Effizienten MAC-Konstruktion aus der Praxis: NMAC

Idee von NMAC:

- Hashe $m \in \{0,1\}^*$ auf einen Hashwert in $\{0,1\}^n$.
- Verwende Π_{MAC3} für Nachrichten fixer Länge auf dem Hashwert.
- Wir konstruieren Π_{MAC3} mittels schlüsselabhängiger Hashfunktion, bei der ein Teil des Hasharguments aus dem Schlüssel besteht.

Algorithmus MAC Π_{MAC3} für Nachrichten fester Länge n

Sei (Gen_h, h) eine Hashfkt $h : \{0,1\}^{2n} \to \{0,1\}^n$.

1. **Gen:** $s \leftarrow Gen_h(1^n)$, s kann öffentlich sein. Wähle $k_1 \in_R \{0,1\}^n$.
2. **Mac:** Bei Eingabe (s, k_1) und $m \in \{0,1\}^n$, berechne
$$t := h_s(k_1||m).$$
3. **Vrfy:** Bei Eingabe (s, k_1) und $(m, t) \in \{0,1\}^n \times \{0,1\}^n$, verifiziere
$$t \stackrel{?}{=} h_s(k_1||m).$$

Achtung: Für Sicherheit von Π_{MAC3} braucht man stärkere Sicherheitseigendschaft als Kollisionsresistenz von (Gen_h, h).

NMAC

Notation: Sei H_s^{IV} eine Merkle-Damgard Hashfunktion, bei der der Initialisierungsvektor auf den Wert IV gesetzt ist.

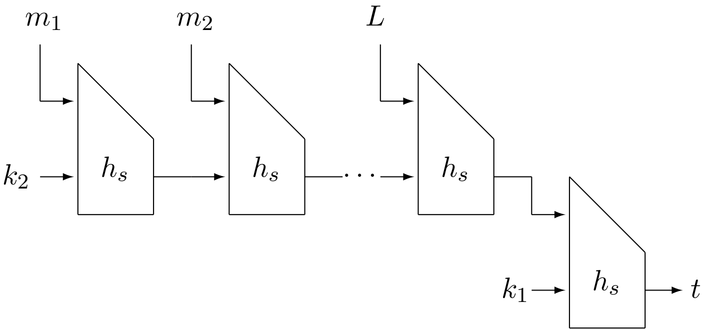
Algorithmus NMAC (Nested MAC)

Sei $\Pi_h = (Gen_h, h)$, $\Pi_{MAC3} = (Gen', Mac', Vrfy')$ wie zuvor. Sei (Gen_h, H) die Merkle-Damgard Transformation von (Gen_h, h).

1. **Gen:** $s \leftarrow Gen_h(1^n)$. Wähle Schlüssel $k_1, k_2 \in_R \{0,1\}^n$.
2. **Mac:** Bei Eingabe (s, k_1, k_2) und $m \in \{0,1\}^n$, berechne
$$t := Mac'_{s,k_1}(H_s^{k_2}(m)) = h_s(k_1 || H_s^{k_2}(m)).$$
3. **Vrfy:** Bei Eingabe (s, k_1, k_2) und $(m, t) \in \{0,1\}^* \times \{0,1\}^n$,
$$\text{Ausgabe} = \begin{cases} 1 & \text{falls } t = Mac_{s,k_1,k_2}(m) \\ 0 & \text{sonst} \end{cases}.$$

Praxis-Variante: Fixiere s, d.h. einzelne Hash-Funktion (z.B. SHA-1).
Anmerkung: Wir können auch $k_2 = 0^n$ setzen. Vorteil von Schlüssel k_2: Sicherheit kann auch unter schwächerer Annahme gezeigt werden.

NMAC



HMAC – Hash-Based MAC

Nachteil von NMAC: Benötigen das Setzen von IV in H.

Idee von HMAC:

- Erzeuge k_1, k_2 durch Vorschalten einer Anwendung von h_s.
- Definieren Konstanten $opad, ipad$ und berechnen

$$k_1 = h_s(IV||k \oplus opad) \text{ und } k_2 = h_s(IV||k \oplus ipad).$$

Algorithmus HMAC

Seien (Gen_H, H) wie zuvor. Seien $opad, ipad, IV \in \{0,1\}^n$ konstant.

1. **Gen:** $s \in Gen_h(1^n)$. Wähle $k \in_R \{0,1\}^n$.
2. **Mac:** Für (s,k) und $m \in \{0,1\}^*$ berechne
$$t = H_s^{k_1}(H_s^{k_2}(m)).$$
3. **Vrfy:** Für (s,k) und $(m,t) \in \{0,1\}^* \times \{0,1\}^n$, verifiziere
$$t \stackrel{?}{=} Mac_{s,k}(m).$$

HMAC ist eine Variante von NMAC

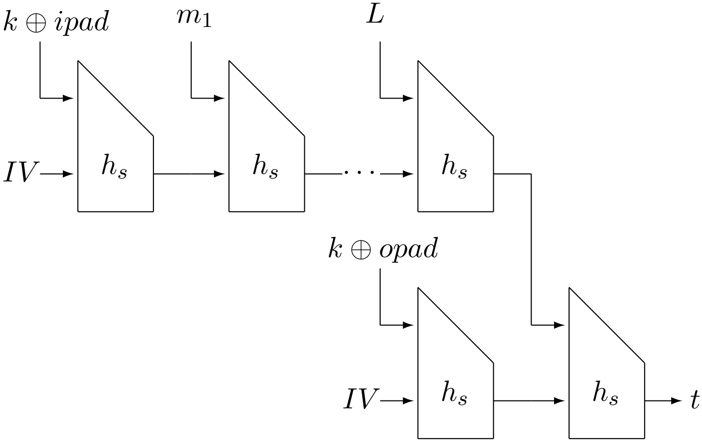
- Wir berechnen beim HMAC den MAC-Wert $H_s^{k_1}(H_s^{k_2}(m))$.
- D.h. die äußere Hashfunktion $H_s^{k_1}$ wird stets auf einen Nachrichtenblock $H_s^{k_2}(m) \in \{0,1\}^n$ fester Länge angewendet.
- Daher ist das Anhängen der Nachrichtenlänge bei $H_s^{k_1}$ unnötig.
- Entspricht der Berechnung von $h_s(k_1 || H_s^{k_2}(m))$, analog zu NMAC.
- D.h. HMAC ist ein Spezialfall von NMAC, wobei k_1 und k_2 aus k mittels Anwendung von h_s abgeleitet werden.
- Wir definieren den folgenden Pseudozufallsgenerator

$$G(k) = \underbrace{h_s(IV || k \oplus opad)}_{k_1} || \underbrace{h_s(IV || k \oplus ipad)}_{k_2}.$$

Korollar Sicherheit von HMAC mittels Sicherheit von NMAC

Sei G ein Pseudozufallsgenerator, (Gen', h) kollisionsresistent und Π_{MAC3} sicher. Dann ist die HMAC-Konstruktion sicher.

HMAC



Praktische Bedeutung von HMAC

Anwendung von HMAC:

- Vorgestellt 1996 von Bellare, Canetti und Krawczyk.
- HMAC ist standardisiert (z.B. IETF RFC, NIST FIPS, ANSI X9.71) und ist weitverbreitet in der Praxis.
- HMAC wird in der Praxis oft in Kombination mit SHA-1 verwendet.
- HMAC findet Anwendung z.B. in den Protokollen Internet Protocol Security (IPSec), Transport Layer Security (TLS), SSH SSL.
- HMAC ist im Vergleich zum CBC-MAC deutlich schneller.
- HMAC wird oft auch als PRF benutzt.

CCA-sichere Verschlüsselung (Encrypt-then-MAC)

Idee:

- Der Verschlüsseler authentisiert c mit Hilfe eines MACs t.
- D.h. nur er ist in der Lage, gültige Paare (c, t) zu erzeugen.
- Damit ist ein CCA-Entschlüsselungsorakel für Angreifer nutzlos.

Algorithmus CCA-sichere Verschlüsselung Π_{cca}

Sei $\Pi_E = (Gen_E, Enc, Dec)$ ein Verschlüsselungsverfahren und $\Pi_M = (Gen_M, Mac, Vrfy)$ ein MAC.

1. **Gen':** $k_1 \leftarrow Gen_E(1^n)$, $k_2 \leftarrow Gen_M(1^n)$.
2. **Enc':** Bei Eingabe von m und (k_1, k_2), berechne $c \leftarrow Enc_{k_1}(m)$ und $t \leftarrow Mac_{k_2}(c)$. Ausgabe des Chiffretextes (c, t).
3. **Dec':** Bei Eingabe von (c, t) und (k_1, k_2),

$$\text{Ausgabe} = \begin{cases} m := Dec_{k_1}(c) & \text{falls } Vrfy_{k_2}(c, t) = 1 \\ \perp & \text{sonst} \end{cases}.$$

Eindeutige Tags

Definition MAC mit eindeutigem Tag

Sei $\Pi_M = (Gen, Mac, Vrfy)$ ein MAC. Π_M besitzt *eindeutige Tags* falls für alle k, m genau ein $t \in \{0,1\}^*$ existiert mit $Vrfy_k(m,t) = 1$.

Anmerkungen:

- D.h. der *Mac*-Algorithmus ist deterministisch. (Nicht hinreichend!)
- Bsp: Π_{MAC}, CBC-MAC, NMAC, HMAC besitzen eindeutige Tags.
- Π_{MAC2} besitzt keine eindeutigen Tags.

Π_{cca} ist CCA-sicher

Satz CCA-Sicherheit von Π_{cca}

Sei Π_E CPA-sicher und Π_M ein sicherer MAC mit eindeutigen Tags. Dann ist Π_{cca} CCA-sicher.

Beweisskizze:

- Offenbar ist Π_{cca} sicher gegenüber CPA-Angreifern $\mathcal{A}$.
- Wir zeigen nun, dass ein $Dec(\cdot)$-Orakel für $\mathcal{A}$ nutzlos ist.
- Sei (c, t) eine Anfrage von $\mathcal{A}$ an $Dec(\cdot)$.

Fall 1: (c, t) kommt aus voriger $Enc(m)$-Anfrage von $\mathcal{A}$.

- Dann weiss $\mathcal{A}$ bereits, dass $Dec(c, t)$ die Antwort m liefert.
- D.h. das Entschlüsselsorakel liefert keine nützliche Information.

Fall 2: (c, t) kommt nicht aus $Enc(m)$-Anfrage.

- Falls $Vrfy_{k_2}(c, t) = 1$, hat $\mathcal{A}$ einen gültigen Tag t für ein neues c konstruiert (folgt aus der Eindeutigkeit der Tags).
- Aufgrund der MAC-Sicherheit geschieht dies mit Ws $\leq \mathrm{negl}(n)$.
- D.h. $Dec(\cdot)$ gibt mit Ws $\geq 1 - \mathrm{negl}(n)$ die nutzlose Ausgabe $\perp$.

Andere Varianten

Alternative Ideen für CCA-sichere Verschlüsselung

- Encrypt-then-Mac: $c \leftarrow Enc_{k_1}(m)$, $t \leftarrow Mac_{k_2}(c)$. $CT = (c, t)$
  CCA sicher. Von SSL benutzt.
- Mac-then-Encrypt: $t \leftarrow Mac_{k_2}(m)$, $c \leftarrow Enc_{k_1}(m||t)$. $CT = c$.
  i.A. nicht CCA sicher! Von IPsec benutzt.
- Encrypt-and-Mac: $c \leftarrow Enc_{k_1}(m)$, $t \leftarrow Mac_{k_2}(m)$. $CT = (c, t)$
  i.A. nicht CCA sicher! Von SSH benutzt.