

---

## 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.

---

1

7. November 2008

---

## Sicherheitsmodell

Der ideale MAC ist wieder eine Zufallsfunktion, für jedes  $k \in K$  soll  $h_k$  nicht von einer „zufällig“ gewählten Funktion  $M \rightarrow H$  unterscheidbar sein.

Spezieller betrachten wir auch folgendes (schwächeres) Sicherheitsmodell (\*):

- Ein Angreifer darf  $h_k(m_i)$  für beliebig von ihm vorgegebene Nachrichten  $m_i$  erhalten (Orakelanfragen).
- Der Angreifer gewinnt (erzeugt eine Fälschung), wenn er ein  $m$  und  $y$  mit  $y = h_k(m)$  berechnet.
- MAC ist sicher, wenn es keinen effizienten Angreifer gibt, der mit signifikanter Wahrscheinlichkeit gewinnt.

Sicherheit eines idealen MAC ist gleich der Anzahl  $\log_2(\#H)$  der Bits der MAC-Werte (d.h. Raten ist bereits die beste Strategie).

---

2

7. November 2008

---

## Konstruktion von MACs

Folgende Konstruktionstypen können unterschieden werden:

- MACs aus Blockchiffren im CBC Modus.
- MACs aus MDCs.
- Spezielle Konstruktionen.

CBC-MACs sind sehr weit verbreitet.

- Standardisiert z.B. in FIPS-113 von 1985.

Erhalten in der Regel Sicherheit von nur  $\log_2(\#H)/2$  Bits.

---

3

7. November 2008

---

## Hash-then-Encrypt

Naheliegend, hat aber keine guten Eigenschaften.

$h$  Hashfunktion und  $E_k$  Blockchiffre.

MAC-Wert von  $x$  ist  $E_k(h(x))$ .

Nachteile:

- Kollisionen können ohne  $k$  bestimmt werden.
- Kollisionen haben gleichen MAC für verschiedene  $k$ .
- $E_k$  darf kein Stromchiffre sein (für bekanntes  $x$  kann man nach dem Schlüsselstrom auflösen).

---

4

7. November 2008

---

## CBC-MAC

Verwendet eine Blockchiffre  $E : K \times \{0, 1\}^b \rightarrow \{0, 1\}^b$ .

CBC-MAC von Nachricht  $M \in \{0, 1\}^*$  unter Schlüssel  $k$ :

- Schreibe  $M = M_1 || \dots || M_n$  mit  $M_i \in \{0, 1\}^b$  (und Padding).
- $y_0 \leftarrow IV \leftarrow 0^b$ .
- Für  $i \leftarrow 1, \dots, n$ :  
$$y_i \leftarrow \mathcal{E}(k, y_{i-1} \oplus m_i).$$
- Ausgabe  $y_n$ .

In anderen Worten: Der CBC-MAC ist der letzte Chiffretextblock der Verschlüsselung von  $m$  mit  $\mathcal{E}$  im CBC Modus, unter Verwendung des konstanten  $IV = 0^b$ .

Vorteil: Leicht aus bestehenden Teilen programmiert.

---

5

7. November 2008

---

## CBC-MAC Padding und Postprocessing

In den Standards sind drei Padding Varianten vorgesehen:

- Nullen anhängen.
- Eine Eins und Nullen anhängen.
- Nullen anhängen und zusätzlichen Block mit Nachrichtenlänge.

Man kann ein optionales Postprocessing vornehmen:

- Wähle Schlüssel  $k_1$ . Dann MAC-Wert  $\mathcal{E}(k, \mathcal{D}(k_1, y_n))$ . Entspricht EDE-Verschlüsselung im letzten Schritt.
- Wähle Schlüssel  $k_1$ . Dann MAC-Wert  $\mathcal{E}(k_1, y_n)$ . Entspricht EE-Verschlüsselung (nicht besonders gut).

Erschwert exhaustive Key-search.

---

6

7. November 2008

---

## CBC-MAC Sicherheit

Gilt als sicher,

- wenn Blockchiffre sicher ist und
- wenn die Nachrichtenlänge konstant ist.

Man kann zeigen: Ist die Blockchiffre eine pseudozufällige Funktion, so auch der CBC-MAC. Nur durch mindestens ungefähr  $2^{b/2}$  Anfragen an die Blockchiffre kann der CBC-MAC von einer zufälligen Funktion unterschieden werden.

Ist aber unsicher, wenn die Nachrichtenlänge nicht konstant ist:

- Erfrage  $y_1 \leftarrow h_k(m_1)$ .
- Erfrage  $y_2 \leftarrow h_k(y_1 || m_2)$ .
- Nun gilt  $y_2 = h_k(m_1 || 0^b || m_2)$ . Liefert eine Fälschung.

Abhilfe:  $h_{h'(m)}(m)$  oder  $h_k(|m| || m)$  (also Padding) verwenden.

---

7

7. November 2008

---

## CBC-MAC Sicherheit

Geburtstagsangriff auf CBC-MAC mit fester Nachrichtenlänge  $d$ :

- Wähle  $1.18 \cdot 2^{b/2}$  Nachrichten  $m_i = m_{1,i} || m_{2,i} || m_3$  mit  $m_{1,i} \in \{0, 1\}^b$  paarweise verschieden,  $m_{2,i} \in \{0, 1\}^b$  zufällig und  $m_3 \in \{0, 1\}^{d-2b}$  beliebig.
- Erfrage alle  $h_k(m_i)$ . Es ergibt sich eine Kollision  $h_k(m_i) = h_k(m_j)$  für  $i \neq j$  mit Wahrscheinlichkeit  $\geq 1/2$ .
- Daraus folgt  $E(k, m_{1,i}) \oplus m_{2,i} = E(k, m_{1,j}) \oplus m_{2,j}$ .
- Wähle beliebiges  $m_\delta \in \{0, 1\}^b$  und erfrage  $y \leftarrow h_k(m_{1,i} || (m_{2,i} \oplus m_\delta) || m_3)$ .
- Nun gilt  $y = h_k(m_{1,j} || (m_{2,j} \oplus m_\delta) || m_3)$ . Liefert eine Fälschung.

Folgerung:  $2^{b/2}$  bestmögliche Sicherheit beim CBC-MAC,  $2^{b/2}$  Schritte genügen, um CBC-MAC von zufälliger Funktion zu unterscheiden.

---

8

7. November 2008

---

## MACs aus MDCs

Gegeben eine Hashfunktion  $h : \{0, 1\}^* \rightarrow \{0, 1\}^b$ .

Drei einfache Varianten:

1. MAC =  $h(k||m)$
2. MAC =  $h(m||k)$
3. MAC =  $h(k_1||m||k_2)$

Liefern für ideale Hashfunktion idealen MAC. Für iterierte Hashfunktionen aber nicht besonders sicher:

Annahme:  $h$  iteriert, also  $h_0 = IV$ ,  $h_{i+1} = f(h_i, m_i)$ , ...

zu 1. Aus  $h(k||m)$  kann leicht  $h(k||m||m')$  bzw.  $h(k||m||p||m')$  ausgerechnet werden, wobei  $p$  das Padding der Hashfunktion ist.

---

## MACs aus MDCs

zu 2. Finde Kollision  $h(m) = h(m')$  („offline“ möglich). Erfrage  $y \leftarrow h(m||k)$ . Dann ist  $y = h(m'||k)$  eine Fälschung.  
(Ist im Prinzip ein Hash-then-encrypt Ansatz, auf den ein analoger Angriff möglich ist.)

zu 3. Durch Erfragen der MACs findet man eine Kollision  $h(k_1||m_i||k_2) = h(k_1||m_j||k_2)$ . Dann gilt auch  $h(k_1||m_i) = h(k_1||m_j)$  mit nicht zu geringer Wahrscheinlichkeit ( $h \mapsto f(h, k_2)$  sollte im wesentlichen injektiv sein).

Man sucht nun einfach in  $K$  nach  $k_1$  und dann nach  $k_2$ .  
Der Aufwand ist damit  $2\#K$  statt  $\#K^2$  ...

Die Konstruktionen 1-3 werden daher so nicht verwendet.

---

## 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  $g_{k_1}$  kollisionsresistent bei unbekanntem Schlüssel und  $h_{k_2}$  ein sicherer (\*) MAC, so ist  $h_{k_2} \circ g_{k_1}$  ein sicherer (\*) MAC.

Wir betrachten dazu folgende Angreifer:

1. Kollisionsangriff bei unbekanntem Schlüssel:  $k_1$  ist geheim, der Angreifer erhält trotzdem die Werte  $g_{k_1}(m)$  für  $m$  seiner Wahl. Er versucht eine Kollision  $g_{k_1}(m_i) = g_{k_1}(m_j)$  mit  $m_i \neq m_j$  zu finden.
2. Kleiner MAC Angreifer: Angreifer gegen  $h_{k_2}$ .
3. Großer MAC Angreifer: Angreifer gegen  $h_{k_2} \circ g_{k_1}$ .

Erwartete (ideale) Sicherheit bei 1. ist  $m/2$  Bits.

Erwartete (ideale) Sicherheit bei 2. ist  $n$  Bits.

---

## Geschachtelte MACs

Ein  $(\epsilon, q, t)$ -Angreifer gegen 1, 2 oder 3 führt einen erfolgreichen Angriff bei zufälliger und gleichverteilter Schlüsselwahl mit Wahrscheinlichkeit  $\epsilon$  und  $q$  Orakelanfragen in Zeit  $t$  aus.

Thm: Gibt es einen  $(\epsilon, q, t)$ -Angreifer gegen 3, so gibt es auch einen  $(\epsilon_1, q+1, t)$ -Angreifer gegen 1 und einen  $(\epsilon_2, q, t)$ -Angreifer gegen 2 mit  $\epsilon_1 + \epsilon_2 = \epsilon$ .

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

Bew: Sei  $A_3$  ein  $(\epsilon, q, t)$ -Angreifer gegen 3. Seien  $(m_i, z_i)$  für  $1 \leq i \leq q$  die Orakelanfragen von  $A_3$  und die Ergebnisse. Es gilt  $h_{k_2}(g_{k_1}(m_i)) = z_i$ . Mit Wahrscheinlichkeit  $\epsilon$  liefert  $A_3$  eine Fälschung  $(m, z)$ , es gilt also  $m \neq m_i$  für alle  $i$  und  $h_{k_2}(g_{k_1}(m)) = z$ .

---

## Geschachtelte MACs

Der Angreifer  $A_1$  gegen 1 wird wie folgt definiert: Er wählt ein zufälliges  $k_2$  und beantwortet die Orakelanfragen von  $A_3$  mit  $z_i = h_{k_2}(g_{k_1}(m_i))$ , wobei er den Wert  $g_{k_1}(m_i)$  durch sein Orakel erhält. Nachdem  $A_3$  den Wert  $(m, z)$  ausgegeben hat, berechnet  $A_1$  den Wert  $g_{k_1}(m)$  durch Orakelanfrage und gibt  $m, m_i$  als Kollision aus, falls es ein  $i$  mit  $g_{k_1}(m) = g_{k_1}(m_i)$  gibt. Es gilt  $m \neq m_i$ , so daß  $A_1$  die Spielregeln befolgt.

Der Angreifer  $A_2$  gegen 2 wird wie folgt definiert: Er wählt ein zufälliges  $k_1$  und beantwortet die Orakelanfragen von  $A_3$  mit  $h_{k_2}(g_{k_1}(m_i))$  unter Verwendung seines Orakels für  $h_{k_2}$ . Nachdem  $A_3$  den Wert  $(m, z)$  geliefert hat, gibt  $A_2$  den Wert  $(g_{k_1}(m), z)$  als Fälschung aus, falls  $g_{k_1}(m) \neq g_{k_1}(m_i)$  für alle  $i$  gilt. Das Orakel  $h_{k_2}$  wurde dann nicht nach  $g_{k_1}(m)$  gefragt, so daß auch  $A_2$  die Spielregeln befolgt.

---

13

7. November 2008

---

## Geschachtelte MACs

$A_1$  und  $A_2$  rufen einmal  $A_3$  auf.

$A_1$ : Laufzeit  $\approx t$ , Orakelanfragen  $q+1$ , Erfolgswahrscheinlichkeit  $=: \varepsilon_1$ .

$A_2$ : Laufzeit  $\approx t$ , Orakelanfragen  $q$ , Erfolgswahrscheinlichkeit  $=: \varepsilon_2$ .

Für  $A_3$  macht es keinen Unterschied, ob er von  $A_1$  oder  $A_2$  aufgerufen wird (in beiden Fällen sind  $k_1, k_2$  zufällig gewählt).

Zufallsquelle von  $A_3$  als Eingabebitstring  $\sigma \in \{0, 1\}^s$  auffassen,  $A_1$  und  $A_2$  unter Einbeziehung der Orakel zu deterministischen Algorithmen  $A'_1, A'_2$  machen, die  $k_1, k_2, \sigma$  als Parameter bekommen.

$A_3$  hat in  $A_1$  und  $A_2$  für gleiches, zufälliges  $k_1, k_2, \sigma$  die gleiche Erfolgswahrscheinlichkeit, liefert gleiche  $m_i, m, z$ . Wenn  $A_3$  für gewählte  $k_1, k_2, \sigma$  Erfolg hat, dann auch entweder  $A'_1$  oder  $A'_2$  für  $k_1, k_2, \sigma$ . Daraus folgt  $\varepsilon = \varepsilon_1 + \varepsilon_2$ .  $\square$

---

14

7. November 2008

---

## HMAC

Als Anwendung des Thm ergeben sich HMACs.

Gegeben eine Hashfunktion  $h : \{0, 1\}^* \rightarrow \{0, 1\}^b$ .

HMAC von Nachricht  $m$  und Schlüssel  $k$ :

- $\text{HMAC} = h(k \parallel \text{opad} \parallel h(k \parallel \text{ipad} \parallel m))$ .
- $\text{HMAC} = h(k \oplus \text{opad} \parallel h(k \oplus \text{ipad} \parallel m))$ .
- $\text{opad} = 36 \cdots 36$ .
- $\text{ipad} = 5C \cdots 5C$ .

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.

---

15

7. November 2008

---

## HMAC

Innere Anwendung von  $h$  im HMAC:

- Benötigt Sicherheit bezüglich Kollisionen bei unbekanntem  $k$ .

Äußere Anwendungen von  $h$  im HMAC:

- Die Länge des Padding 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  $2^{b/2}$  (aber „online“ Angriff).

Relativ gutes Beispiel:

- HMAC mit  $h = \text{SHA-256}$ , MAC-Wert bei Bedarf auf 128 Bit kürzen.

---

16

7. November 2008

---

## Benutzung von MDCs und MACs

Datenintegrität und -authentizität:  $m||MAC_k(m)$ .

Datenintegrität bei authentischem Kanal (Absender bekannt):  $m, h(m)$ .

- $m$  beliebig schicken,  $h(m)$  über integren, authentischen Kanal.
- Beispiel: Cryptohandy, Authentizität durch Stimmerkennung.

Datenintegrität mit Verschlüsselung:  $E_{k_1}(m||MAC_{k_2}(m))$ .

- $k_1$  und  $k_2$  unbedingt unabhängig!
- CBC-MAC und CBC-Verschlüsselung mit  $k_1 = k_2$  und gleichem  $IV$  liefert letzten Chiffretextblock  $E_k(0)$ , weil vorletzter Chiffretextblock =  $E_k(m)$  = letzter Datenblock =  $MAC_k(m)$ . Ist unabhängig von  $m$ !

---

## Sicherer Kanal

Wir nehmen an, ein geheimer 256 Bit Schlüssel  $K$  sei ausgetauscht. Um einen sicheren (= verschlüsselten und authentifizierten) Kommunikationskanal zu erhalten, kann man grob wie folgt vorgehen:

Für jede Kommunikations- und Authentifizierungsrichtung erzeuge man einen eigenen Schlüssel  $K_i$  (also insgesamt 4).

- $K_i \leftarrow \text{SHA-256}(K||\text{„Name der Operation“})$ .

Man verwende AES mit 256 Bit Schlüssellänge.

Man verwende HMAC mit SHA-256, also für 256 Bit Schlüssel  $K_j$ :

$$MAC = \text{SHA-256}((K_j \oplus opad)||\text{SHA-256}((K_j \oplus ipad)||m)).$$

Man verwende eindeutige Nachrichtennummern.

Man verwende den CTR Modus.

---

## Sicherer Kanal

Verschlüsselt wird die Nachricht zusammen mit dem MAC.

Mit Nachrichtennummern Schlüsselstrom für CTR erzeugen.

Nachrichtennummer in Authentifizierung eingehen lassen.

Nachrichtennummer in klar im Chiffretext.

Beim Entschlüsseln MAC checken. Wenn falsch, dann Authentifizierungsfehler.

Wenn Nachrichtennummer schon einmal erhalten wurde (kleiner ist als ein mitgeführter Zähler), dann Nachrichten-Ordnungsfehler.

Schnellere Alternativen, Verschlüsselung und Authentifizierung in einem: OCB (patentgeschützt), CCM.

---

## Paßphrasen und Pseudozufallszahlen

Paßphrasen: Technik, um sich „lange“ Schlüssel zu merken.

- Der Schlüssel wird als Hashwert eines langen Satzes definiert.
- Siehe z.B. pgp, gpg.

Pseudozufallszahlen:

- Die Ausgabe einer Hashfunktion sieht sehr zufällig aus.
- Im Betriebssystem werden regelmäßig Zufallsereignisse angezapft: Maus, Tastatur, Interrupts, Netzwerk, Festplatten etc. Ergebnisse werden mit Hashfunktion zusammengemischt:  $state = h(state, counter++, new input data)$ .
- Extrahieren von Pseudozufallszahlen:  $r = h(state, counter++)$ .

Sehr gefährlich, wenn Pseudozufallszahlen vorhersehbar. Dann Programmablauf deterministisch und Angreifer kann alles nachrechnen. Daher genügend Entropie in Pools sammeln.