Message Authentication Codes

Problem setup

Alice

Bob

Message Authentication Codes (MACs)

Definition 1

Let $\mathcal{M}$ be a message space, $\mathcal{H}$ a hash (or tag) space; and $\mathcal{K}$ a key space. A MAC over $(\mathcal{K}, \mathcal{M}, \mathcal{H})$ is a pair $MAC = (S, V)$, where:

- $S : \mathcal{K} \times \mathcal{M} \to \mathcal{D}(\mathcal{H})$ is a (possibly randomized) signing algorithm; and
- $V : \mathcal{K} \times \mathcal{M} \times \mathcal{H} \to \{true, false\}$ is a deterministic verification algorithm.

We require that for all $k \in \mathcal{K}$, $m \in \mathcal{M}$ the equality

$$V(k, m, S(k, m)) = true$$

holds with probability one.

Secure MACs

An existential forgery attack game between the challenger and the adversary $\mathcal{A}$ proceeds as follows:

- The challenger samples a key k from $\mathcal{K}$ uniformly at random.
- The adversary selects a number of rounds N for which the game will be played.
- In each round i:
  - The adversary computes a message $m_i \in \mathcal{M}$ and sends it to the challenger.
  - The challenger computes $t_i = S(k, m_i)$ and send t_i to the adversary.

  After the final round, the adversary computes a tuple $(m_{N+1}, t_{N+1}) \in \mathcal{M} \times \mathcal{H}$ s.t. $(m_{N+1}, t_{N+1}) \notin \{(m_1, t_1), \ldots, (m_N, t_N)\}$. The adversary wins the game if $V(k, m_{N+1}, t_{N+1}) = true$.

The advantage of $\mathcal{A}$ against MAC $MAC = (S, V)$ is the quantity

$$\mathcal{ADV}_{\mathcal{EF}}(MAC, \mathcal{A}) = \mathbb{P}(\mathcal{A} \text{ wins the e.f. game}).$$

A MAC is ε-secure if $\mathcal{ADV}_{\mathcal{EF}}(MAC, \mathcal{A}) \leq \varepsilon$ for every efficient adversary $\mathcal{A}$.

HMAC: MAC from (Unkeyed) Hash Function h

Idea: tag = hash of a string constructed from the k and m in some deterministic way.

HMAC: MAC from (Unkeyed) Hash Function h

Idea: tag = hash of a string constructed from the k and m in some deterministic way.

- $S(k, m) = h(k \,||\, m)$: not generally secure, in particular if h is constructed via Merkle-Damgård paradigm, where it is vulnerable to a length-extension attack (LEA):

HMAC: MAC from (Unkeyed) Hash Function h

Idea: tag = hash of a string constructed from the k and m in some deterministic way.

- $S(k, m) = h(k \,||\, m)$: not generally secure, in particular if h is constructed via Merkle-Damgård paradigm, where it is vulnerable to a length-extension attack (LEA):
  - secure under sponge paradigm – KMAC (Keccak-based MAC)

HMAC: MAC from (Unkeyed) Hash Function h

Idea: tag = hash of a string constructed from the k and m in some deterministic way.

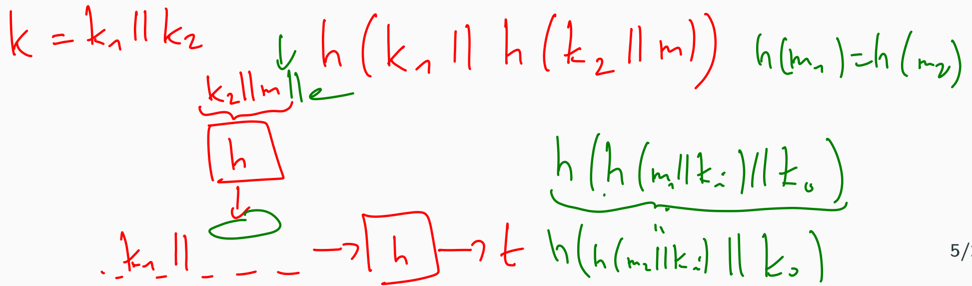
- $S(k, m) = h(k \,||\, m)$: not generally secure, in particular if h is constructed via Merkle-Damgård paradigm, where it is vulnerable to a length-extension attack (LEA):
  - secure under sponge paradigm – KMAC (Keccak-based MAC)
- $S(k, m) = h(m \,||\, k)$: also has issues; in particular vulnerable if h has known collisions.

HMAC: MAC from (Unkeyed) Hash Function h

Idea: tag = hash of a string constructed from the k and m in some deterministic way.

- $S(k, m) = h(k \,||\, m)$: not generally secure, in particular if h is constructed via Merkle-Damgård paradigm, where it is vulnerable to a length-extension attack (LEA):
  - secure under sponge paradigm – KMAC (Keccak-based MAC)
- $S(k, m) = h(m \,||\, k)$: also has issues; in particular vulnerable if h has known collisions.

Instead, the HMAC standard uses a two-key nested construction. (See next slide.)



HMAC standard

HMAC uses a hash function h based on Merkle-Damgård (typically some version of SHA-256). Let j be the block length of the underlying compression function f. It is assumed that h's output size is $\leq j$.

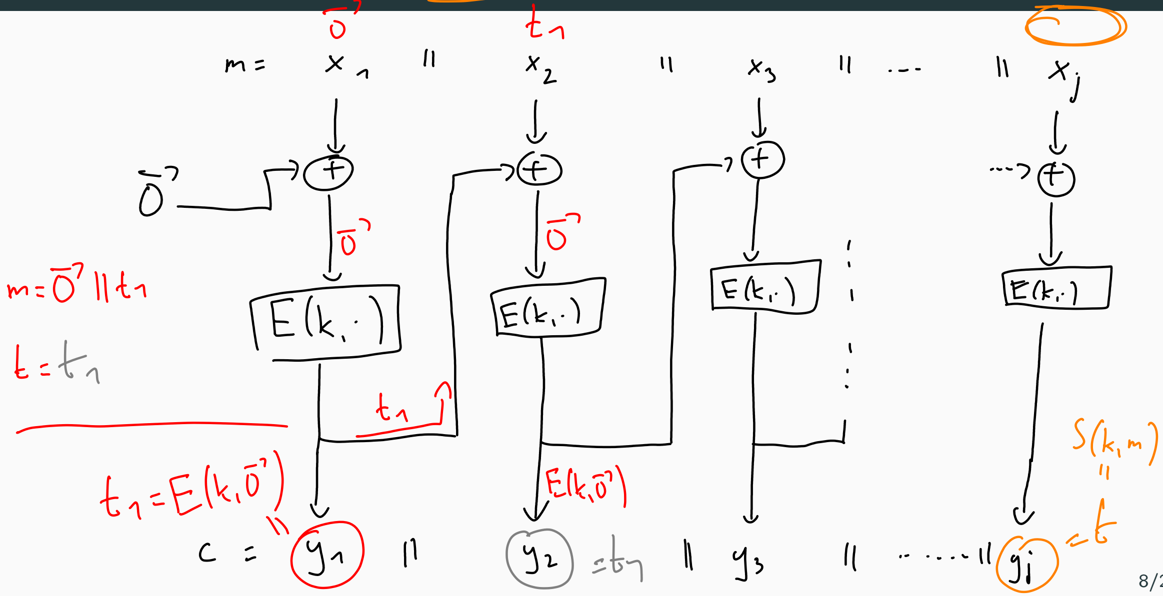
Algorithm 1: HMAC based on h.

Input: $k \in \mathcal{K}, m \in \mathcal{M}$
Output: $S_{HMAC}(k, m)$
$j \leftarrow$ message block length of h's compression function f;
if $len(k) > j$ **then** $k \leftarrow h(k)$;
if $len(k) < j$ **then** pad k with zero bytes to length j;
$ipad \leftarrow$ `0x5c` byte repeated to match the key length;
$opad \leftarrow$ `0x36` byte repeated to match the key length;
$k_i = k \oplus ipad$; $k_o = k \oplus opad$;
return $h(k_o \,||\, h(k_i \,||\, m))$

Notes on HMAC

- Frequently used in the internet environment (SSH, TLS, IPSec,...)
- There is a security proof for HMAC w.r.t. the existential forgery property:
  - The proof rests on the assumption that the underlying compression function is computationally indistinguishible from a pseudorandom function even under related key attacks.
  - The proof does not need to assume that the compression function is collision resistant! Hence, even e.g. HMAC based on MD5 is considered to be secure w.r.t. existential forgery, though not recommended for use in newly designed protocols.
- Deterministic - verifier just runs the signing algorithm with his key k and message m and compares the two tags.

MACs from Block Ciphers: Vanilla CBC



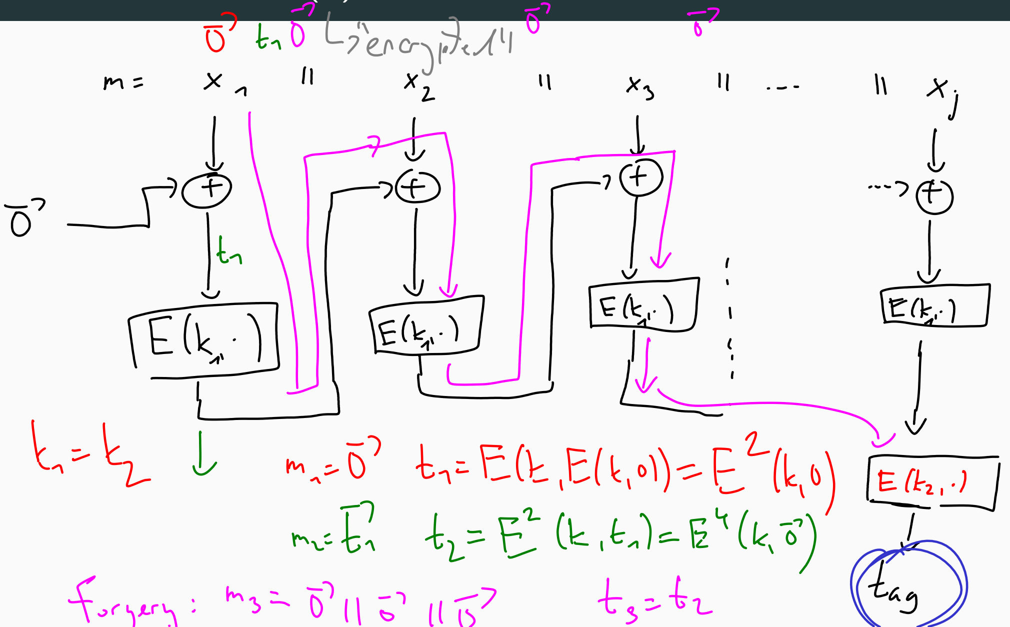
Partial security of Vanilla CBC-MAC

An adversary $\mathcal{A}$ in the existential forgery attack game is prefix-free if no $\mathcal{A}$'s query m_i is a proper prefix of any other query m_j, $j \in \{1, \ldots, N+1\} \setminus \{i\}$ (i.e., including the final guess) in the same attack game.

Theorem 1

If the underlying block cipher is ε-secure for negligible ε, then the Vanilla CBC-MAC is δ-secure for negligible δ *provided that we restrict the attack game to prefix-free adversaries.*

Secure MAC from CBC: (E)CBC-MAC



if $t_1 = t_2$

$m_1 = \vec{0}$ $t_1 = E(k, E(k, 0)) = E^2(k, 0)$

$m_2 = t_1$ $t_2 = E^2(k, t_1) = E^4(k, \vec{0})$

Forgery: $m_3 = \vec{0} \| \vec{0} \| \vec{0}$ $t_3 = t_2$

ECBC-MAC: pseudocode

Algorithm 2: ECBC-MAC based on a block cipher with encryption function E and with block length ℓ.

Input: $(k_1, k_2) \in \mathcal{K}^2, m \in \mathcal{M}$.
Output: $S_{ECBC\text{-}MAC}(k, m)$
if *len*(m) *is not a multiple of* ℓ **then**
  $m \leftarrow pad(m)$
let $m = x_1 \,||\, \cdots \,||\, x_j$ with each $x_i \in \{0,1\}^\ell$;
$y_1 \leftarrow E(k_1, x_1)$;
for $i \in \{2, \ldots, j\}$ **do**
  $y_i \leftarrow E(k_1, y_{i-1} \oplus x_i)$
return $E(k_2, y_j)$

Deterministic: the same procedure can be used for verification.

The padding function must be injective!

Security of ECBC-MAC

Theorem 2

Let E be an encryption function of an ε-secure block cipher with block space $\mathcal{X}$. Assume that each message consists of at most n blocks. Then, when restricting adversaries that play the attack game for at most N rounds, the ECBC-MAC based on E (with an injective padding function) is δ-secure for

$$\delta = 2\varepsilon + \frac{(N(n+1))^2 + 2N^2}{2|\mathcal{X}|}.$$

$$\approx (N \cdot n)^2 \approx \sqrt{|\mathcal{X}|}$$

Security proof outline for ECBC-MAC

The proof consists of three conceptual steps:

- First, prove that the vanilla CBC-MAC is a secure universal hash function (UHF). A MAC is an UHF if any efficient adversary wins the following one-round game with negligible probability:
  - The challenger randomly samples k from $\mathcal{K}$.
  - The adversary computes two messages, m, m'. He wins the game if $S(k, m) = S(k, m')$.

Security proof outline for ECBC-MAC

The proof consists of three conceptual steps:

- First, prove that the vanilla CBC-MAC is a secure universal hash function (UHF). A MAC is an UHF if any efficient adversary wins the following one-round game with negligible probability:
  - The challenger randomly samples k from $\mathcal{K}$.
  - The adversary computes two messages, m, m'. He wins the game if $S(k, m) = S(k, m')$.
- Then, prove that a composition of a secure pseudo-random permutation with a secure UHF (such as the ECBC-MAC) is computationally indistinguishible from a random function $\mathcal{K} \times \mathcal{M} \to \mathcal{H}$.
- A random function = MAC secure against all adversaries. Prove that something indistinguishable from a random function is a secure MAC against efficient adversaries.

Optimality of security bounds

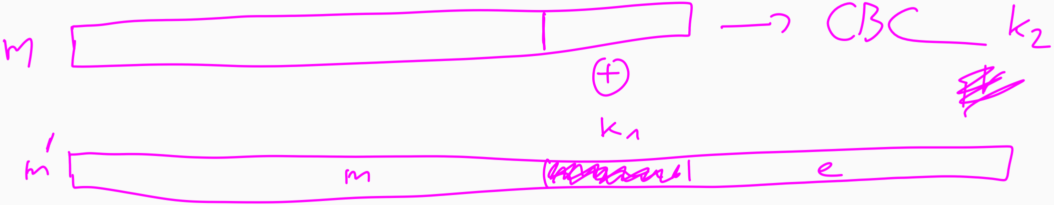
ECBC-MAC has the property that if $S_{ECBC\text{-}MAC}(k, m) = S_{ECBC\text{-}MAC}(k, m')$, and both m, m' have lengths equal to multiples of block length, then for any x it holds $S_{ECBC\text{-}MAC}(k, m \,||\, x) = S_{ECBC\text{-}MAC}(k, m' \,||\, x)$.

Hence, a forgery can be trivially constructed once such a collision in the ECBC-MAC is found. This can be done by the birthday attack in time $\mathcal{O}(\sqrt{|\mathcal{X}|})$.

Notes on CBC-MAC

- standardized by ANSI, used frequently in banking and retail
- padding: typically append 10^*, creating a new block if the original message has length = multiple of block size
- there is a NIST-standardized variant CMAC (see next slide), with the following features:
  - uses a single key
  - no need to add new padding block when processing block-aligned messages

CMAC: Idea



Vanilla CBC-MAC was secure against all prefix-free adversaries.

Idea behind CMAC: before signing each message, modify it so as to make the probability of one message being a prefix of another negligible.

Prefix-free encoding: preliminary idea through randomization

Let E be the encryption function used in the CBC chain, ℓ its block size. For a message m we compute its modification *p-free-r*(m) as follows:

- if *len*(m) is not a multiple ℓ, pad the last block with 10^*, otherwise keep it as is;
- sample a block $r \in \{0,1\}^\ell$ uniformly at random and modify m by xor'ing its last block with r; output the modified message as *p-free-r*(m)

Then the probability that for two given messages m, m' the string *p-free-r*(m) is a prefix of *p-free-r*(m') or vice versa is $\approxeq \frac{1}{2^\ell}$.

CMAC: pseudocode

Algorithm 3: CMAC built over a block-cipher encryption algorithm E with block length ℓ

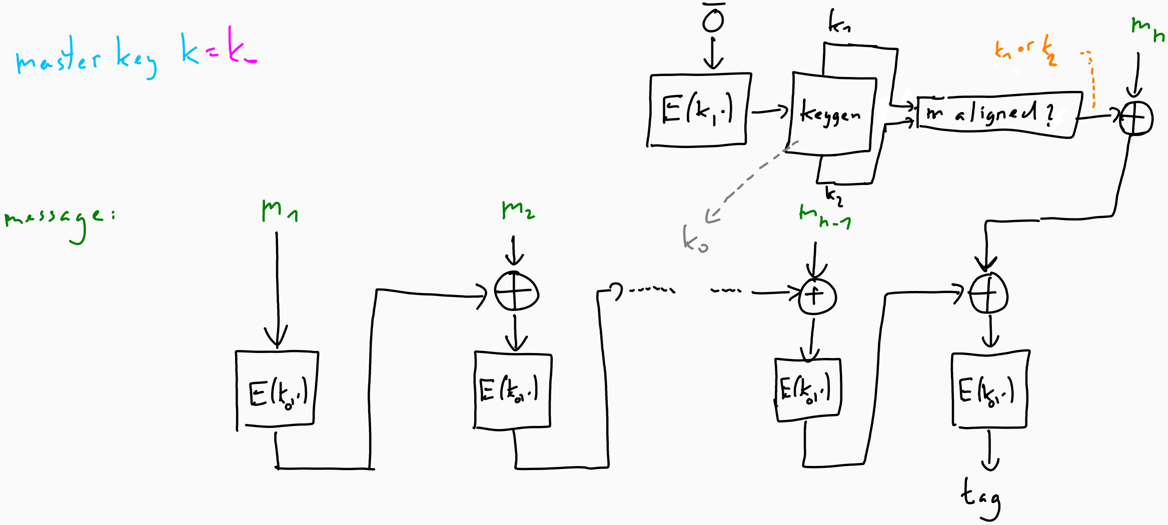
Input: $k \in \mathcal{K}, m \in \mathcal{M}$

Output: $S_{CMAC}(k, m)$

```
(k_0, k_1, k_2) ← CMAC-Keygen(k);
if len(m) is a multiple of ℓ then
    m ← m with last block xor'ed with k_1
else
    m ← m || 10* ;                          // Pad to nearest multiple of ℓ.
    m ← m with last block xor'ed with k_2
return S_vanilla-CBC-MAC(k_0, m)
```

Comes also with a security theorem!

CMAC: picture



Prefix-free encoding in standardized CMAC

Let E be the encryption function used by the CBC chain and ℓ its block size. We use the signing key k to derive three sub-keys k_0, k_1, k_2 where $k_0 = k$ and k_1, k_2 are computed as follows:

Algorithm 4: Sub-key generation algorithm for CMAC with block lenth $\ell = 128$.

Input: key k
Output: sub-keys k_0, k_1, k_2
function `CMAC-Keygen(`k`)`
- $M \leftarrow E(k, 0^\ell)$;
- **if** *most sign. bit of M is 0* **then** $k_1 \leftarrow SHIFTL(M, 1)$;
- **else** $k_1 \leftarrow SHIFTL(M, 1) \oplus 0^{\ell-8}10000111$;
- **if** *most sign. bit of* k_1 *is 0* **then** $k_2 \leftarrow SHIFTL(k_1, 1)$;
- **else** $k_2 \leftarrow SHIFTL(k_1, 1) \oplus 0^{\ell-8}10000111$;

Towards alternative constructions: One-Time Mac

Let (S, V) be a MAC over $(\mathcal{K}, \mathcal{M}, \mathcal{H})$. A one-time existential forgery attack game between the challenger and the adversary $\mathcal{A}$ proceeds as follows:

- the challenger samples $k \in \mathcal{K}$ uniformly at random;
- $\mathcal{A}$ computes a message m and sends it to the challenger;
- the challenger computes $S(k, m)$ and sends it to the adversary

Then, the adversary computes a message $m' \neq m$ and tag $t \in \mathcal{H}$. He wins the game if $V(k, m', t) = true$.

The one-time advantage of $\mathcal{A}$ against MAC $MAC = (S, V)$ is the quantity

$$\mathcal{ADV}_{ot\mathcal{EF}}(MAC, \mathcal{A}) = \mathbb{P}(\mathcal{A} \text{ wins the one-time e.f. game}).$$

A MAC is ε-one-time secure if $\mathcal{ADV}_{ot\mathcal{EF}}(MAC, \mathcal{A}) \leq \varepsilon$ for every efficient adversary $\mathcal{A}$.

OMAC:

Let ℓ be a block size and p a prime number larger than 2^ℓ. Consider the following MAC:

- $\mathcal{H} = \{0, 1, \ldots, p-1\} = \mathbb{Z}_p$, $\mathcal{K} = \mathbb{Z}_p^2$; $k=(a,b)$
- for

$$m = m_1 \,||\, m_2 \,||\, \cdots \,||\, m_j$$

MAC(17 || 38) using (a,b), $m_1 = 17$, $m_2 = 38$

$P_m(y) = y^3 + 38 \cdot y^2 + 17 \cdot y$

we construct a polynomial

$P_1 = y^2 + y$

$$P_m(y) = y^{j+1} + m_j \cdot y^j + m_{j-1} \cdot y^{j-1} + \cdots + m_1 \cdot y$$

and put

$$S((a, b), m) = P_m(a) + b \pmod{p}.$$

$a^3 + 38 \cdot a^2 + 17 \cdot a + b$

$m_1 = 0 \quad P_0 = y^2 + 0 \cdot y = y^2 \quad a^2$

round 1: 0 .. $t_1 = a^2 + b$

Forge: $m = 0 || 0 || 0 \quad P_{0||0||0} = y^4 \quad t = a^4$

round 2: $m_2 = 1 \quad t_2 = a^2 + a + b$

Carter-Wegman (CW) MACs: From one-time to many-time MACs

Let:

- $\mathcal{E} = (E, D)$ be a secure block cipher over $(\mathcal{K}_1, \mathcal{H})$
- $MAC_{ot} = (S_{ot}, V_{ot})$ be a secure one-time MAC over $(\mathcal{K}_2, \mathcal{M}, \mathcal{H})$; and

A CW-style MAC built from MAC_{ot} and $\mathcal{E}$ is a MAC $MAC = (S, V)$ over $(\mathcal{K}_1 \times \mathcal{K}_2, \mathcal{M}, \mathcal{H}^2)$ defined as follows:

- $S((k_1, k_2), m)$ is produced randomly by:
  - first, sampling a block $r \in \mathcal{H}$ uniformly at random
  - then, computing $S((k_1, k_2), m) = (r, E(k_1, r) \oplus S_{ot}(k_2, m))$
- $V((k_1, k_2), m, (r, t))$ returns *true* if and only if

one-time MAC

$u = E(k_1, r)$ $\quad u \oplus t \rightarrow V_{ot}$

Carter-Wegman (CW) MACs: From one-time to many-time MACs

Let:

- $\mathcal{E} = (E, D)$ be a secure block cipher over $(\mathcal{K}_1, \mathcal{H})$
- $MAC_{ot} = (S_{ot}, V_{ot})$ be a secure one-time MAC over $(\mathcal{K}_2, \mathcal{M}, \mathcal{H})$; and

A CW-style MAC built from MAC_{ot} and $\mathcal{E}$ is a MAC $MAC = (S, V)$ over $(\mathcal{K}_1 \times \mathcal{K}_2, \mathcal{M}, \mathcal{H}^2)$ defined as follows:

- $S((k_1, k_2), m)$ is produced randomly by:
  - first, sampling a block $r \in \mathcal{H}$ uniformly at random
  - then, computing $S((k_1, k_2), m) = (r, E(k_1, r) \oplus S_{ot}(k_2, m))$
- $V((k_1, k_2), m, (r, t))$ returns *true* if and only if $V_{ot}(k_2, m, (E(k_1, r) \oplus t)) = true$