Hashfunktionen und MACs

Prof. Dr. Rüdiger Weis

TFH Berlin

Sommersemester 2008

Message Authentication Code

MAC: “Message Authentication Code”

- Was ist ein MAC?
- Der CBC-MAC
- Der XOR-MAC
- Kryptographische Hashfunktionen
- Iterierte Hashfunktionen

Message Authentication Code

- Nachrichten werden nicht (notwendigerweise) geheimgehalten, sondern vor *Veränderung* geschützt (authentisiert).
- Für eine Nachricht X wird ein schlüsselabhängiger *Authentifikationscode* $A = \mathrm{MAC}_K(X)$ berechnet.
- Zu jedem MAC gehört ein schlüsselabhängiger Test, ob ein Paar (Nachricht, Authentificationscode) *richtig* ist.
  Ein Paar $(X, \mathrm{MAC}_K(X))$ ist immer *richtig*.

Aufgabe eines MAC

MACs sichern die *Echtheit* (Authentizität) von Nachrichten!

Beispiele:

- Schutz von Bank-Transaktionen vor Verfälschung
- Schutz von Computerprogrammen vor Viren
- Schutz von mobilem Code vor
  - Veränderungen der Arbeitsweise
  - *Denial of Service* Angriffen
- ...

Formale Beschreibung eines Angriffs: "MAC-Orakel"

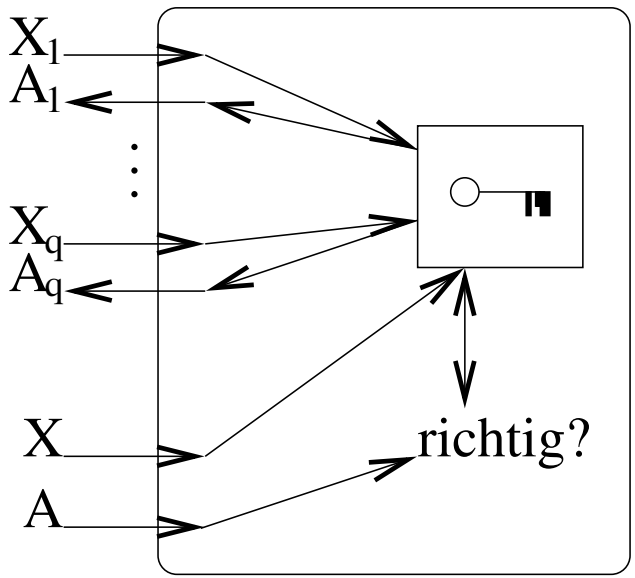
Fragephase:

Für $i := 1$ bis q:

- X_i an Orakel.
- Antw. $A_i = \mathrm{MAC}_K(X_i)$.

Ratephase: (X, A) an Orakel, $X \notin \{X_1, \ldots, X_q\}$.

Wertung: Gegner *gewinnt*, wenn (X, A) *richtig*.



Entwurfsziel:

Minimiere W'keit, dass der Gegner gewinnt!

Beispiele

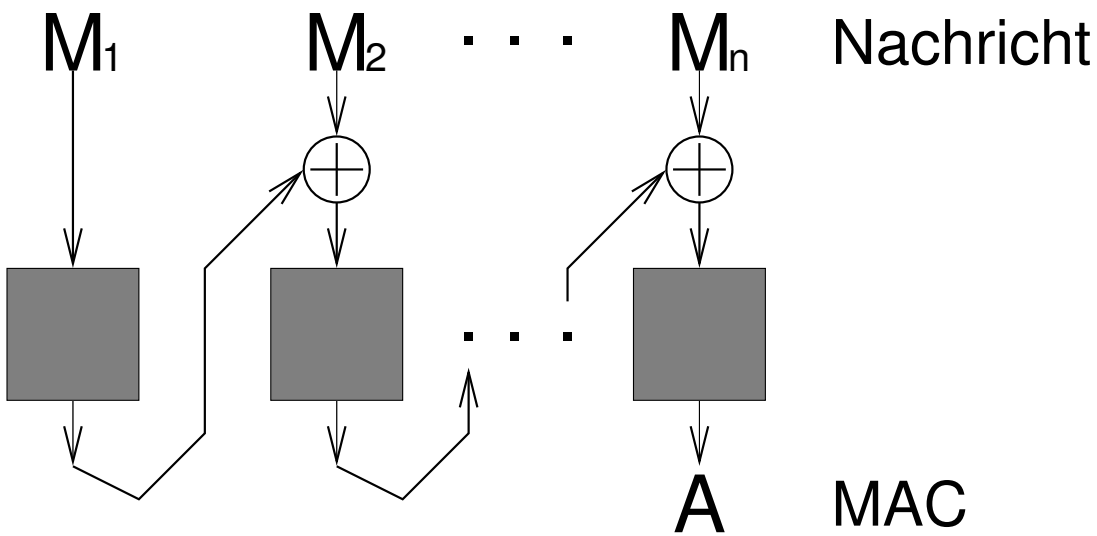
Welche sind *gute* MACs, welche *schlechte*?

1. Primzahl p, Nachricht $M = (M_1, \ldots, M_n) \in (\mathbb{Z}_p)^n$,
   Schlüssel $K = (K_1, \ldots, K_n) \in (\mathbb{Z}_p)^n$,
   MAC $A = M_1K_1 + M_2K_2 + \cdots + M_nK_n \mod p$.
2. Nachricht $M = (M_1, \ldots, M_n) \in (\{0,1\}^b)^n$,
   b-bit Blockchiffre E, Schlüssel K (für E),
   MAC $A = E_K(E_K(M_1) \oplus E_K(M_2) \oplus \cdots \oplus E_K(M_n))$.
3. Nachricht $M \in \{0,1\}^n$
   PZFG $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^l$, Schlüssel K,
   MAC $A = F_K(M)$.

In diesem Kapitel: Blockchiffre-basierte MACs ($\to$ Bsp. 2)

Das CBC-MAC Schema

Sei E eine b-bit Blockchiffre.
Verschlüssele $M = (M_1, \ldots, M_n) \in (\{0,1\}^b)^n$ im CBC-Modus, mit festem IV$= 0$.
Ignoriere $C_0, \ldots, C_{n-1}$ und benutze $A = C_n$ als MAC.



CBC-MAC (Bemerkungen)

- Der CBC-MAC ist weit verbreitet.
- Er gilt als sicher,
  wenn E sicher ist und die Länge der Nachrichten konstant ist.
- Der CBC-MAC ist aber unsicher, wenn
  Nachrichten verschiedener Länge authentifiziert werden!

CBC-MAC (Angriff)

Angriff bei variabler Nachrichtenlänge:

Fragephase:

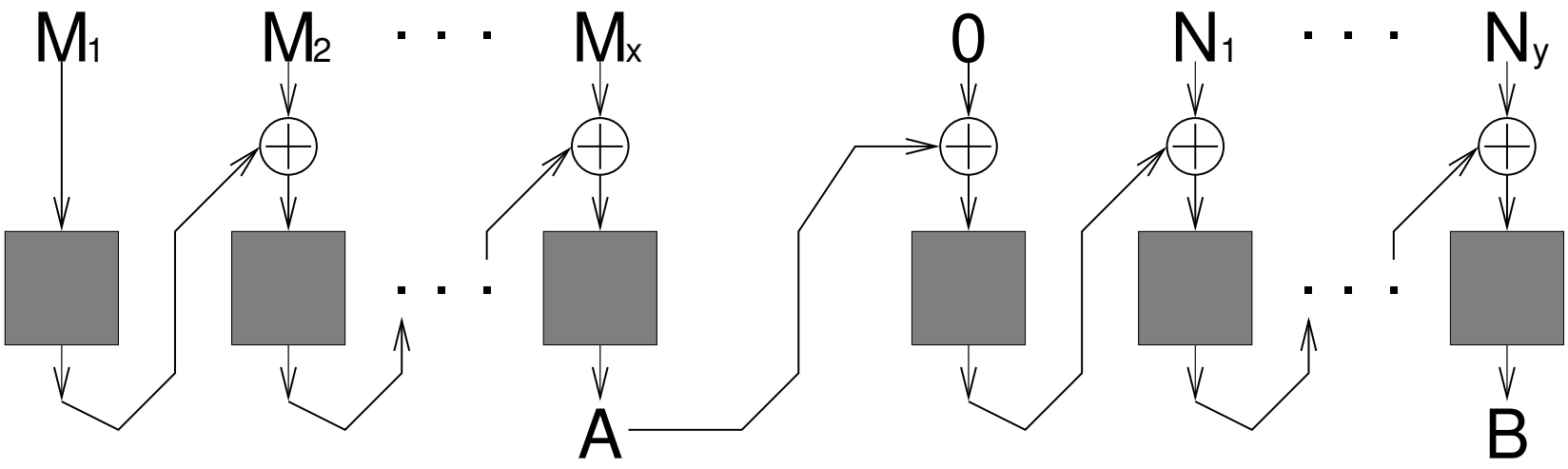
1. Erfrage MAC A für die Nachricht $M = (M_1, \ldots, M_x)$.
2. Erfrage MAC B für die Nachricht $N = (A, N_1, \ldots, N_y)$.

Ratephase: Das Paar (Z, B) mit

$$Z = (M_1, \ldots, M_x, 0, N_1, \ldots, N_y)$$

ist *richtig*!

CBC-MAC (Angriff, Graphik)



Das XOR-MAC Schema

- Blockchiffre E, b-bit Blocks, b gerade
- Parameter $L < b - 1$ (maximale Nachrichtenlänge), Parameter $m = b - L - 1$
- Konkatenation der Bit-Strings a und b: „$a||b$“
- „$< i >$“: Zahl $i \in \{0, \ldots, 2^L - 1\}$ als L-bit Zahl

Berechnen des XOR-MACs

Eine Nachricht $M = (M_1, \ldots, M_n) \in (\{0,1\}^m)^n$ wird wie folgt unter dem Schlüssel K verschlüsselt:

0. Wähle $R \in \{0,1\}^{b-1}$.
1. Berechne

$$Z := E_K(1||R) \oplus \bigoplus_{1 \le i \le n} E_K(0|| < i > ||M_i).$$

2. MAC $A = (R, Z)$.

Beachte:
Für jede Nachricht M gibt es viele *richtige* MACs. (Wieviele?)

Überprüfen des XOR-MACs

Ein Nachricht-MAC Paar (M, A) mit $M = (M_1, \ldots, M_n)$ und $A = (R, Z)$ wird wie folgt überprüft:

1. Berechne

$$Z^* := E_K(1||R) \oplus \bigoplus_{1 \leq i \leq n} E_K(0|| < i > ||M_i).$$

2. *Richtig* $\Longleftrightarrow Z = Z^*$.

Sicherheit des XOR-MACs

Wie sicher ist das XOR-MAC Schema?

Ein MAC-Schema ist (q, p)-sicher, wenn ein Gegner, der maximal q Orakelfragen stellt, höchstens mit der Wahrscheinlichkeit p gewinnt.

Sicherheit des XOR-MACs (2)

Man beachte, dass die Umkehrbarkeit der Funktion E_K weder gebraucht wird, um MACs zu berechnen, noch gebraucht wird, um Nachrichten mit einem MAC zu überprüfen.
Für die Analyse nehmen wir zunächst an, E_K sei keine Permutation sondern eine zufällige *Funktion*.

Theorem (XOR-MAC)

Sei $E_K : \{0,1\}^b \to \{0,1\}^b$ *eine Zufallsfunktion. Dann ist das XOR-Schema*

$$\left(q, \frac{q^2+1}{2^b}\right) \text{-sicher.}$$

Bemerkung

Die in Satz [XOR-MAC] angegebene Schranke ist ziemlich scharf:

- Für $q = 0$ ist ein Angriff mit der Erfolgswahrscheinlichkeit

$$\frac{q^2 + 2}{2^{n+1}} = \frac{2}{2^{n+1}} = \frac{1}{2^n}$$

  trivial.
- Sei $q = 2u + 1$.

Sicherheit des XOR-MACs (Satz)

Theorem

Sei $E_K : \{0,1\}^b \to \{0,1\}^b$ *eine Zufallspermutation. Dann ist das XOR-Schema*

$$\left(q, \frac{3q^2+2}{2^{b+1}}\right) \text{-sicher.}$$

Beweis-Skizze

- Aus Kapitel 8: Unterscheidung zwischen Zufallsfkt. und Zufallsperm. mit dem Vorteil $\leq q^2/2^{b+1}$.
- Zusammen mit Satz [XOR-MAC] ergibt dies den behaupteten Vorteil

$$\leq \frac{q^2}{2^{b+1}} + \frac{q^2+1}{2^b} = \frac{3q^2+2}{2^{b+1}}. \qquad \square$$

Kryptographische Hashfunktionen

- Eine **Hashfunktion** bildet einen *großen* Input auf einen *kleinen* Output ab.
- Kollisionen sind Inputs $x \neq y$ mit $h(x) = h(y)$.
- Kollisionen sind „schlecht“!
  (In der Kryptographie und anderswo . . .)

Bsp: $h : (\{0,1\}^c)^* \to \{0,1\}^c$, $h(x_1, \ldots, x_n) = x_1 \oplus \cdots \oplus x_n$.

Sicherheitsanforderungen

Sei h eine (kryptographische) Hashfunktion. Wir definieren die folgenden Eigenschaften für h:

Kollisionsresistenz: Es ist praktisch unmöglich, eine Kollision zu finden.

Einweg-Eigenschaft: Sei ein zufälliger Input x gegeben. Es ist praktisch unmöglich, eine Kollision für x zu finden, d.h., einen Wert y mit $h(x) = h(y)$.

Theorem

Sei h eine kollisionsresistene Hashfunktion. Dann ist h auch (erst recht!) eine Einweg-Hashfunktion.

Beweis: (Trivial!)

Davies-Hash

Hashfunktionen aus Blockchiffren:

Man betrachte eine b-bit Blockchiffre mit k-bit Schlüsseln:

$$E : \{0,1\}^k \times \{0,1\}^b \to \{0,1\}^b$$

Wir konstruieren die Hashfunktion

$$h : \{0,1\}^{k+b} \to \{0,1\}^b, h(K,X) = E_K(X) \oplus X.$$

Wie sicher kann h sein,

(a) wenn E=DES oder E=Triple-DES ist, oder

(b) wenn E=AES gilt?

Iterierte Hashfunktionen

Sei $c < d$ und $h : \{0,1\}^d \to \{0,1\}^c$ eine Hashfunktion für d-bit Inputs. Wir konstruieren die Hashfunktion $H : \{0,1\}^* \to \{0,1\}^c$ für Inputs beliebiger Länge:

Eingabe: String $x \in \{0,1\}^L$ beliebiger Länge L.

Ausgabe: String $z = H(x) \in \{0,1\}^c$ der Länge c

Konstruktion: ($\to$ nächste Folie)

Iterierte Hashfunktionen (Konstr.)

1. **Padding:** Sei $k = \lceil (L+1)/(d-c) \rceil$.
   Erzeuge $y = (y_1, \ldots, y_k) \in \{0,1\}^{k(d-c)}$ aus
   1.1 den L bit des Strings x, gefolgt von
   1.2 einer 1 und $k(d-c) - L - 1$ Nullen.
2. **Iteration:**
   2.1 Sei $H_0 \in \{0,1\}^c$ ein festgelegter Initialwert.
   2.2 Für $i := 1$ bis k

$$\text{berechne } H_i = h(H_{i-1}, y_i) \in \{0,1\}^c.$$

Gib den Hashwert $z = H_k$ aus.

Iterierte Hashfunktionen (Sicherheit)

Theorem (Damgaard-Merkle)

Sei $c < d$ und sei $H : \{0,1\}^* \to \{0,1\}^c$ eine mit Hilfe von h $h : \{0,1\}^d \to \{0,1\}^c$ konstruierte iterierte Hashfunktion (wie zuvor beschrieben).

Dann gilt:

> *Wenn h kollisionsresistent ist, dann ist auch H kollisionsresistent.*

Hashfunktionen in der Praxis

Beispiele: SHA-1, RIPE-MD 160, ...

Iterierte Hashfunktionen auf Basis einer Hashfunktion konstanter Länge („Kompressionsfunktion“, s.u.)

Hash-Werte: mindestens 160 bit.
(Außer bei „abwärtskompatiblen“ Produkten ...)

Kompressionsfunktionen:

- Davies-Hash
- keine Allzweck-Blockchiffre (AES)
  sondern speziell entworfene „Blockchiffren“, mindestens 160 bit Blockgröße, typischerweise 512-bit „Schlüssellänge“.

Danksagungen

Danksagungen

- Aus einer Vorlesung von Stefan Lucks
- Erstellt mit Freier Software