

# Themen zur Computersicherheit

## Teilnehmerauthentisierung

PD Dr. Reinhard Bündgen  
bueundgen@de.ibm.com

# Authentisierung vs. Authentifizierung

- Authentisierung
  - „aktiv“
  - Subjekt authentisiert sich: beweist seine Identität
- Authentifizierung
  - „passiv“
  - Subjekt authentifiziert Objekt: erlangt Beweis über Identität des Objekts

# Ziel der Authentisierung

- Ein Teilnehmer, der Zugang zu einem System anfordert muss seine Identität nachweisen
- Teilnehmer wird im System durch Teilnehmer-Identifikator (Mitgliedsnummer, Kontonummer, e-mail Adresse o.ä.) beschrieben
  - evtl. mit Attributen (wie Adresse, Geburtsdatum, Kreditkartennummer,...) versehen
- Die Verifikation, dass die Teilnehmerattribute wirklich zum Teilnehmer gehören
  - außerhalb des Systems, z.B. persönliche Anmeldung, Postident-Verfahren
  - Plausibilitätschecks: Rückfrage an e-mail Adresse, Konsistenz von Kreditkartendaten
  - elektronische Signaturen (ePA)

# Richtung der Authentisierung

- Teilnehmer beweist dem System seine Identität
  - z.B login
- System beweist dem Teilnehmer seine Identität
  - z.B. die meisten HTTPS Server
- einseitige Authentisierung
  - z.B klassisches login
- beidseitige Authentisierung
  - z.B.ssh

# Methoden der Authentisierung

- Besitz
  - z.B. Ausweis, Smartcard, Handy
  - → mTAN
- Wissen
  - Passwort, PIN, TAN-Liste
- (körperliche) Merkmale
  - Irismuster, Fingerabdruck, Retina, Handschrift, Stimme, Handgeometrie, DNA?,
- Lokation
  - GPS Daten, Konsole, Netzwerksegment
- Einfaktor-Authentisierung
- Mehrfaktor-Authentisierung
- Vier-Augenprinzip

# Passwortverfahren - Anforderungen

- Passwort: auch Kennwort, Passphrase, PIN
- Anforderungen
  - nur Teilnehmer darf Passwort kennen
  - System muss Passwort erkennen können
  - System muss unterschiedliche Passwörter unterscheiden können
- abgeleitete Anforderungen
  - Passwort darf nur verschlüsselt gespeichert werden
  - Passwort darf nur verschlüsselt übertragen werden

# Kryptographische Hashfunktionen

im folgenden gilt  $\Sigma = \{0,1\}$

**Definition** Eine Funktion  $h: \Sigma^* \rightarrow \Sigma^n$  heißt *Hashfunktion* wenn es einen effizienten Algorithmus gibt, der für jedes  $m \in \Sigma^*$  den Wert  $h(m)$  berechnet.

**Definition** Eine Hashfunktion  $h$  heißt *Einweg-Hashfunktion*, wenn zu zufällig gewähltem  $y \in \Sigma^n$  der Wert  $h^{-1}(y)$  nicht effizient berechnet werden kann.

**Definition** Eine Hashfunktion  $h$  heißt *schwach kollisionsresistent* (*target collision resistant*, *2<sup>nd</sup> pre-image resistant*) wenn es keinen effizienten Algorithmus gibt, der zu gegebenen  $m \in \Sigma^*$  ein  $m' \in \Sigma^*$  findet mit  $m \neq m'$  und  $h(m) = h(m')$ .

**Definition** Eine Hashfunktion  $h$  heißt *stark kollisionsresistent* wenn es keinen effizienten Algorithmus gibt der  $m \in \Sigma^*$  und  $m' \in \Sigma^*$  findet mit  $m \neq m'$  und  $h(m) = h(m')$ .

**Definition** Eine stark kollisionsresistente Einweg-Hashfunktion heißt *kryptographische Hashfunktion*.

# Zum Nachdenken

- Ist jede schwach kollisionsresistente Hashfunktion stark kollisionsresistent?
  - Wenn ja, warum?
- Beschreibe ein nicht kollisionsresistente Einweg-Hashfunktion

# Ideale kryptographische Hashfunktionen

- Charakterisierung (Ferguson et al): Eine ideale Hashfunktion verhält sich wie eine Zufallsabbildung.
- Eine Attacke auf eine Hashfunktion: eine nicht generische Methode, die einen Unterschied zu einer idealen Hashfunktion findet.
  - nicht-generisch: nutzt spezifische Eigenschaft der zu attackierenden Hashfunktion

# Das Geburtstagsparadox

- **Satz** Zu beliebiger Hashfunktion  $h: \Sigma^* \rightarrow \Sigma^n$  gibt es einen Algorithmus, der zur Berechnung von  $h^{-1}(y)$  für ein zufällig gewähltes  $y \in \Sigma^n$  im Durchschnitt  $2^{n-1}$  Hashoperationen benötigt
- **Satz (Geburtstagsparadoxon)** Zu beliebiger Hashfunktion  $h: \Sigma^* \rightarrow \Sigma^n$  ist die Wahrscheinlichkeit  $P(n,k)$ , dass eine Menge  $M \subseteq \Sigma^*$  mit  $|M| = k \geq 1,2 \cdot 2^{n/2}$  zwei Elemente  $m, m' \in \Sigma^*$  mit  $m \neq m'$  und  $h(m) = h(m')$  enthält, größer als 0,5.

- **Beweis:**

$$1 - P(n, k) = \prod_{i=0}^{k-1} \frac{2^n - i}{2^n} \leq \prod_{i=0}^{k-1} e^{-\frac{i}{2^n}} \leq e^{-\frac{(k-1)^2}{2 \cdot 2^n}} \leq e^{-\frac{(k-1)^2}{2 \cdot 2^n}}$$

$$1 - P(n, 1,2 \cdot 2^{n/2} + 1) \leq e^{-\frac{2 \cdot 2^n \cdot \ln 2}{2 \cdot 2^n}} = e^{-\ln 2} = 1/2$$

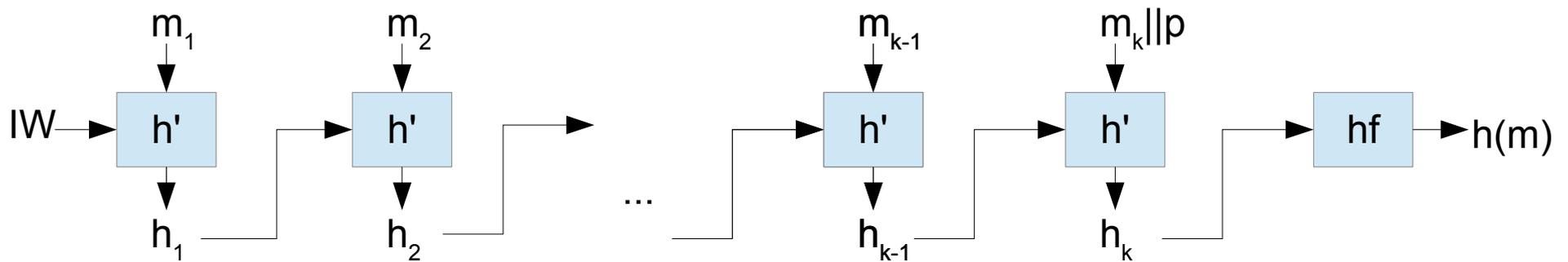
- **Geburtstagsangriff:**

- Berechne die Hashwerte von  $1,2 \cdot 2^{n/2}$  zufällig gewählten Elementen aus  $\Sigma^*$ .

# Meet-in-the-middle Angriff

- Sei die Hashfunktion  $h$  auf dem eingeschränkten Definitionsbereich  $\Sigma^n$  bijektiv
- $D \subseteq \Sigma^n$  mit  $|D| = 2^d$
- $h|_D : D \rightarrow \Sigma^n$  eine vorberechnete Tabelle von Hashwerten
- dann ist im Durchschnitt jeder  $2^{n-d}$ -te Hashwert in  $D$  enthalten
- Angriff
  - wähle  $D$  mit  $|D| = 2^{n/2}$
  - Berechne  $h|_D(m)$  für alle  $m \in D$ :  $2^{n/2}$  Operationen
  - sortiere  $h(D)$ :  $O(n/2 \cdot 2^{n/2})$  Operationen
  - wähle zufällige Menge  $E \subseteq \Sigma^n \setminus D$  mit  $|E| = 2^{n/2}$
  - Berechne  $h(m)$  für alle  $m \in E$ :  $2^{n/2}$  Operationen
  - suche  $h(m)$  in  $h(D)$ : je  $O(n/2)$  Operationen
  - Kollision wird mit Wahrscheinlichkeit  $\geq 0,5$  gefunden
  - Aufwand:  $O(n \cdot 2^{n/2})$

# Typische Struktur von Hashfunktionen



- Hashfunktion  $h: \Sigma^* \rightarrow \Sigma^n$  wird iterativ mit elementarer Hashfunktion  $h': \Sigma^k \times \Sigma^b \rightarrow \Sigma^k$  berechnet mit  $k \geq n$
- $hf: \Sigma^k \rightarrow \Sigma^n$  ist eine finale Filterfunktion
- Teile Nachricht  $m$  in Blöcke der Länge  $b$  auf:
  - $m = m_1 || m_2 || \dots || m_{k-1} || m_k$  mit  $|m_i| = b$  für  $i < k$  und  $|m_k| \leq b$
- sei  $p \in \Sigma^*$  mit  $|p| = b - |m_k|$  ein Padding
- sei  $IW \in \Sigma^k$  ein fester Initialisierungswert
- $h_0 = IW$
- $h_i = h'(h_{i-1}, m_i)$  für  $1 \leq i \leq k-1$
- $h_k = h'(h_{k-1}, m_k || p)$
- $h(m) = hf(h_k)$

**Notation:**  
|| ist der Konkatenationsoperator

# Einige wichtige Hashfunktionen

Name	Autor	Jahr	Blockgröße	Hashgrößen (bit)	Aufwand Kollision
MD4	Rivest	1990	512	128	1995: $2^{20}$ (heute 2?)
MD5	Rivest	1992	512	128	2009: $2^{16}$
SHA-1	NSA	1995	512	160	2007: $2^{61}$
SHA-2	NIST	2002	512, 1024	224, 256, 384, 512 <sup>1)</sup>	
SHA-3 (Keccak)	Bertoni et al	2011	<1600	224 – 512	

Empfohlene sichere Hashlängen ([www.keylength.com](http://www.keylength.com))  
je nach Autor/Organisation

- für 2015: 156 – 224 bits (BSI 224 bits)
- für 2020: 163 – 256 bits (BSI 256 bits)
- für 2030: 176 – 256 bits
- für 2050: 203 – 512 bits

<sup>1)</sup> Hashgrößen passen zu 2DES, AES-128, AES-192, AES-256 Chiffren

# Ein Maß für die Sicherheit

- die Anzahl der kryptographischen Operationen, die (im Durchschnitt) durchprobiert werden muss um eine kryptographische Funktion zu attackieren.
- Da diese Zahl sehr groß ist nimmt ihren 2er-Logarithmus und nennt ihn *Sicherheitsniveau*.
- Die Masseinheit für das Sicherheitsniveau: Bits
- Heute wird meist ein Sicherheitsniveau von 128 Bits empfohlen.
- Das Sicherheitsniveau einer idealen Hashfunktion mit einem Hashwert der Länge  $n$  ist  $n/2$ .

# Große Zahlen und Zeit

- 1 Zyklus auf 4GHz Rechner:  $0.25 \cdot 10^{-9}\text{s} \approx 2^{-32}\text{s}$
- 1h = 3 600s <  $2^{12}\text{s} \approx 2^{44}$  Zyklen
- 1d = 86 400s <  $2^{17}\text{s} \approx 2^{49}$  Zyklen
- 1y = 31 536 000s  $\approx 2^{25}\text{s} \approx 2^{57}$  Zyklen

# Passwortverfahren - Methoden

- System hat Passwortdatenbank  $pdb: T \rightarrow EP$  die jedem autorisierten Teilnehmer  $t \in T$  ein verschlüsseltes Passwort in  $EP$  zuweist.
- Passwort  $p_t \in P$  des Teilnehmers  $t$  wird mit Passwortverschlüsselungsfunktion  $ep: P \rightarrow EP$  verschlüsselt und mit hinterlegten verschlüsselten verglichen:  $ep(p_t) = pdb(t)$ ?
- Folgerung
  - $ep^{-1}: EP \rightarrow P$  mit  $ep^{-1}(ep(p)) = p$  muss *praktisch nicht berechenbar* (also sehr aufwendig) sein  $\rightarrow$  *Einwegfunktion*
  - $ep$  muss jedoch mit vertretbarem Aufwand berechenbar sein

# Passworthasfunktionen

- bekannte Funktion gemäß Kerkhoffs Prinzip
- Passwortverschlüsselung: kryptographische Hashfunktion
- dasselbe Passwort muss auf unterschiedlichen Systemen unterschiedliche Hashes ergeben: Salz (Pfeffer)
- sie muss schwer zu knacken sein: einstellbarer Kostenfaktor mehrfaches Iterieren einer zugrundeliegenden Hashfunktion
- z.B. PBKDF2 aus PKCS#5 v2 oder bcrypt

# PBKDF2

- Password based key derivation function 2
- PBKDF2:  $\Sigma^* \times \Sigma^* \times \mathbf{N} \times \mathbf{N} \rightarrow \Sigma^*$  berechnet Passworthash (bzw den Schlüssel)
- $F: \Sigma^* \times \Sigma^* \times \mathbf{N} \times \mathbf{N} \rightarrow \Sigma^n$  berechnet Schlüsselteil
- PRF:  $\Sigma^* \times \Sigma^* \rightarrow \Sigma^n$  berechnet schlüsselabhängigen Hashwert einer Nachricht
- $\text{PBKDF2}(\text{pw}, \text{salt}, \text{iter}, \text{keylen}) = T_1 || \dots || T_{m-1} || \text{msb}(T_m, r)$
- mit
  - $T_j = F(\text{pw}, \text{salt}, \text{iter}, j)$
  - $m = \lceil \text{keylen}/n \rceil$  ;  $r = \text{keylen} - (m - 1) \cdot n$
  - $F(p, s, it, j) = U_1 \oplus \dots \oplus U_{it}$  mit
    - $U_1 = \text{PRF}(\text{pw}, \text{salt} || \text{INT}(j))$
    - $U_i = \text{PRF}(\text{pw}, U_{i-1})$

**Notation:**

$\mathbf{N}$  sind die natürlichen Zahlen  
 $\oplus$  ist der bitweise xor Operator  
 $\text{msb}(m, r)$  r signifikanteste Bits von m

# Wie werden Passwörter geknackt (1)

- online Angriff
  - Passwort raten + Versuch einzuloggen
- offline Angriff
  - Gegeben ein verschlüsseltes Passwort suche Urbild zu bekannter Passworthashfunktion
  - Gegeben eine Liste verschlüsselter Passwörter suche Urbild zu einem der verschlüsselten Passwörter

# Gegenmaßnahmen

- online Angriff:
  - erlaube nur endliche Zahl von Fehlversuchen
  - forciere nach einem Fehlversuch eine Pause, Pausen werden mit jedem Fehlversuch größer
  - jailing: vorspielen eines gelungenen Einbruchs
  - CAPTCHAs (Completely Automated Public Turing Test to tell Computers and Humans Apart): verhindern automatischer Angriffe
- offline Angriff
  - Passworthashes (/etc/shadow) dürfen nur autorisierten Lesern zugänglich sein
  - langsame Passworthashfunktionen
  - (systemspezifisch) pfeffern
  - (nutzerspezifisch) salzen

# Wie werden Passwörter geknackt (2)

- Katalog von vorberechneten Hash-Urbildern (rainbow tables)
- Passwortkataloge
  - beliebte Passwörter (z.B. 12345678, letmein, asdfghj)
  - Wörterbücher verschiedener Sprachen
  - aussprechbare Buchstabenketten
  - jemals geknackte Passwörter
- Variationen von Katalogelementen
  - Prefixe, Suffixe,
  - Ersetzung Buchstaben durch Ziffern, Sonderzeichen (z.B. i → 1, s → \$, at → @, for → 4)
  - Transformationen: vorwärts, rückwärts
- Indexierung von Festplatten
- Nutzung persönlicher Informationen
  - Phishing

# Rainbow Tables (1)

- Ziel: erstelle vorberechnete Tabelle von Passwörtern und Passworthashes
- Problem: Suchraum zu groß,
  - z.B. alle alphanumerische Passwörter der Länge 8: ca 1 PByte
- Lösung: kompakte Darstellung der Vorbereitung.
  - Speichere weniger vorberechnete Werte
  - auf Kosten teurerer Nachschlagefunktionen
  - Hashketten:  $\text{StartPW} \rightarrow h_1 \rightarrow h_2 \rightarrow \dots \rightarrow h_n$
  - speichere pro Hashkette nur StartPW und Kettenendwert  $h_n$
  - $h_1, \dots, h_{n-1}$  müssen jeweils berechnet werden
  - Passwortsuche für gegebenen Hashwert:
    - berechne Hashketten bis gespeicherter Kettenendwert gefunden,
    - Dann verfolge Hashkette ab gespeichertem StartPW

# Rainbow Tables (2)

- Passwort Hashfunktion  $ep: P \rightarrow EP$
- Reduktionsfunktion  $r: EP \rightarrow P$
- einfache Hashkette:
  - $p_0 \xrightarrow{ep} h_0 \xrightarrow{r} p_1 \xrightarrow{ep} h_1 \xrightarrow{r} p_2 \xrightarrow{ep} \dots \xrightarrow{r} p_n$
  - erlaubt Kollisionen: mehrere Hashketten haben das gleiche Ende
- rainbow table Hashketten
  - unterschiedliche Reduktionsfunktionen  $r_i: EP \rightarrow P$  für  $i \in \{0, \dots, n\}$
  - $p_0 \xrightarrow{ep} h_0 \xrightarrow{r_0} p_1 \xrightarrow{ep} h_1 \xrightarrow{r_1} p_2 \xrightarrow{ep} \dots \xrightarrow{r_n} p_n$
  - Suche Passwort zu Passworthash  $h$ :
    - für  $i$  von 0 bis  $n$ :
      - berechne Teilhashkette ab Position  $i$  mit  $h_i = h$
      - falls Endwert der berechneten Teilhashkette gleich einem Endwert einer vorberechneten Teilhashkette ist: verfolge diese vorberechnete Kette und bestimme  $p_i$

# Zum Nachdenken

- Welche Angriffe werden von rainbow tables unterstützt?
- Welche Komponenten von PBKDF2 schützen vor dem Einsatz von rainbow tables?
- Welche Eigenschaften hat eine gute Reduktionsfunktion aus?

# Beispiele gefundener Passwörter

ilovemySister31  
windermere2313  
k1araj0hns0n  
gonefishing1125 Qbesancon321  
Apr!l221973 ilovetofunot allineedislove  
tmdmmj17 Sh1a-labe0uf  
iloveyousomuch all of the lights  
qeadzcwrsfxv1331 @Yourmom69  
DG091101%  
BandGeek2014 Philippians4:13  
Philippians4:6-7 i hate hackers

Schneier newsletter 03/2014: „Last year, Ars Technica gave three experts a 16,000-entry encrypted password file, and asked them to break as many as possible. The winner got 90% of them, the loser 62% -- in a few hours.“

# Gute Passwörter

- Großer Zeichensatz: Buchtaben (groß und klein), Zahlen Sonderzeichen
- echte Zufalls Wörter ( $62^8 > 10^{14}$ )
  - Kontospezifisch verziert
- Anfangsbuchstaben von *persönlichen* Phrasen
  - Schneier: "This little piggy went to market" -> "tlpWENT2m"
  - keine Shakespearezitate!
- Passwortwiederherstellungsfragen:
  - Antworten dürfen nicht unsicherer sein als Passwort → lügen!