Kryptographische Hashfunktionen

Proseminar/Seminar
Kryptographie und Datensicherheit
SoSe 2009 – Universität Potsdam

Jan Jantzen

Gliederung

- Datenintegrität
- Hash- und Kompressionsfunktionen
- Sicherheit von Hashfunktionen
  - Zufallsorakelmodell
  - Vergleich von Sicherheitskriterien
- Konstruktion von Hashfunktionen
  - Das Merkle-Damgård-Konstrukt
  - Der Secure Hash Algorithm (SHA)
- Message Authentication Codes (MACs)
- Unbedingt Sichere MACs

Datenintegrität

In der Kryptographie ist Datenintegrität ein Sicherheitsaspekt, der verhindern soll, dass Nachrichten gefälscht oder manipuliert werden können.

Problem:

- Verschlüsselung schützt nur vor passiven Angriffen, nicht gegen aktive.
- Angreifer könnten Nachrichten abfangen und modifiziert weiterleiten.
- Der Empfänger kann nicht feststellen, dass eine Nachricht angegriffen wurde.

Hash- und Kompressionsfunktionen

Schutz durch separate Kontrollinformation.

Im Prinzip:

- Hashfunktionen erzeugen Hashwerte als Kontrollinformation.
- Absender ergänzt Kontrollinformation zur Nachricht
- Empfänger berechnet Hashwert aus Nachricht und vergleicht.
- Werte sollen nur übereinstimmen, wenn die Nachricht nicht verändert wurde.

Hash- und Kompressionsfunktionen

Def: Eine Hashfamilie ist ein Tupel (X, Y, K, H), für das gilt:

1. X ist eine Menge möglicher Nachrichten.
2. Y ist eine endliche Menge möglicher Hashwerte.
3. K ist eine endliche Menge möglicher Schlüssel.
4. Für jedes $k \in K$ existiert eine Hashfunktion $h_k : X \to Y \in H$.

Schlüsselfreie Hashfuktionen können als Familien angesehen werden, in denen |K| = 1 ist.

Ist X eine endliche Menge, so ist die Hashfunktion eine Kompressionsfunktion. Hier gilt $|X| \leqslant |Y|$.

Sicherheit von Hashfunktionen

Bei einer Hashfunktion $h : X \rightarrow Z$ ist es meist das Ziel, dass Paare (x,y) mit h(x)=y zu finden nur durch Auswahl einer Nachricht x und errechnen von h(x) effizient lösbar ist.

Hierzu gibt es drei Aspekte die eine Hashfunktion erfüllen soll:

1. Einwegfunktion (Urbild resistent)
2. Schwach kollisionsresistent
3. Stark kollisionsresistent

Sicherheit von Hashfunktionen

Problem 1: Finde Urbild

Gegeben: Hashfunktion h , message digest y

Finde: x, mit h(x)=y.

Eine Hashfunktion h ist eine **Einwegfunktion**, wenn es keinen effizienten Algorithmus gibt, der Problem 1 löst.

Problem 2: Finde zweites Urbild

Gegeben: Hashfunktion $h : X \rightarrow Z$, Urbild x.

Finde: x', mit $x \neq x'$ und $h(x')=h(x)$.

Eine Hashfunktion h ist **schwach kollisionsresistent**, wenn es keinen effizienten Algorithmus gibt, der Problem 2 löst.

Sicherheit von Hashfunktionen

Problem 3: Finde Kollision
Gegeben: Hashfunktion $h : X \rightarrow Z$.
Finde: x, x', mit $x \neq x'$ und $h(x')=h(x)$.

Eine Hashfunktion h ist **stark kollsionsresistent**, wenn es keinen effizienten Algorithmus gibt, der Problem 2 löst.

Starke kollisionsresistenz setzt schwache kollisionsresistenz und Einwegfunktion voraus, da sich Problem 3 auf Problem 2 oder Problem 1 reduzieren lässt.

Zufallsorakelmodell

Zufallsorakelmodell

Auch Random Oracle Model (RO).

Ist theoretische Herangehensweise.

- Hashfunktionen werden idealisiert als zufällige Funktionen modelliert.
- Ermöglicht und vereinfacht Untersuchungen und Sicherheitsbeweise kryptographischer Verfahren, welche Hashfunktionen verwenden.
- Für in der Praxis verwendete Hashfunktionen kann man die Einwegeigenschaft oder die Kollisionsresistenz nicht beweisen.
- Hashfunktionen im RO haben diese Eigenschaften.
- Man kann eine zufällige (Hash)Funktion nicht effizient als ganzes beschreiben.
- Daher Realisierung/Simulierung durch ein Orakel.

Zufallsorakelmodell

- Bei einer Anfrage nach dem Hashwert von x geht das Orakel wie folgt vor:
  - Das Orakel überpüft, ob h(x) schon einmal erfragt und berechnet wurde, wenn
    - ja, dann wird dieser Wert zurückgegeben.
    - nein, dann wird ein zufälliger Wert zurückgegeben und als h(x) gespeichert.
- Es wird eine zufällige Funktion $h : X \rightarrow Y$ definiert.
- Eine Orakelanfrage zählt in der Laufzeit eines Algorithmus als ein Schritt (konstante Zeit).

Zufallsorakelmodell

Angriffe im Zufallsorakelmodell

- **Las Vegas Typ Algorithmus:** Ein Algorithmus, mit Zufallselementen, der keine feste Laufzeit hat. Hier so modifiziert, dass er in bestimmter Zeit terminiert, aber Fehlschlag als Ergebnis liefern kann.
- **(e,Q)-Algorithmus:** Wird hier benutzt, um einen Las Vegas Typ Algorithmus zu bezeichnen, der mit Q Orakelbefragungen mit einer Durchschnittswahrscheinlichkeit von e ein Ergebnis liefert.
- **Geburtstagsparadoxon:** Beispiel, dafür dass Wahrscheinlichkeiten intuitiv häufig falsch eingeschätzt werden.

Es besagt, dass die Wahrscheinlichkeit bei 23 zufällig ausgewählten Personen zwei zu haben, die den gleichen Geburtstag haben größer als 50% ist.(Bei angenommener Gleichverteilung der Geburtstage)

Zufallsorakelmodell

Algorithmus 1

Finde_Urbild(h,y,Q){

wähle zufällig $X_0 \subseteq X$, $|X_0| = Q$.

für jedes $x \in X_0$ tue {

befrage Orakel nach h(x).

wenn h(x) = y Ergebnis: x.

}

Ergebnis: Fehlschlag.

}

Bei h : X →Y

Ist die Erfolgswahrscheinlichkeit für Algorithmus 1:

$e=1-(1-1/|Y|)^Q$.

Bei $|Y| = 2^{16} = 65535$ Muss Q größer gleich 45426 sein um eine Wahrscheinlichkeit größer 0.5 zu erreichen.

Zufallsorakelmodell

Algorithmus 2

Finde_2_Urbild(h,x,Q){

wähle zufällig $X_0 \subseteq X / x$, $|X_0| = Q - 1$.

y = h(x)

für jedes $x_0 \in X_0$ tue {

befrage Orakel nach $h(x_0)$.

wenn $h(x_0) = y$ Ergebnis: x_0.

}

Ergebnis: Fehlschlag.

}

Bei $h : X \rightarrow Y$
Ist die
Erfolgswahrscheinlichkeit
für Algorithmus 1:

$e = 1 - (1 - 1/|Y|)^{Q-1}$

Bei $|Y| = 2^{16} = 65535$ Muss Q größer gleich 45425 sein um eine Wahrscheinlichkeit größer 0.5 zu erreichen.

Zufallsorakelmodell

Algorithmus 3

Finde_Kollision(h,Q){

wähle zufällig $X \subseteq X, |X_0| = Q - 1$.

für jedes $x \in X_0$ tue {

befrage Orakel nach h(x).

wenn h(x) = h(x') für ein x≠x' Ergebnis: x.

}

Ergebnis: Fehlschlag.

}

Bei h : X → Y ist die Erfolgswahrscheinlichkeit für Algorithmus 3:

$$e = 1 - \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\ldots\left(\frac{M-Q+1}{M}\right) \qquad M = |Y|$$

Für e =0,5 ca. $1{,}18 * \sqrt{M}$.

Dies ist also ein $(0.5, O(\sqrt{M})) - Algorithmus$

Zufallsorakelmodell

Im Zufallsorakelmodell ergibt sich zusammenfassend:

- k Bit Sicherheit bezüglich der Einweg-Eigenschaft und der schwachen Kollisionsresistenz.
- Nur k/2 $((\sqrt{2^k})=2^{(k/2)})$ Bit Sicherheit bezüglich der starken Kollisionsresistenz.

Von einer ”guten“ Hashfunktion fordert man daher im Standardmodell (d.h. nicht im Zufallsorakelmodell), dass Urbilder und Kollisionen nur mit Aufwand

ungefähr 2^k

bzw. $2^{k/2}$

berechnet werden können sollen.

In der Praxis fordert man zur Zeit $k \geq 160$.

Vergleich von Sicherheitskriterien

Die folgenden zwei Reduktionen sind im Standardmodell gültig.

Sei $h : X \rightarrow Z$ eine Hashfunktion.
Können wir zweite Urbilder berechnen, so können wir Kollisionen berechnen:

- Wähle $x \in X$ zufällig.
- Berechne ein zweites Urbild $x' = x$ mit $h(x') = h(x)$.
- Ausgabe von x, x'.

Resistenz gegen Kollisionen impliziert Resistenz gegen schwache Kollisionen.

Vergleich von Sicherheitskriterien

Sei $h : X \rightarrow Y$ eine Kompressionsfunktion mit $|X| \geq 2|Y|$.
Können wir Urbilder berechnen, so können wir Kollisionen berechnen:

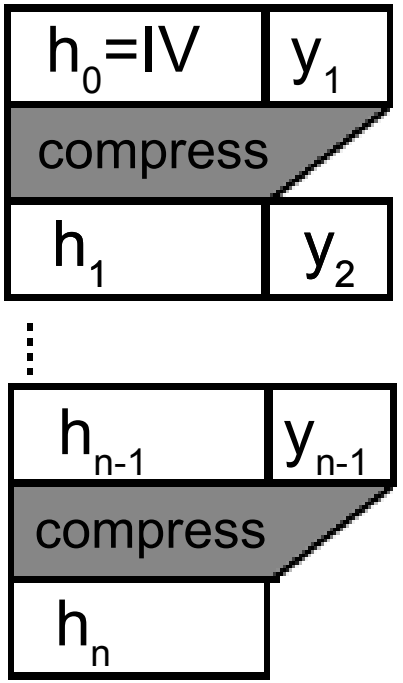
- Wähle $x \in X$ zufällig.
- Berechne ein Urbild x' von $h(x)$.
- Ausgabe von x, x', wenn $x \neq x'$. Sonst Fehler.

Resistenz gegen Kollisionen impliziert die Einweg-Eigenschaft.

Konstruktion von Hashfunktionen

Neben Einwegeigenschaft und Kollisionsresistenz sollen Hashwerte von langen Nachrichten effizient ohne großen Speicheraufwand berechnet werden können.

Allgemeines Prinzip: **Iterierung**
Nachricht mit geeignetem Bitstring und Nachrichtenlänge auffüllen (padden), dann in geeignete Blöcke m_n der Blocklänge (Bitlänge) r aufteilen.
Bei der Berechnung des Hashwerts eine Zustandsvariable h_n der Bitlänge i mitführen.
Erster Zustand ist h_0 = konstant IV, letzter Zustand h_n ist Hashwert.



In jedem Schritt eine Kompressionsfunktion auf y_{n+1} und h_n anwenden, liefert h_n+1.

Das Merkle-Damgård-Konstrukt

Hashfunktion $h : \{0,1\}^* \rightarrow \{0,1\}^n$ aus Kompressionsfunktion bauen.

Sei $f : \{0,1\}^m \rightarrow \{0,1\}^n$ eine Kompressionsfunktion und $r = m-n \geq 2$.

Der Hashwert von $x \in \{0,1\}^*$ wird dann wie folgt ausgerechnet:

- x hinten mit beliebigen Bits und der Nachrichtenlänge auffüllen und in s Blöcke $x_i \in \{0,1\}^r$ mit $0 \leq i \leq s-1$ aufteilen.
- $h_0 = IV$ für einen festen Initialwert.
- $h_i+1 = f(h_i||x_i)$ für $0 \leq i \leq s-1$.
- Der Hashwert ist h_s.

Das Merkle-Damgård-Konstrukt

Theorem: Für eine kollisionsresistente Kompressionsfunktion g ist auch die durch die Merkle-Damgård erhaltene Hashfunktion h kollisionsresistent.

Beweisidee: Widerspruchbeweis durch zeigen, das gilt: Wenn h nicht kollisionsresistent, dann ist g auch nicht kollisionsresistent.

Das Merkle-Damgård-Konstrukt

x_i = i. Block von x, s = Anzahl Blöcke,
h_s = Hashwert.

Beweis: Sei $h(x) = h(x')$ mit $x = x'$.
Definiere x_i, x'_i, s, s', h_i, h'_i wie in der Konstruktion. Wir können
$s \leq s'$ annehmen. Es gilt $h_s = h'_{s'}$.
Gilt $(h_{s-j}, x_{s-j}) = (h'_{s'-j}, x'_{s'-j})$ für alle $1 \leq j \leq s$, so folgt $s = s'$
wegen des Paddings und dann $x = x'$ im Widerspruch zur Annahme.
Sei also j minimal mit $1 \leq j \leq s$ und $(h_{s-j}, x_{s-j}) = (h'_{s'-j}, x'_{s'-j})$.
Dann gilt
$h_{s-(j-1)} = h'_{s'-(j-1)}$, $h_{s-(j-1)} = g(h_{s-j} || x_{s-j})$ und $h'_{s'-(j-1)} = g(h'_{s'-j} || x'_{s'-j})$.
Hieraus folgt: $g(h_{s-j} || x_{s-j}) = g(h'_{s'-j} || x'_{s'-j})$
Somit gibt es eine Kollision der Kompressionsfunktion. Diese kann durch einen Algorithmus ”effizient“ gefunden werden, in dem die Merkle-Damgård Konstruktion für x und x' ausgeführt wird.

Secure Hash Algorithm

SHA-1 (Secure Hash Algorithm):

- Von der NSA 1995 veröffentlicht,
- 160 Bit Hashwerte (interne Blockgröße 512 Bit).
- Am weitesten verbreitete Hashfunktion.
- Im ISO/IEC 10118-3 und FIPS180-1 standardisiert.
- Sicherheit angeknackst:

  Auf der CRYPTO 2005 wurde von Xiaoyun Wang, Andrew Yao und Frances Yao ein verbesserter Kollisionsangriff auf SHA-1 mit einer Laufzeit von ca. 2^{63} anstelle von 2^{80} vorgestellt.

SHA-256, SHA-384, SHA-512:

- Im FIPS180-2.
- 256, 384 und 512 Bit Hashwerte.
- 2001 veröffentlicht.

Secure Hash Algorithm

SHA-1

SHA-1 Padding:
- Eingabe x. Setze d ←(447−|x|) mod 512.
- L ← Binärdarstellung von |x|, wobei |L| = 64.
- y←(x||1||0d||L). Ausgabe y.

∨ logisches Oder, ∧ logisches Und, - Negation.

80 Funktionen:

$$f_i(B,C,D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & \text{für } 0 \le i \le 19 \\ B \oplus C \oplus D & \text{für } 20 \le i \le 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{für } 40 \le i \le 59 \\ B \oplus C \oplus D & \text{für } 60 \le i \le 79. \end{cases}$$

Secure Hash Algorithm

SHA-1

80 Konstanten:

$$K_i = \begin{cases} 5A827999 & \text{für } 0 \le i \le 19 \\ 6ED9EBA1 & \text{für } 20 \le i \le 39 \\ 8F1BBCDC & \text{für } 40 \le i \le 59 \\ CA62C1D6 & \text{für } 60 \le i \le 79. \end{cases}$$

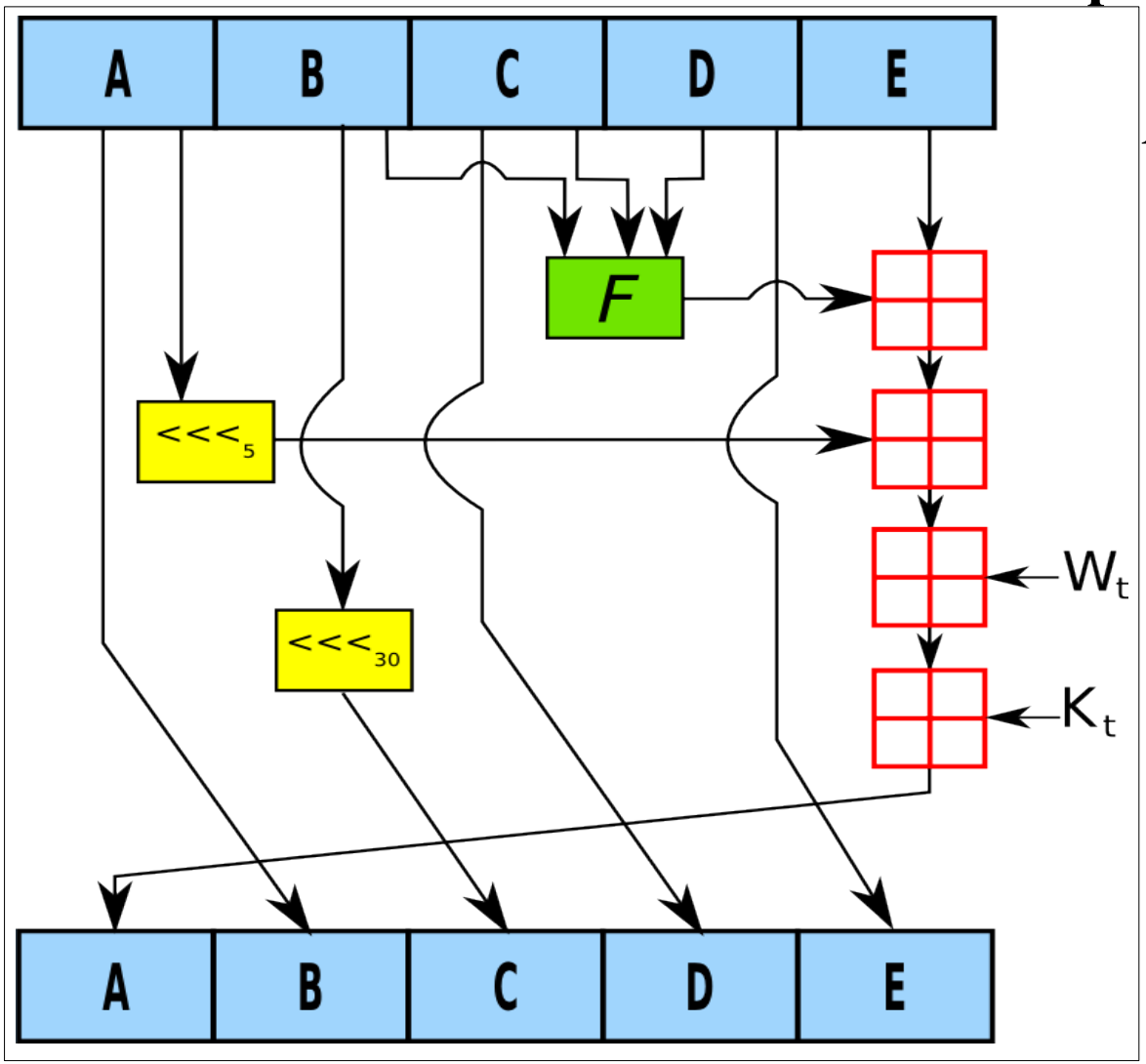
$ROTL^x$ = zyklischer Shift um x Bits nach Links. + Addition modulo 2^{32}.

Kompressionsfunktion $f : \{0,1\}^{512} \times \{0,1\}^{160} \rightarrow \{0,1\}^{160}$:

- Eingabe $M \in \{0,1\}^{512}$, $H \in \{0,1\}^{160}$.
- Schreibe $M = W_0 || \ldots || W_{15}$ mit $W_i \in \{0,1\}^{32}$.
- Schreibe $H = H_0 || \ldots || H_4$ mit $H_i \in \{0,1\}^{32}$.
- Für $t \leftarrow 16, \ldots, 79$: $W_t \leftarrow ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$.
- $A \leftarrow H_0, \ldots, E \leftarrow H_4$.

Secure Hash Algorithm

Eine Iteration der SHA-1 Kompressionsfunktion



A, B, C, D und E sind 32-bit Wörter.

F: variierende Funktion

$<<<_n$ ist $ROTL^n$

W_t ist das Nachrichtenwort Aus Runde t,

K_t ist die Konstante der Runde t.

⊞ ist Addition modulo 2^{32}.

Secure Hash Algorithm

SHA-1

- Für t ←0, . . . ,79:
  y←ROTL^5 (A)+ f_t(B,C,D)+E +W_t +K_t.
  E ←D, D←C, C←ROTL^{30}(B).
  B←A, A←y.
- H_0←H_0 +A, . . . , H_4←H_4 +E.
- Ausgabe H_0|| . . . ||H_4.

SHA-1:

- Eingabe x ∈ {0,1}*.
- x← SHA-1 Padding(x).
- Schreibe x = M_1|| . . . ||M_n mit M_i ∈ {0,1}^{512}.
- H ←67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0.
- Für i←1, . . . ,n: H ← f (M_i,H).
- Ausgabe von H.

Message Authentication Codes

Message Authentication Codes

Entspricht Hashfunktionen mit geheimen Schlüsseln.
$h : K \times M \rightarrow H$, $MAC = h_k(m)$.

- h parametrisierte Hashfunktion.
- m Nachricht.
- k geheimer Schlüssel.

Mit der Nachricht m wird $h_k(m)$ übertragen. Der Empfänger berechnet $h_k(m)$ aus m und k und vergleicht mit dem gesendeten MAC.

Liefert Datenintegrität und Authentizität.

Message Authentication Codes

Angriffe auf Message Authentication Codes:

- **No message attack**
  - Angreifer kennt Hashfamilie aber nicht den konkreten Schlüssel
  - Angreifer versucht ein gültiges Paar (x,MAC) zu bestimmen
- **Known message attack**
  - Angreifer kennt gültige Paare (x_1,MAC_1), ... , (x_Q,MAC_Q)
  - Angreifer versucht ein neues gültiges Paar zu erzeugen
- **Chosen message attack**
  - Angreifer kann MACs für selbstgewählte x_1, … , x_Q bestimmen ohne K zu kennen
  - (e, Q)-Fälschung: Angreifer versucht mit Wahrscheinlichkeit e>0 ein neues gültiges Paar zu erzeugen
  - Ein MAC ist sicher, wenn keine chosen message attack effizient ist

Message Authentication Codes

Geschachtelte MACs

Seien $g : K_1 \times \{0,1\}^* \rightarrow \{0,1\}^m$ und $h : K_2 \times \{0,1\}^m \rightarrow \{0,1\}^n$, mit $m \geq n$.

Wir zeigen: Ist g_{k1} kollisionsresistent bei unbekanntem Schlüssel und H_{k2} ein sicherer MAC, so ist $h_{k2} \circ g_{k1}$ ein sicherer MAC.

Wir betrachten dazu folgende Angreifer:

1. Kollisionsangriff bei unbekanntem Schlüssel: k_1 ist geheim, der Angreifer erhält trotzdem die Werte $g_{k1}(x)$ für x seiner Wahl. Er versucht eine Kollision $g_{k1}(x_i) = g_{k1}(x_j)$ mit $x_i = x_j$ zu finden.
2. Kleiner MAC Angreifer: Angreifer gegen h_{k2}.
3. Großer MAC Angreifer: Angreifer gegen $h_{k2} \circ g_{k1}$.

Erwartete (ideale) Sicherheit bei 1. ist x/2 Bits.
Erwartete (ideale) Sicherheit bei 2. ist n Bits.

Message Authentication Codes

Geschachtelte MACs

Ein (e,Q, t)-Angreifer gegen 1, 2 oder 3 führt einen erfolgreichen Angriff bei zufälliger und gleichverteilter Schlüsselwahl mit Wahrscheinlichkeit e und Q Orakelanfragen in Zeit t aus.
Theorem: Gibt es einen (e,Q, t)-Angreifer gegen 3, so gibt es auch einen $(e_1,Q+1, t)$-Angreifer gegen 1 und einen (e_1,Q, t)-Angreifer gegen 2 mit $e_1 + e_2 = e$.

Anwendung: Falls es keinen ($\geq e, \leq Q, \leq t$)-Angreifer gegen 1 oder 2 gibt, so gibt es auch keinen ($\geq 2e, \leq Q-1, \leq t$)-Angreifer gegen 3.

Message Authentication Codes

HMAC

Als Anwendung des Theorems ergeben sich HMACs.
Gegeben eine Hashfunktion $h : \{0,1\}^* \rightarrow \{0,1\}^b$.

HMAC von Nachricht x und Schlüssel k:

- $HMAC = h(k||opad||h(k||ipad||x))$.
  $HMAC = h(k \oplus opad||h(k \oplus ipad||m))$.
- opad = 0x3636 ... 36.
- ipad = 0x5C5C ... 5C.

Die Benutzung von k anstelle von k_1 und k_2 basiert auf der Annahme, dass der ”Unterschied“ von einem Angreifer aufgrund der Hashfunktionseigenschaften nicht bemerkt werden kann.

Message Authentication Codes

HMAC

Innere Anwendung von h im HMAC:

- Benötigt Sicherheit bezüglich Kollisionen bei unbekanntem k.

Äußere Anwendungen von h im HMAC:

- Die Länge des Paddings wird so eingestellt, dass eine volle Blocklänge der Kompressionsfunktion von h erreicht wird.
- Damit wird bei der zweiten Berechnung von h nicht intern iteriert.
- Benötigt Sicherheit der Kompressionsfunktion als MAC (Pseudozufallsfunktion).

Wegen Geburtstagsangriffen ist die Sicherheit von HMAC bei iterierten Hashfunktionen trotzdem nur $2^{b/2}$.

Relativ gutes Beispiel:

- HMAC mit h = SHA-256, MAC-Wert bei Bedarf auf 128 Bit kürzen.

Message Authentication Codes

CBC-MAC: Iterative Konstruktion

- Verwende Kryptosystem mit E: K x $\{0, 1\}^m \rightarrow \{0, 1\}^m$
- Zerlege Bitstring (nach Padding) in Wörter $x_1, x_2, \ldots, x_t$ der Länge m
- Wähle Initialwert $y_0 \in \{0, 1\}^m$ (z.B. $y_0 := 0^m$)
- In Schritt $i \leq k$ berechne $y_i = E_K(y_i - 1 \oplus x_i)$
- Ergebnis ist $MAC(K, x) = y_t$

Sicherheit:

- Bester bekannter Angriff (Geburtstagsattacke) liefert $(0.5, 2^{m/2})$ fälschung

Unbedingt sichere MACs

Hier gehen wir davon aus, dass der Sclhüssel nach jeder Nachricht ausgetauscht wird.

Deshalb sind nur (e,0)- und (e,1)-Fälschungen möglich.

Ein MAC-Algorithmus ist dann unbedingt sicher, wenn es keine (e,0)- oder (e,1)-Fälscher gibt, bei denen e einen relevanten Wert hat.

Pd_q für q = {0,1} :

Deception Probability → max. Wert für e für eine (e,q)-Fälschung

payoff(x,y) :

Wahrscheinlichkeit, dass (x,y) gültiges Paar ist

payoff(x',y' ; x,y) :

bedingte Wahrscheinlichkeit, dass (x',y') gültiges Paar ist, wenn (x,y) gültig ist

Unbedingt sichere MACs

Authentizitätstabelle

Hash Family (X, Y, K, H)

X = Y = {0,1,2}

K = {0,1,2} x {0,1,2}

für alle K = (a,b)

h(a,b)= a x + b mod 3

(3,3)-Hash Family

key	0	1	2
(0,0)	0	0	0
(0,1)	1	1	1
(0,2)	2	2	2
(1,0)	0	1	2
(1,1)	1	2	0
(1,2)	2	0	1
(2,0)	0	2	1
(2,1)	1	0	2
(2,2)	2	1	0

Unbedingt sichere MACs

Beispiel für eine (e,0)-Fälschung:

Auswählen einer Nachricht x und raten eines korrekten y.

(x,y) ist Fälschung, wenn $h_k(x)=y$

$Pd_0 = 1/3$

Beispiel für eine (e,1)-Fälschung:

Angenommen es wurde das Paar (0,0) abgefangen.

Dies grenzt mögliche Schlüssel auf {(0,0),(1,0),(2,0)} ein.

Nun ist z.B. Paar (1,2) genau dann eine Fälschung, wenn k = (2,0).

$Pd_1 = 1/3$.

key	0	1	2
(0,0)	0	0	0
(0,1)	1	1	1
(0,2)	2	2	2
(1,0)	0	1	2
(1,1)	1	2	0
(1,2)	2	0	1
(2,0)	0	2	1
(2,1)	1	0	2
(2,2)	2	1	0

Referenzen

- Douglas R. Stinson: Cryptography: Theory and Practice. 3rd Edition, Chapman & Hall/CRC 2006
- Script zur Vorlesung Kryptographie im Wintersemester 2007 an der Technischen Universität Berlin. Dozenten: Prof. Dr. Florian Heß, Dipl.-Math. Osmanbey Uzunkol
  http://www.math.tu-berlin.de/~hess/krypto-ws2007/
- Wiki Projekt Kryptologie
  http://de.wikipedia.org/wiki/Wikipedia:WikiProjekt_Kryptologie
- Folien zur Veranstaltung Kryptographie und
  http://www.cs.uni-potsdam.de/ti/lehre/07-Kryptographie/slides.html
  Komplexität Universität Potsdam, Wintersemester 2007/2008.
  Prof. Kreitz

Alle Webadressen stand:18.05.2009