_{A,B}$ zwischen zwei Partnern A und B (Session-Keys, Kommunikationsschlüssel)
- Sehr weit verbreitetes Protokoll: Kerberos (kommt später genauer)

# Probleme mit KDC

**Probleme bei Ansätzen mit einem KDC:**

- **Keine Perfect Forward Secrecy**: Falls der Masterkey $K_A$ kompromittiert ist, kann Angreifer alle vorherigen Ciphertexte entschlüsseln
- **Single point of failure**: Der KDC speichert alle Masterkeys $K_A$, Zugriff auf diese DB eröffnet Zugriff auf alle Kryptotexte.
- **Communication bottleneck**:
  - Der KDC ist in jede Kommunikation der Partner eingebunden;
  - durch lange Lebenszeiten von Kommunikationsschlüsseln (Session Keys) kann das abgemildert werden, **aber …**

## 3.4.2 Schlüsselaustausch mit asymmetrischen Verfahren

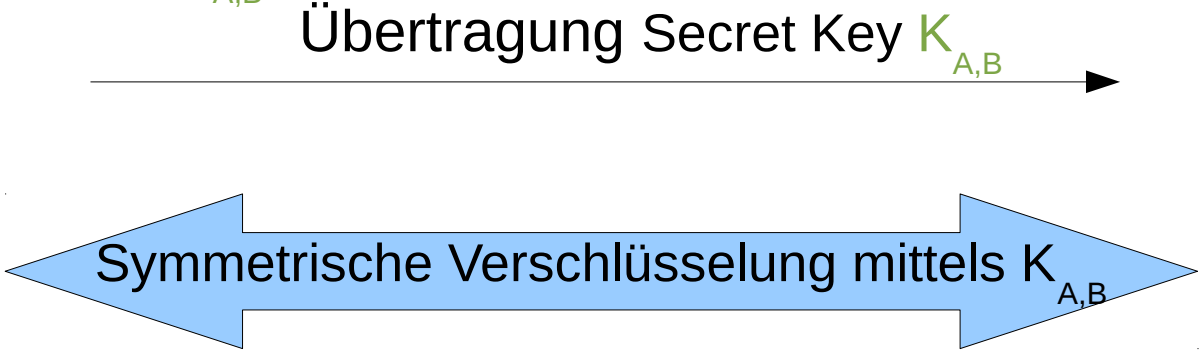
### Hybrid-Verfahren

- Symmetrischer Schlüssel $K_{A,B}$ soll ausgetauscht werden
- Verwendung eines Public-Key Verfahrens, z.B. RSA

**Ablauf**: obdA: Alice initiiert den Austausch mit Bob

**Alice** **Bob**

- Generiert Secret Key $K_{A,B}$



# 3.4.2 Schlüssel-Agreement

## 3.4.3 Schlüssel-Agreement: Diffie-Hellman-Verfahren (DH)

- 1976 von Whitfield Diffie und Martin Hellman vorgeschlagen
- Sehr weit verbreitet: u.a. Secure Shell (SSH), TLS, IPSec
- Basiert auf dem diskreten Logarithmus Problem
- DH basiert darauf, dass:
  - Exponentation in $Z_q^*$ mit $q$ Prim ist eine Einweg-Funktion und
  - die Exponentation kommutativ ist: $k = (a^y)^x \equiv (a^x)^y \bmod q$

**Funktionsweise des DH-Verfahrens**

(1)Wähle große Primzahl $q$ (allen Teilnehmern bekannt);

(2)wähle allen bekannten Wert $\alpha$, der eine primitive Wurzel von $q$ in der zyklischen Gruppe $\mathbf{Z}_q^*$ der primen Reste modulo $q$ ist.

$\alpha \in \mathbf{Z}_q^*$ und $\{1,..., q - 1\} = \{ \alpha^1,..., \alpha^{q-1}\}$

| Schritte | Teilnehmer A | Teilnehmer B |
|---|---|---|
| Wähle geheimen Schlüssel | $K_{priv\_A} \in \{2,...,q-2\}$ | $K_{priv\_B} \in \{2,...,q-2\}$ |
| Berechne öffentlichen Schlüssel und | $K_{pub\_A} = (\alpha^{K_{priv\_A}}) \bmod q$ | $K_{pub\_B} = (\alpha^{K_{priv\_B}}) \bmod q$ |
| tausche ihn aus | | |
| Berechne **gemeinsamen** Schlüssel $K_{AB}$ | $K_{AB} = (K_{pub\_B}^{K_{priv\_A}}) \bmod q$ | $K_{AB} = (K_{pub\_A}^{K_{priv\_B}}) \bmod q$ |
| **Angreifer**: Welche Kentnisse? | Aufwand $K_{AB}$ Problem: | zu berechnen? Man-In-The-Middle |

# Diffie-Hellman auf elliptischen Kurven

**Diffie-Hellman auf elliptischen Kurven (ECDH)**

Bezeichne $(d_A, Q_A)$ und $(d_B, Q_B)$ die Schlüsselpaare von Alice und Bob mit

$Q_A = d_A G$ und $Q_B = d_B G$.

- Alice → Bob: $Q_A$ Bob → Alice: $Q_B$
- Alice: $(x_k, y_k) = d_A Q_B$ Bob: $(x_k, y_k) = d_B Q_A$
- Es gilt: $(x_k, y_k) = d_B Q_A = d_B(d_A G) = d_A(d_B G) = d_A Q_B$

Beide verwenden $x_k = K_{AB}$ als gemeinsamen Schlüssel

# Zertifikate / digitale Signatur

# 3.5 Zertifikate

**Aufgabe**: Authentizitätsnachweis für öffentliche Schlüssel

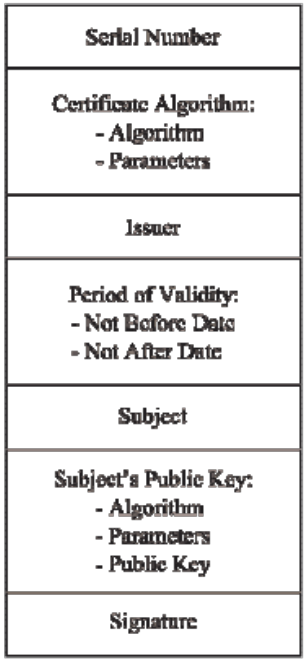
- **Genereller Lösungsansatz:**
  - Digitale Signatur über Public Key$_A$ und ID von A
  - Zertifikat = Public Key$_A$ + ID(A) + sig(Public Key$_A$ + ID(A))
  - Zertifikat bindet Identität A an den Public Key$_A$
- Standard für Zertifikate: X.509.v3 Format (RFC 2459)

| **Inhalt** | **Erläuterung** |
|---|---|
| Versionsnummer | beschreibt verwendetes Zertifikatformat |
| Seriennummer | eindeutiger Identifikator |
| Signatur | verwendete Algorithmen und Parameter |
| Zertifikatsaussteller | Name der ausstellenden Instanz |
| Gültigkeitsdauer | Angabe eines Zeitintervalls |
| Benutzername | eindeutiger Name des Benutzers |
| Schlüsselinformationen | Schlüssel des Benutzers und Algorithmen |
| Erweiterungen | in Version v2, v3 |
| …. | … |

# X.509.v3 Format

## X.509.v3 Format

- bestätigt die Gültigkeit des Public Key$_A$
- Public Key$_A$ ist digital signiert durch vertrauenswürdige Instanz, die CA: Certificate Authority



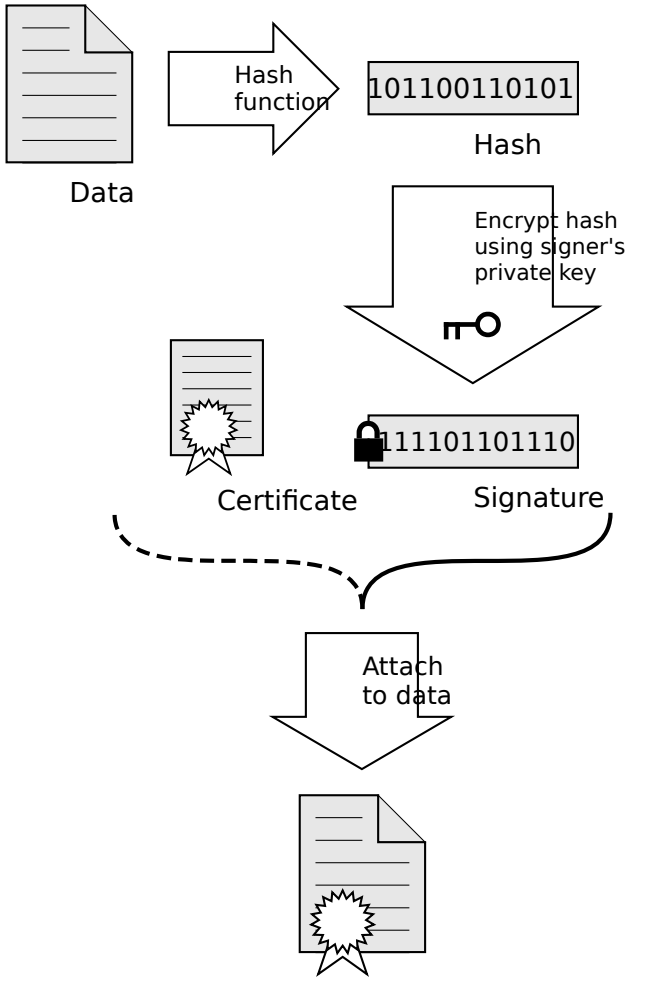

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=XY, ST=Austria, L=Graz, O=TrustMe Ltd, OU=Certif:
Authority, CN=CA/Email=ca@trustme.dom
        Validity
            Not Before: Oct 29 17:39:10 2000 GMT
            Not After : Oct 29 17:39:10 2001 GMT
        Subject: C=DE, ST=Austria, L=Vienna, O=Home, OU=Web Lab,
CN=anywhere.com/Email=xyz@anywhere.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c4:40:4c:6e:14:1b:61:36:84:24:b2:61:c0:b5:
                    d7:e4:7a:a5:4b:94:ef:d9:5e:43:7f:c1:64:80:fd:
                    9f:50:41:6b:70:73:80:48:90:f3:58:bf:f0:4c:b9:
                    90:32:81:59:18:16:3f:19:f4:5f:11:68:36:85:f6:
                    1c:a9:af:fa:a9:a8:7b:44:85:79:b5:f1:20:d3:25:
                    7d:1c:de:68:15:0c:b6:bc:59:46:0a:d8:99:4e:07:
                    50:0a:5d:83:61:d4:db:c9:7d:c3:2e:eb:0a:8f:62:
                    8f:7e:00:e1:37:67:3f:36:d5:04:38:44:44:77:e9:
                    f0:b4:95:f5:f9:34:9f:f8:43
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                email:xyz@anywhere.com
            Netscape Comment:
                mod_ssl generated test server certificate
            Netscape Cert Type:
                SSL Server
    Signature Algorithm: md5WithRSAEncryption
        12:ed:f7:b3:5e:a0:93:3f:a0:1d:60:cb:47:19:7d:15:59:9b:
        3b:2c:a8:a3:6a:03:43:d0:85:d3:86:86:2f:e3:aa:79:39:e7:
        82:20:ed:f4:11:85:a3:41:5e:5c:8d:36:a2:71:b6:6a:08:f9:
        cc:1e:da:c4:78:05:75:8f:9b:10:f0:15:f0:9e:67:a0:4e:a1:
        4d:3f:16:4c:9b:19:56:6a:f2:af:89:54:52:4a:06:34:42:0d:
        d5:40:25:6b:b0:c0:a2:03:18:cd:d1:07:20:b6:e5:c5:1e:21:
        44:e7:c5:09:d2:d5:94:9d:6c:13:07:2f:3b:7c:4c:64:90:bf:
        ff:8e
```

**Probleme: u.a.**

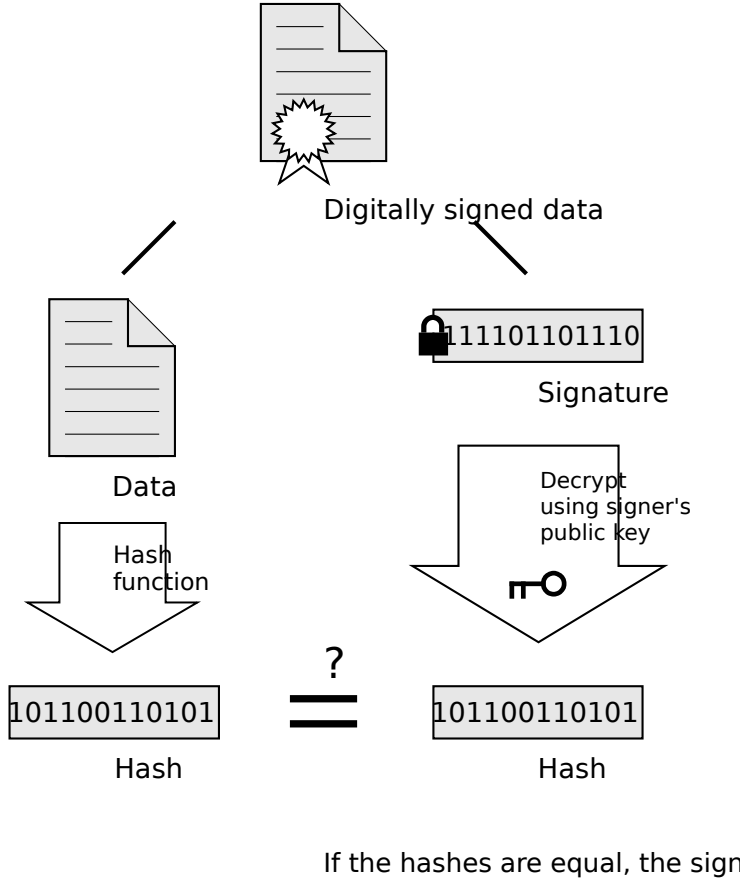
- Aussagekraft des Zertifikats
- Vertrauen in den Aussteller (CA)

**Lösung:** u.a. PKI, was sonst bekannt?

## Signing




## Verification



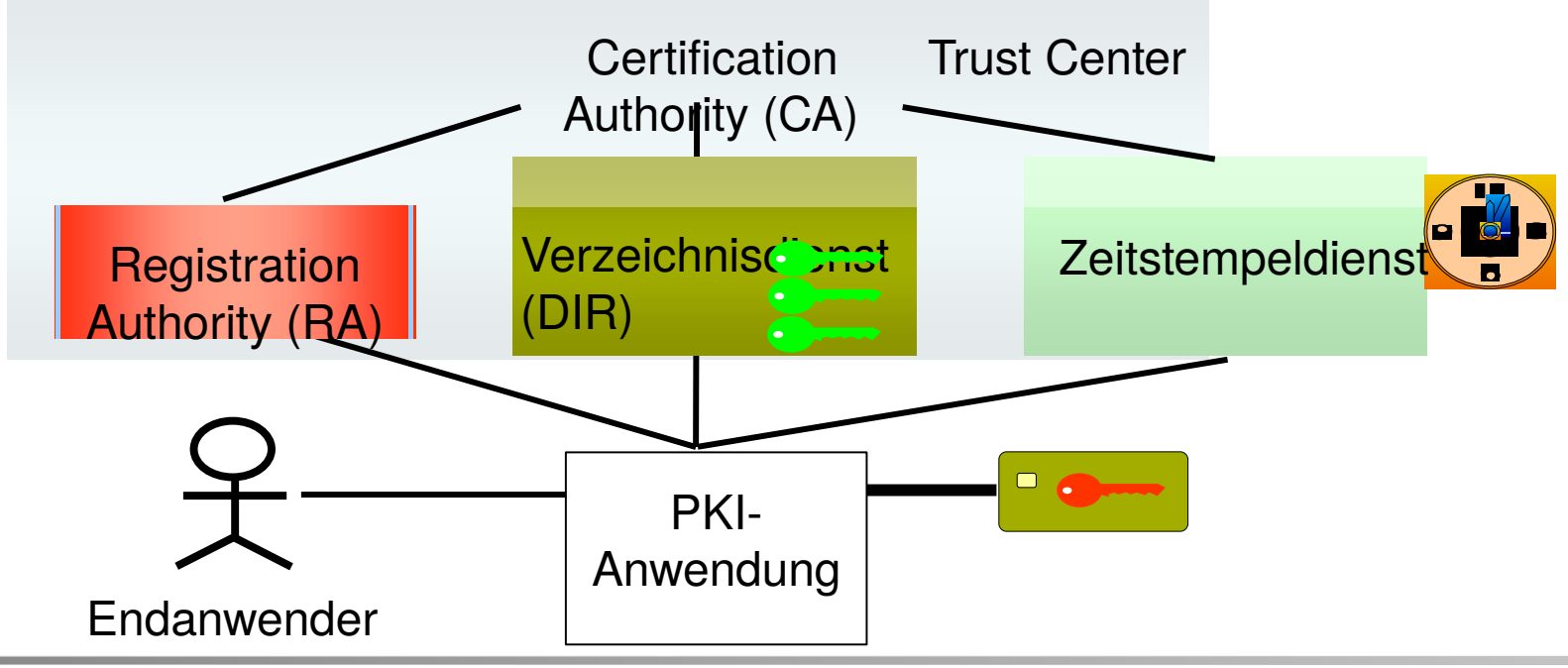

If the hashes are equal, the sign

## 3.6 Public Key Infrastructure PKI

**Aufgabe:**

- Infrastruktur, um Zertifikate auszustellen und
- Zertifikate durch Dritte prüfen lassen zu können

**Komponenten einer PKI**



# Komponenten einer PKI

**Komponenten einer PKI**: Erläuterung

- Certification Authority (CA):
  - Stellt Zertifikate aus, signiert und veröffentlicht sie
  - Erstellt und veröffentlicht Listen von ungültigen Zertifikaten (CRLs), Certificate Revocation List
  - bietet Online Certificate Status Protocol (OCSP) für Clients
- Registration Authority (RA):
  - bürgt für die Verbindung zw. öffentlichem Schlüssel und Identitäten/Attributen der Zertifikatsinhaber
- Verzeichnisdienst: Verteilung der Zertifikate und CRLs
- Zeitstempeldienst: signierte Zeitstempel (Gültigkeitsdauern, …)
- Personalisierung: Übertragung von Schlüssel und Zertifikaten

## Hierarchie von CAs

- Higher-level CAs stellen Zertifikate für untergeordnete CAs aus
- jeder vertraut der top-level CA, der Root-CA (z.B. Bundesnetzagentur)
- Validierung eines Zertifikats: Aufbau eines Zertifizierungspfades

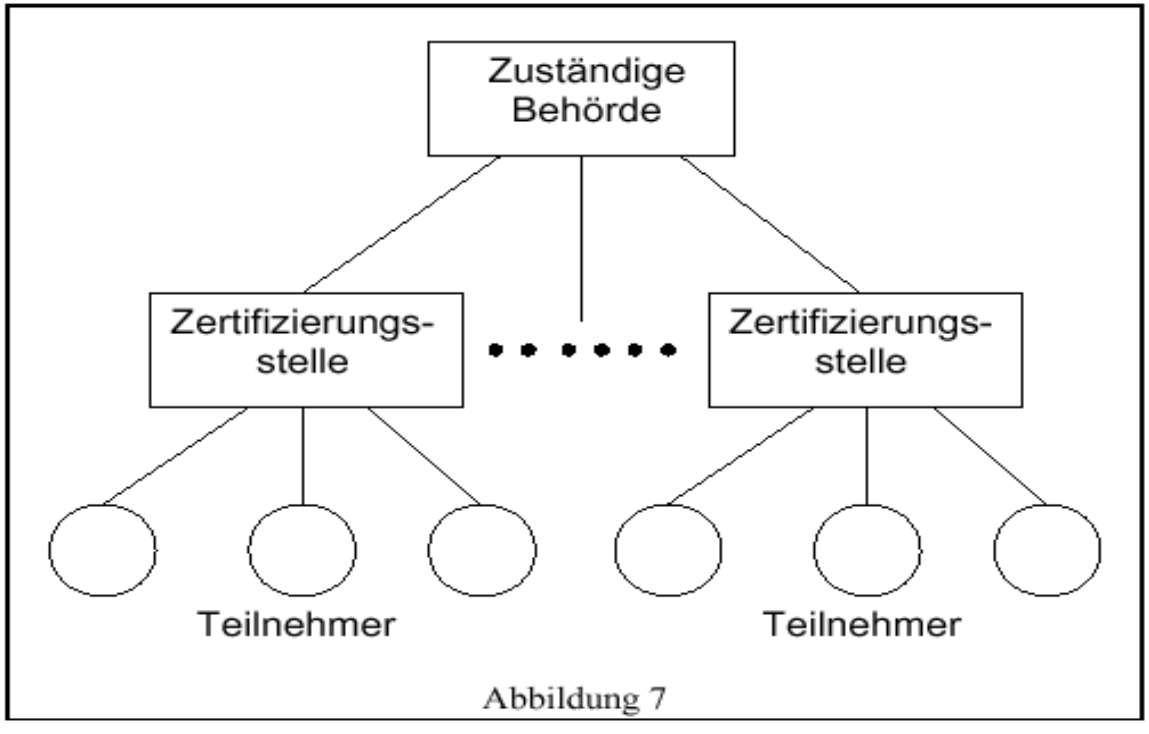


Abbildung 7

Hierarchie von CAs

Nächste Woche
Zugriffsschutz