

Hashfunktionen

Roman Brunnemann – Stephan Müller

23. November 2004



Einleitung

Sicherheitsanforderungen

Konstruktion von Hashfunktionen

Präsentation einer eigenen Hashfunktion

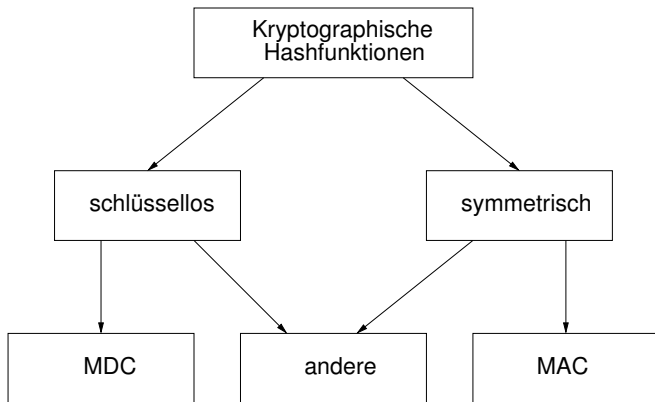
MD5

MD5 vs. SHA1

Angriffe auf Hashfunktionen



Einteilung



Motivation

- ▶ Auffinden von Daten in einer Datenbank
- ▶ Generierung von Schlüsseln
- ▶ Digitales Signieren von Nachrichten
- ▶ Erzeugen eines kompakten repräsentativen Abbilds einer Nachricht
- ▶ Feststellen von Manipulationen

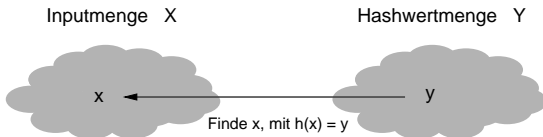
Anforderungen

- ▶ Allgemeine Anforderung
 - ▶ Datenreduktion
 - ▶ Zufälligkeit
 - ▶ Eindeutigkeit
 - ▶ Effizienz
- ▶ Kryptographische Anforderungen
 - ▶ Urbild Resistenz
 - ▶ 2. Urbild Resistenz
 - ▶ Kollisions Resistent

1. Urbild Problem

Es soll nicht möglich sein, zu einem existierenden Hashwert, ein Urbild zu finden.

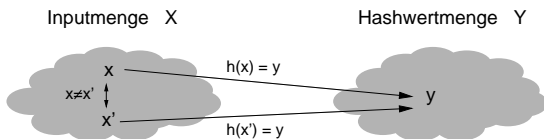
- ▶ **geg.:** $h : X \rightarrow Y$, Hashwert $y \in Y$
- ▶ **ges.:** $x \in X$, mit $h(x) = y$
- ▶ \Rightarrow Einweg-Hashfunktion



2. Urbild Problem

Es soll nicht möglich sein, zu einem gegebenen Urbild (Nachricht oder Text) mit zugehörigem Hashwert, ein zweites Urbild zu finden, dass den gleichen Hashwert hat.

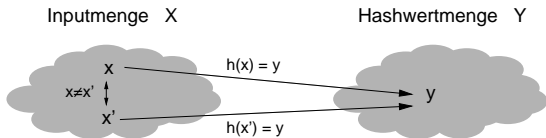
- ▶ **geg.:** $h : X \rightarrow Y$, Text $x \in X$
- ▶ **ges.:** $x' \in X$, mit $x' \neq x$ und $h(x') = h(x)$
- ▶ \Rightarrow 2.Urbild resistent



Kollisionsproblem

Es soll nicht möglich sein, zwei unterschiedliche Texte oder Nachrichten zu finden, die denselben Hashwert besitzen.

- ▶ **geg.:** $h : X \rightarrow Y$
- ▶ **ges.:** $x, x' \in X$, mit $x' \neq x$ und $h(x') = h(x)$
- ▶ \Rightarrow Kollisionsresistent



Problemreduktion

- ▶ Einfachstes der drei Probleme: Kollisionsproblem
- ▶ Reduktionsbeweis (*Cryptography - Theory and Practice* p.125-127)
- ▶ Mit anderen Worten:
Wenn ich eine kollisions-resistente Hashfunktion habe, erfülle ich auch die anderen beiden Kriterien

Zufallsorakelmodell (ZOM)

- ▶ Das Zufallsorakel stellt eine idealisierte Hashfunktion dar
- ▶ Wählt eine Funktion h aus der Menge aller Hashfunktionen aus
- ▶ Besitzt die folgenden Eigenschaften bei einer Anfrage mit dem Text x :
 - ▶ Die einzige Möglichkeit $h(x)$ zu bestimmen ist das Orakel zu befragen
 - ▶ Gab es schon eine Anfrage für x , so wird mit dem gleichen Hashwert wie bei der vorherigen Anfrage geantwortet.
 - ▶ Gab es noch keine Anfrage, so wird mit einem völlig zufälligen Hashwert geantwortet
 - ▶ Sei Y die Menge aller Hashwerte und $|Y| = M$, so beträgt die Wahrscheinlichkeit

$$\Pr[h(x) = y \mid \forall x \in X \text{ und } \forall y \in Y] = 1/M,$$
 für alle noch nicht angefragten x

Konstruktion von iterativen Hashfunktionen

Idee:

- ▶ Nachricht wird in Blöcke gleicher Länge zerlegt
- ▶ Schleife durchläuft alle Blöcke und manipuliert dabei den Hashwert

Konstruktion von iterativen Hashfunktionen erfolgt in 3 Schritten:

1. **Preprocessing**
2. **Processing**
3. **Optionale Output-Transformation**



Preprocessing

- ▶ Auffüllen der Nachricht auf ein Vielfaches der Blockgröße (*padding*)
- ▶ Zerlege die Nachricht in r Blöcke $(b_1, b_2, b_3, \dots, b_r)$, wobei t die Länge eines Blocks ist und $r \cdot t$ die Länge der gesamten Nachricht ist.

Processing

- ▶ Nehme den öffentlichen Startwert der Hashfunktion
- ▶ Startwert wird einmal für eine Hashfunktion festgelegt
- ▶ Länge des Startwertes ist die Länge des Hashwertes

$z_0 \leftarrow$ Startwert

$z_1 \leftarrow \text{compress}(z_0, b_1)$

$z_2 \leftarrow \text{compress}(z_1, b_2)$

⋮

$z_r \leftarrow \text{compress}(z_{r-1}, b_r)$



Optionale Output-Transformation

- ▶ Transformation am Ende der Blockschleife.
- ▶ Identität, Konkatenation

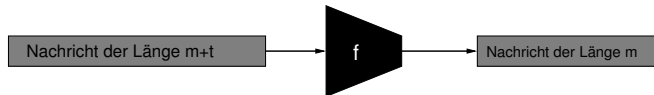
Merkle-Damgard Konstruktion

- ▶ Benutzt eine Komprimierungsfunktion (*compress*)
- ▶ *compress* braucht eine Eingabe fester Länge
- ▶ *compress* liefert eine Ausgabe fester Länge
- ▶ *compress* sollte kollisions-resistent sein
- ▶ **Bewiesen:** Wenn *compress* kollisions-resistent ist, dann ist auch die konstruierte Hashfunktion kollisions-resistent

Komprimierungsfunktion

Input

Output



$$f : 0, 1^{m+t} \rightarrow 0, 1^m \mid t \geq 2$$

f komprimiert immer um mindestens 2 Bit.

Input: Alter Zwischenhashwert + Block

Output: Neuer Zwischenhashwert

Merkle-Damgard Algorithmus

```

n ← |x|
k ← ⌈n/(t - 1)⌉
d ← n - (k - 1)(t - 1)
for i ← 1 to k - 1
    bi ← xi
bk ← xk || 0d
bk+1 ← Binärdarstellung von d
z1 ← 0m+1 || b1
g1 ← compress(z1)
for i ← 1 to k
    zi+1 ← gi || 1 || bi+1
    gi+1 ← compress(zi+1)
h(x) ← gk+1
return h(x)

```

n = Länge der Nachricht

x = Originalnachricht

t = Blockgröße

k = Anzahl der Blöcke

d = Aufzufüllende Länge

x_i = ein Block der

Originalnachricht

b_i = ein Kodierungsblock

m = Länge des Hashwertes

z_i = Temporäre Variable

g_i = Zwischenhash

compress :

$\{0, 1\}^{m+t} \rightarrow \{0, 1\}^m$

Eine eigene Hashfunktion

Sicherheitsaspekte

- ▶ *4bit* Hashsumme $\Rightarrow 2^4 = 16$ verschiedene Hashwerte

1. Aufteilung in 512-Bit Blöcke

Die Antwort lautet 42.

↓ Binäre Codierung

10101010111 ... 0100010100011

← b-Bits →

↓ Aufteilung in m 512-bit Blöcke (Padding)

10101 ... 11001 11101 ... 00101 ... 00111 ... 00011

← 512-bits →

← 512-bits →

← 512-bits →



2. Padding (Auffüllen des letzten Blocks)

Problem: Selten ist die Länge der binären Darstellung einer Nachricht ein Vielfaches von 512.

⇒ Auffüllen des Blocks nach folgenden Regeln:

1. Schreibe eine "1"
2. $k = |\text{Letzter Block}|$
 $d = 512 - 1 - k - 64$
 Schreibe d -mal eine "0"
3. $b = |\text{gesamte Nachricht}|$
 Schreibe die 64-Bit Darstellung von $b \bmod 2^{64}$



3. Definition von Konstanten (1)

Es werden 3 Felder definiert:

$$y[j] = \text{Ersten } 32 - \text{Bits von } |\sin(j + 1)| \text{ wobei } 0 \leq j \leq 63$$

$$z[0..15] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$$

$$z[16..31] = [1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12]$$

$$z[32..47] = [5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2]$$

$$z[48..63] = [0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9]$$

$$s[0..15] = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22]$$

$$s[16..32] = [5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20]$$

$$s[33..47] = [4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23]$$

$$s[48..63] = [6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]$$



3. Definition von Konstanten (2)

Rundenfunktionen:

Runde 1: $f_1(X, Y, Z) = XY \vee \bar{X}Z$

Runde 2: $f_2(X, Y, Z) = XZ \vee Y\bar{Z}$

Runde 3: $f_3(X, Y, Z) = X \oplus Y \oplus Z$

Runde 4: $f_4(X, Y, Z) = Y \oplus (X \vee \bar{Z})$

Runde 1, wenn $0 \leq j \leq 15$, dann $i=1$

Runde 2, wenn $16 \leq j \leq 32$, dann $i=2$

Runde 3, wenn $33 \leq j \leq 47$, dann $i=3$

Runde 4, wenn $48 \leq j \leq 63$, dann $i=4$

3. Definition von Konstanten (3)

Startvariablen:

$$h_1 = 67452301$$

$$h_2 = \text{EFCDAB89}$$

$$h_3 = \text{98BADCFE}$$

$$h_4 = 10325476$$

4. Schleifendurchlauf

Für jeden der 512 Bit-Blöcke, tue

1. Aufteilung des 512-Bit Blocks in 32-Bit Blöcke (16 Stück)
2. for($j=0$; $j \leq 63$; $++j$) (4 Funktionen x 16 Blöcke = 64)
 - 2.1 $(A, B, C, D) \leftarrow (H_1, H_2, H_3, H_4)$
 - 2.2 $temp \leftarrow (A + f_i(B, C, D) + X[z[j]] + y[j])$
 - 2.3 $A \leftarrow D$
 - 2.4 $B \leftarrow B + ROTL(temp, s[j])$
 - 2.5 $(C, D) \leftarrow (B, C)$
 - 2.6 $(H_1, H_2, H_3, H_4) \leftarrow (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$
3. Hashwert: $H_1 || H_2 || H_3 || H_4$

4. Aufteilung in 32-Bit Blöcke

101010101110101 ... 11010010100011

← 512-Bits →

↓ Aufteilung in 16 Blöcke zu je 32 Bits

10101...1101

← 32-Bits →

X_0

00001...0101

← 32-Bits →

X_1

...

00110...0110

← 32-Bits →

X_{15}

Vergleich MD5 mit SHA1

Kriterium	MD5	SHA-1
Hashlänge	128	160
Startwerte	67452301 EFC DAB89 98BADC FE 10325476	67542301 EFC DAB89 98BADC FE 10325476 C3D2E1F0
Rundenzahl	4	4
Blocklänge	512	512

Geburtstagsparadoxon

- ▶ Bei 23 beliebigen Personen ist die Wahrscheinlichkeit, daß 2 Personen am selben Tag Geburtstag haben $>50\%$
- ▶ Kein echtes Paradoxon, aber Aussage scheint im ersten Moment unwahrscheinlich
- ▶ N Personen: $\frac{N \cdot (N-1)}{2}$ verschiedene Paare, die am selben Tag Geburtstag haben könnten
→ Pr steigt für kleine N quadratisch

Birthday-Attack

- ▶ Idee: Es ist einfacher beliebige Kollisionen für eine One-Way-Hashfunktion zu finden, als Urbilder bzw. 2. Urbilder für einen konkreten Hashwert zu bestimmen
- ▶ Eingegebener Text x_1 wird durch symmantisch äquivalenten Text x'_1 ersetzt
- ▶ Anwendung:
 - ▶ Angreifer signiert Text mit $h(x'_1)$ und behauptet später x'_2 signiert zu haben
 - ▶ Angreifer fordert Opfer auf x'_1 zu signieren und behauptet dann, dass Opfer x'_2 signiert hat

Algorithmus – Yuvals Angriff

- ▶ **Eingabe:** Seriöse Nachricht x_1 ; Böswillige Nachricht x_2 ; m -bit One-Way Hashfunktion h
- ▶ **Ausgabe:** x'_1, x'_2 resultierend aus kleinen Veränderungen von x_1, x_2 mit $h(x'_1) = h(x'_2)$
 1. Generierung von $t = 2^{\frac{m}{2}}$ kleinen Veränderungen x'_1 von x_1
 2. Hashen dieser veränderten Nachrichten und Abspeichern der Kombination Hashwert–Nachricht
 3. Generierung von kleinen Veränderungen x'_2 von x_2 bis $h(x'_2)$ in Tabelle gefunden wird
- ▶ Kann auf alle unkeyed Hashfunktionen angewendet werden und liegt in der Komplexitätsklasse $O(2^{\frac{m}{2}})$ (sei m die Bitlänge des Hashwertes)

Praxisrelevanz

- ▶ Für 128 bit Hashfunktion reichen 64 potentielle Modifikationspunkte aus, um 2^{64} Nachrichtenvariationen zu erzeugen
- ▶ Komplexität: $O(2^{64})$, durch extreme Parallelisierung berechenbar
- ▶ Aber: Speicherproblem:
 - ▶ 148 Mio TB bei 8 Byte Speicher für Modifikationspunkte
 - ▶ 3 Nachrichten pro m^2 der Erdoberfläche

Pseudo-Kollisionen

- ▶ Attacken auf modifizierte Varianten von Hashfunktionen
- ▶ Z.B. freie Wahl der IV; stellt obere Schranken der Sicherheit einer Hashfunktion dar
- ▶ Eingeschränkte Angriffe können in einigen Fällen auf die gesamte Hashfunktion ausgedehnt werden

Schwächen von MD5

- ▶ August 2004 – Chinesisches Wissenschaftlerteam (Wang, Feng, Lai, Yu) findet erste Kollision in der vollständigen MD5-Funktion, sowie in MD4, HAVAL-128 und RIPEMD
- ▶ 1024 bit Nachrichten (2 Blöcke)
- ▶ $h(M|N) = h(M'|N')$
- ▶ IBM P690-Cluster, eine Stunde zur Bestimmung von M und M' , danach 15s bis 5m um N und N' zu finden
- ▶ Weiterhin schwierig Kollisionen für einen bestimmten Hash zu finden



Quellen



FERGUSON und SCHNEIDER: *Practical Cryptography*.
Wiley Publishing, Inc, Indianapolis, Indiana, 2003.



MENEZES, OORSCHOT und VANSTONE: *Applied Cryptography*.
CRC Press, Boca Raton, Florida, 1997.



Wikipedia.
URL <http://www.wikipedia.de>.



STINSON: *Cryptography – Theory and Practice*.
Chapman & Hall/CRC, Boca Raton, Florida, 2. Auflage, 2002.

