CRYE 6127 Introduction to Cryptology

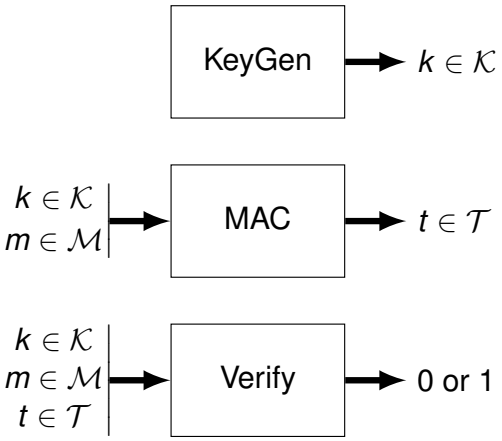
Jorge L. Villar

UBa Cyber Crypto Center, Fall 2025

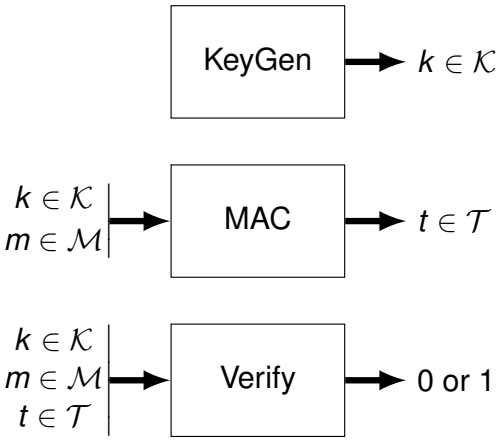
Message Authentication Codes and Hashing

Outline

Message Authentication Codes: Syntax

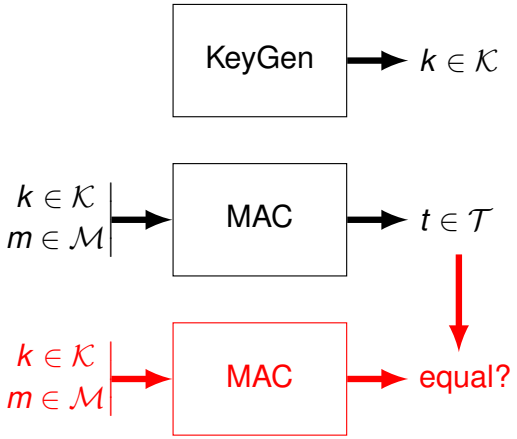


Message Authentication Codes: Correctness



$$\forall m \in \mathcal{M},\ \forall k \in \mathcal{K}, \quad \mathsf{Verify}(k, m, \mathsf{MAC}(k, m)) = 1$$

Typical Verification Procedure



$$\text{Verify}(k, m, t) = 1 \;\Leftrightarrow\; t = \text{MAC}(k, m)$$

MAC: Perfect Unforgeability

Informal definition:
"Impossible to find a new pair (m', t') from (m, t) without k"

Definition (Perfect One-Time Unforgeability)

For a uniformly distributed $K \in \mathcal{K}$ and for any different $m, m' \in \mathcal{M}$, **the tags** $\text{MAC}(K, m)$ **and** $\text{MAC}(K, m')$ **are independent random variables.**

Definition (Perfect n-Times Unforgeability)

For a uniformly distributed $K \in \mathcal{K}$ and for any different $m_1, m_2, \ldots, m_n, m' \in \mathcal{M}$, **the** $n+1$ **tags** $T_1 = \text{MAC}(K, m_1), \ldots, T_n = \text{MAC}(K, m_n)$ **and** $T' = \text{MAC}(K, m')$ **are independent random variables.**

One-Time Perfect MAC

q is a ℓ-bit long prime, $\mathcal{M} = \mathcal{T} = \mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ and $\mathcal{K} = \mathbb{Z}_q \times \mathbb{Z}_q$.

Construction

Public Parameters(ℓ) :
Choose a ℓ-bit long prime q;
output $pp = (q)$;

KeyGen(pp) :
Parse $pp = (q)$;
Choose random $a, b \leftarrow \mathbb{Z}_q$;
output $k = (a, b)$;

MAC(k, m) :
Parse $k = (a, b)$;
output $t = am + b \mod q$;

Unforgeability (one time): Even knowing a valid pair $(m, t = am + b)$, still $t' = am' + b$ remains random for any $m' \neq m$.

n-Times Perfect MAC

Now $\mathcal{K} = \mathbb{Z}_q^{n+1} = \underbrace{\mathbb{Z}_q \times \cdots \times \mathbb{Z}_q}_{n+1 \text{ copies}}$.

Construction

Public Parameters(ℓ) : (as before)

KeyGen(pp) :
Choose random $a_1, \ldots, a_n, b \leftarrow \mathbb{Z}_q$;
output $k = (a_1, \ldots, a_n, b)$;

MAC(k, m) :
output $t = P(m) = a_n m^n + \ldots + a_1 m + b \mod q$;

Unforgeability (n times): Even knowing n valid pairs $(m_1, t_1 = P(m_1)), \ldots, (m_n, t_n = P(m_n))$, still $t' = P(m')$ remains random for any $m' \notin \{m_1, \ldots, m_n\}$. (By Lagrange polynomial interpolation.)

Practical MAC Constructions

Perfect MACs are not practical:

- The key is larger than the message.
- Even in the n-times secure MAC, the key is larger than the size of the n messages together.
- The tag has the same size as the message.

Known efficient (non-perfect) constructions are based on

- hash functions (see next topic)
- block ciphers
- ...

Block Cipher Based MAC

CBC-MAC is a MAC based on any block cipher in CBC mode.

- A short long-term key is used to compute many tags.
- The tag length is independent of the message length.
- The computational complexity is similar to encrypting the message.

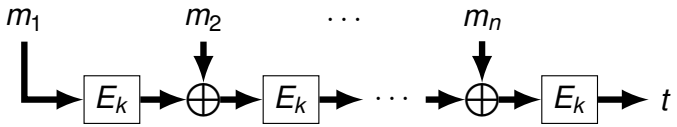
Construction

Public Parameters(ℓ) :
Choose a block cipher with message block size ℓ and a padding mechanism;

KeyGen(pp) :
Generate a random key k for the block cipher;
output k;

MAC(k, m) :
Encrypt (padded) m with key k using CBC mode and taking $iv = 0$;
output the last block of the encryption;

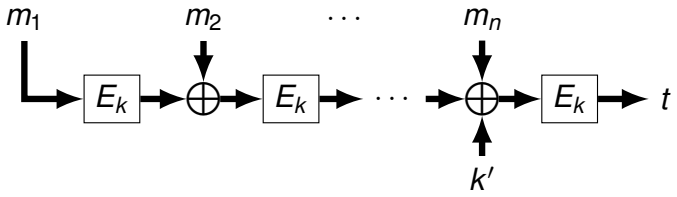
CBC-MAC Diagram



Unforgeability:

- CBC-MAC for messages of a fixed length (exact multiple of the block size) is **proven** to be as secure as the block cipher in CBC mode.
  Any attack against CBC-MAC implies a similar attack against the encryption scheme.
- A known (concatenation) attack exists against CBC-MAC for any block cipher.
  $t = \mathrm{MAC}(k, m_1, \ldots, m_n) = \mathrm{MAC}(k, m_1, \ldots, m_n, m_1 \oplus t, m_2 \ldots, m_n)$

One-Key MAC



CMAC solves the variable length security problem of CBC-MAC without increasing the size of the key.

- The last message block is "tweaked" with a key k' derived from k.
- Two different values of k' are generated: one is used when the last block requires to be padded, and the other when no padding is necessary.

Outline

Hash Functions

A map $H : \{0,1\}^* \to \{0,1\}^\ell$ behaving like a **random function**.

Definition (Preimage Resistance)

Given a random $y \in \{0,1\}^\ell$, it is infeasible to compute $x \in \{0,1\}^*$ such that $H(x) = y$.

Definition (Second Preimage Resistance)

Given a random $x_1 \in \{0,1\}^*$, it is infeasible to compute a different $x_2 \in \{0,1\}^*$ such that $H(x_1) = H(x_2)$.

Definition (Collision Resistance)

It is infeasible to compute different $x_1, x_2 \in \{0,1\}^*$ such that $H(x_1) = H(x_2)$.

Ideal Behavior

Definition (Random Function)

For any different $x_1, x_2, \ldots \in \{0,1\}^*$, $H(x_1), H(x_2), \ldots$ are independent uniform random variables.

Preimage Resistance: Expected 2^ℓ hash computations to break it.

Second Preimage Resistance: Expected 2^ℓ hash computations to break it.

Collision Resistance: Expected $2^{\ell/2}$ hash computations to break it.

Birthday paradox: The probability to find a collision in n random values in $\{0,1\}^\ell$ is approx. $n^2 2^{-(\ell+1)}$.

Random Oracle Model

Random Oracle Model (ROM): The security of a cryptosystem is analyzed by replacing a Hash function by a truly random function available to all parties.

No real random function (available to all parties) exists.

The random function is simulated with a dynamically generated random table $(x_i, y_i = H(x_i))$.

The security proofs in the ROM are just heuristic.

There are known weak systems that are secure in the ROM.

Practical Constructions

Hash Functions are not Symmetric Key objects, but the tools used to build them are very similar:

- Diffusion
- Iteration
- (No key expansion)

Construction strategies:

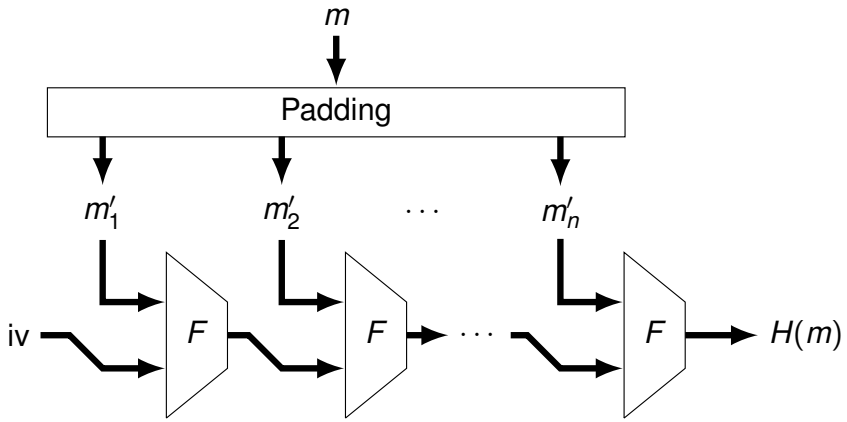
- Merkle-Damgård (e.g. MD5, SHA-1, SHA-2)
- Sponge (e.g. SHA-3)

Merkle-Damgård Construction

Generic construction of H from a compression function $F : \{0,1\}^r \times \{0,1\}^\ell \to \{0,1\}^\ell$.

- The message m is padded to a length multiple of r.
- The padded message m' is split into r-bit blocks $(m'_1, \ldots, m'_n)$.
- A IV is fixed by design.
- The iteration is
  $s_0 = IV$
  $s_i = F(m'_i, s_{i-1})$.
- $H(m)$ is set to $G(s_n)$, for some finalization function G.

Merkle-Damgård Construction



Merkle-Damgård Security

The padding scheme must fulfil:

- m is a prefix of its padded version m'.
- If x and y have equal lengths, then so do x' and y'.
- If x and y have different lengths, then the last blocks in x' and y' are different.

E.g., length-padding: $m' = m \| 10 \ldots 0 \| d$, where d is a 64-bit representation of the bit-length of m.

Proposition

If F is collision resistant then H is also collision resistant.

MD5

Message block size: $r = 512$ bits

Hash result and internal state size: $\ell = 128$ bits

Number of rounds: 64

Round structure:

- Split the internal state into (A, B, C, D).
- $A = A + G_i(B, C, D) + M_i + K_i \mod 2^{32}$.
- $A = \text{rotl}(A, s_i) + B \mod 2^{32}$.
- Cyclically permute (A, B, C, D) one position to the left.

The resulting internal state after processing a data block is added to the original one (the state before starting the iteration).

No finalization function is applied (the hash value is just the last internal state).

SHA-1 and SHA-2

Similar to MD5, but with improved security.

- SHA-1 uses $\ell = 160$, the internal state is split in 5 words (A, B, C, D, E) in a bit more complex round structure.
- SHA-2 family uses a larger ℓ (224, 256, 384 and 512), and a similar round design (with more complex nonlinear functions).

Known attacks:

- MD5: Easy to find lots of collisions.
- SHA-1: Some collisions known but still considered as second preimage resistant.
- SHA-2: No practical attack known to date.

Sponge Construction

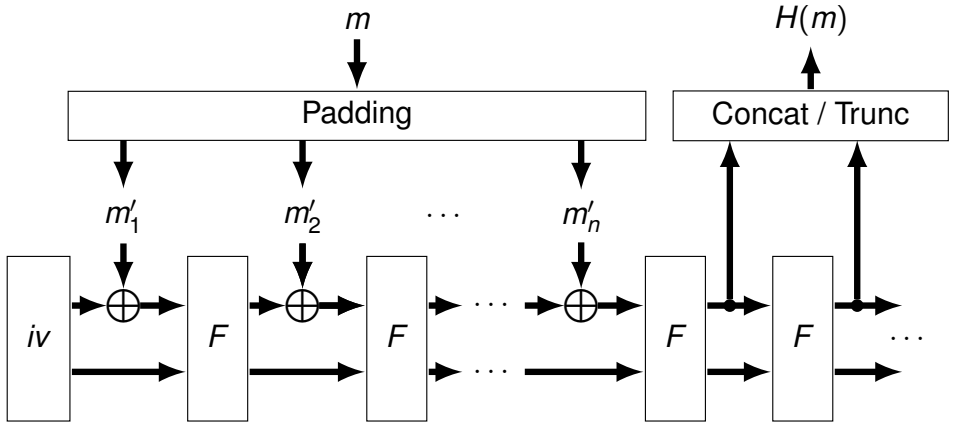
Design elements:

- Asymmetric accumulator (R, C) as internal state.
- Stirring function $F : \{0,1\}^{r+c} \to \{0,1\}^{r+c}$.
- Only the R part interacts with input/output values.

Operation:

- The message m is padded to a length multiple of r.
- The padded message is split into r-bit blocks $(m'_1, \ldots, m'_n)$.
- An IV is fixed by design: $(R_0, C_0) = IV$.
- The "input" iteration is: $(R_i, C_i) = F(R_{i-1} \oplus m'_i, C_{i-1})$
- The "output iteration" is: $(R_{n+j}, C_{n+j}) = F(R_{n+j-1}, C_{n+j-1})$
- The output value is $H(m) = R_n \| \cdots \| R_{n+u}$, $u = \lfloor \ell / r \rfloor$
- The last block can be truncated to obtain the desired ℓ.

Sponge Construction



SHA-3

Sponge construction with $r + c = 1600$ bits

Sizes: $u = 0$, $c = 2\ell$ and $r = 1600 - 2\ell$

Number of rounds: 24

Round structure:

- (R, C) is organized as a 5×5 64-bit words matrix.
- XOR elements with the XOR of adjacent columns.
- Apply different bit-rotations to the 25 words.
- Permute the 25 words.
- XOR rows with a nonlinear function of the other rows.
- XOR one of the 25 words with the output of an LFSR.

Different members in the family: $\ell = 224, 256, 384, 512$

Outline

HMAC

$H(m)$ is a digest of message m (no key involved).

Turn it into a MAC: Append the key and the message
$HMAC(k, m) = H(k\|m)$

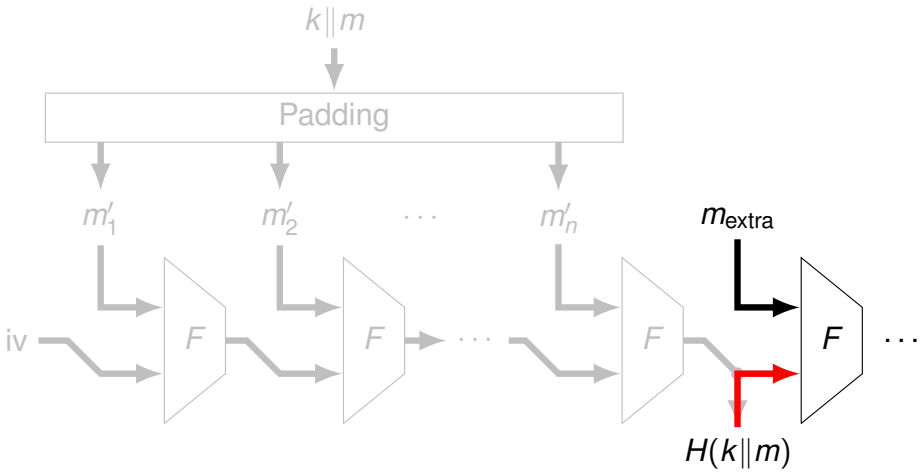
Insecure for some Merkle-Damgård based hashes!
(e.g. MD5, SHA1, SHA2)

Fix: Use two passes
$HMAC(k, m) = H((k' \oplus P_{\text{out}})\|H((k' \oplus P_{\text{in}})\|m))$

If k is longer than a hash block, then $k' = H(k)$.
Otherwise, $k' = k$, or $k' = k\|0\cdots0$.

Not necessary for SHA-3:
$HMAC\text{-}SHA3(k, m) = SHA\text{-}3(k\|m)$

Merkle-Damgård Prolongation Attack



Hash Puzzles

Hash preimage resistance $\longrightarrow$ Problem set with **tunable** hardness.

Problem (Hash Puzzle)

Given t, find $x \in \{0,1\}^*$ such that $H(x)$ has at least t leading zeros.

- The expected number of hash computations is 2^t.
- No possible speed-up.
- No possible precomputation if x must have a given fresh prefix.
- Applications as proof-of-work in antispam tools or blockchain.

Merkle Trees

Efficient hashing of a collection of documents.

- Documents are the leaves of a binary tree.
- The value of each node in the tree is the hash of its childrens' values.
- The value of the root is the hash of the collection.
- Efficient proof that a document is part of the collection.
- Efficient document update.
- Efficient addition of new documents.

Merkle Trees

Root hash:
$h_\emptyset = H(1 \| h_0 \| h_1)$

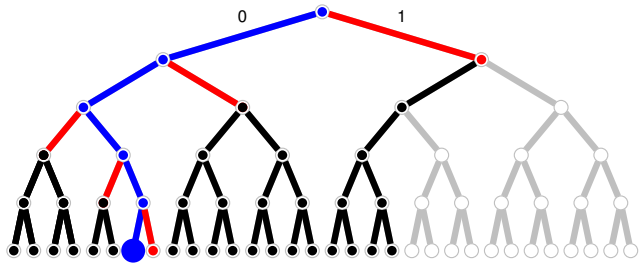
Intermediate node hash:
$h_x = H(1 \| h_{x0} \| h_{x1})$, if node $x1$ exists.
$h_x = H(1 \| h_{x0})$, otherwise

Leave:
$h_x = H(0 \| m_x)$

Example of **proof of membership** for $m = m_{00110}$
$\pi_{00110} = (h_1, h_{01}, h_{000}, h_{0010}, h_{00111})$

If some sibling does not exist the hash is replaced by $\perp$.

Example



Example of a proof of membership for document m_{00110}, in an incomplete binary tree.

$\pi_{00110} = (h_1, h_{01}, h_{000}, h_{0010}, h_{00111})$

CRYE 6127 Introduction to Cryptology

Jorge L. Villar

UBa Cyber Crypto Center, Fall 2025

Message Authentication Codes and Hashing

—END—