Introduction to Modern Cryptography
Recitation 5

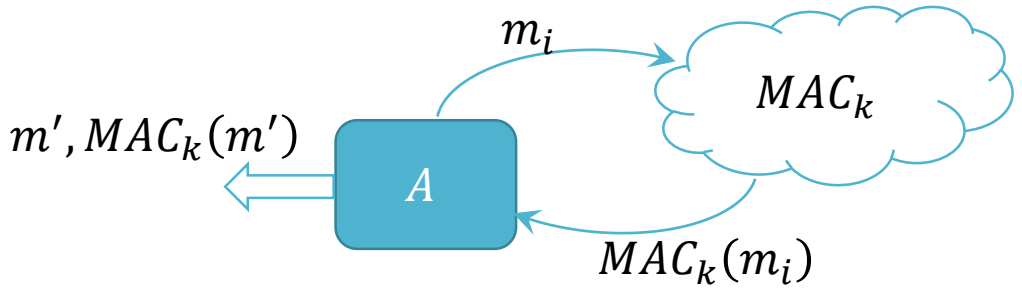
Orit Moskovich
Tel Aviv University
November 30, 2016

Message Authentication Codes (MAC)

- Goals:
  - Message integrity
  - Ensure that the message hasn’t been tampered with (detection)
- The parties must share a private key that the adversary does not know

Message Authentication Codes (MAC)

- A MAC is an algorithm that is applied to a message m
- The output of the algorithm is a tag $MAC_k(m)$ that is sent along with the message
- Security:
  - Any computationally bounded adversary cannot construct a new legal pair $\left(m', MAC_k(m')\right)$
  - Even after seeing n legal pairs $\left(m_i\ , MAC_k(m_i)\right)$ where $(i\ =\ 1, 2, \ldots, n)$



Message Authentication Codes (MAC)

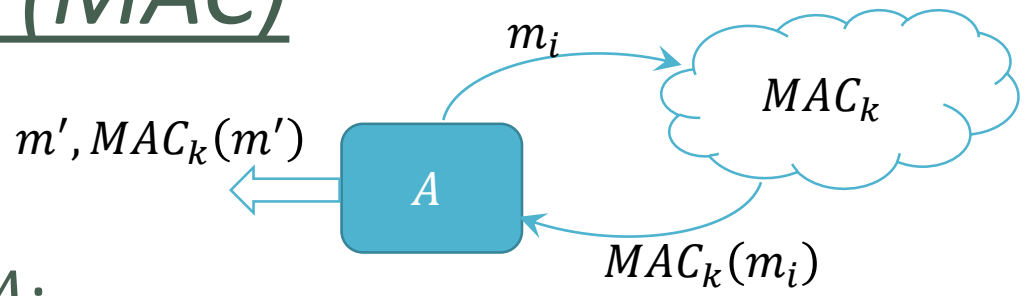
- A MAC is an algorithm that is applied to a message m
- The output of the algorithm is a tag $MAC_k(m)$ that is sent along with the message

Definition. A **message authentication code (MAC)** is a tuple of probabilistic polynomial-time algorithms $(Gen, MAC, Vrfy)$ such that:

1. Gen outputs a uniformly distributed key k
2. MAC receives for input a key k and a message m and outputs a tag $MAC_k(m)$
3. $Vrfy$ receives for input a key k and a tag t and a message m and verify its correctness, i.e., $Vrfy_k(m, t) = 1 \Leftrightarrow t = MAC_k(m)$

(Done by executing authentication algorithm on m and k, and comparing with the value received, $MAC_k(m)$)

Message Authentication Codes (MAC)



The message authentication experiment for A:

1. A random key k is chosen
2. The adversary A is given oracle access to $MAC_k(\cdot)$. Let Q denote the queries asked by A during the execution.
3. A outputs a pair (m, t)

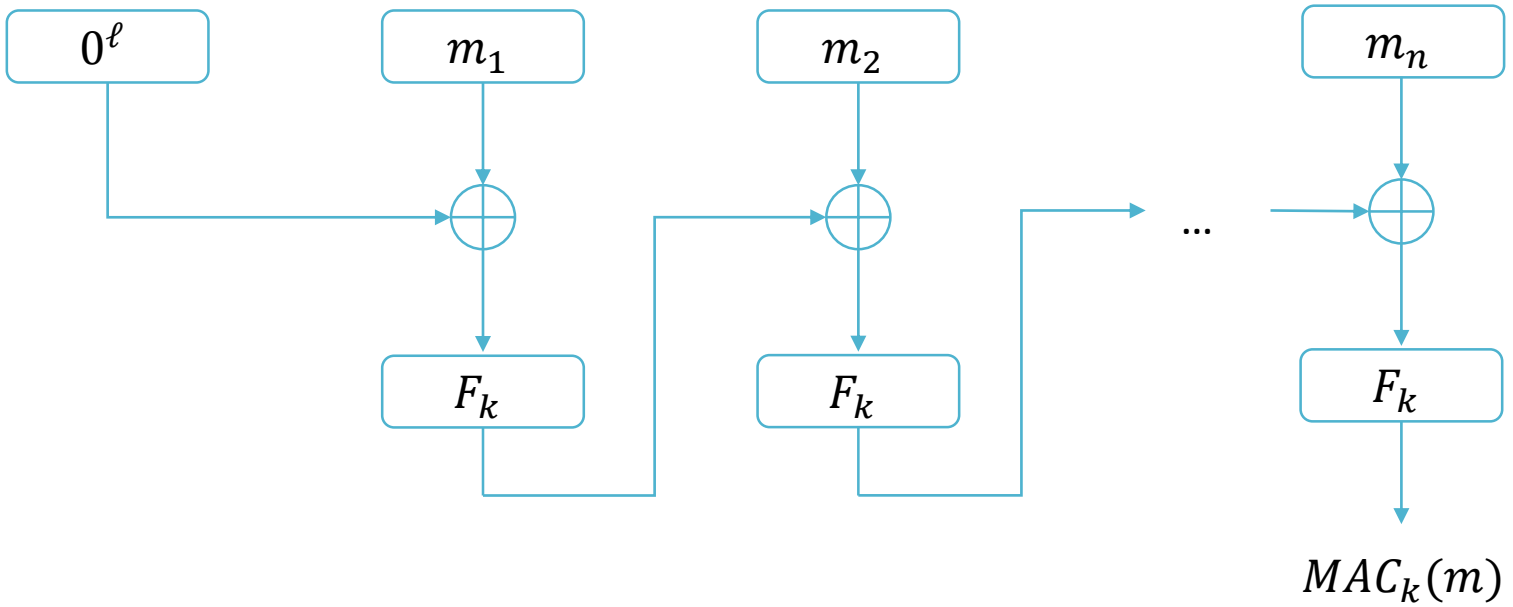
$$A \text{ forges a valid MAC} \Leftrightarrow t = MAC_k(m) \text{ and } m \notin Q$$

Definition. A MAC is ε-secure if for any PPT adversary A:

$$\Pr[A \text{ forges a valid MAC}] < \varepsilon$$

CBC-MAC

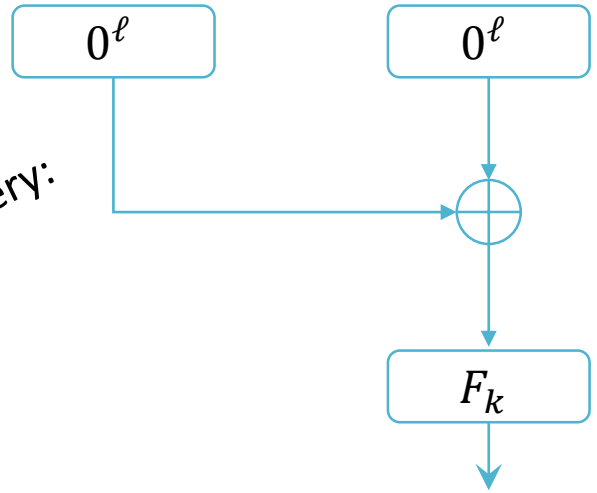
- Start with the all zero seed
- Given a message consisting of n blocks, $m = m_1, m_2, \ldots, m_n$
- Apply CBC mode encryption (using the secret key k)



CBC-MAC

- Not secure for messages of variable length

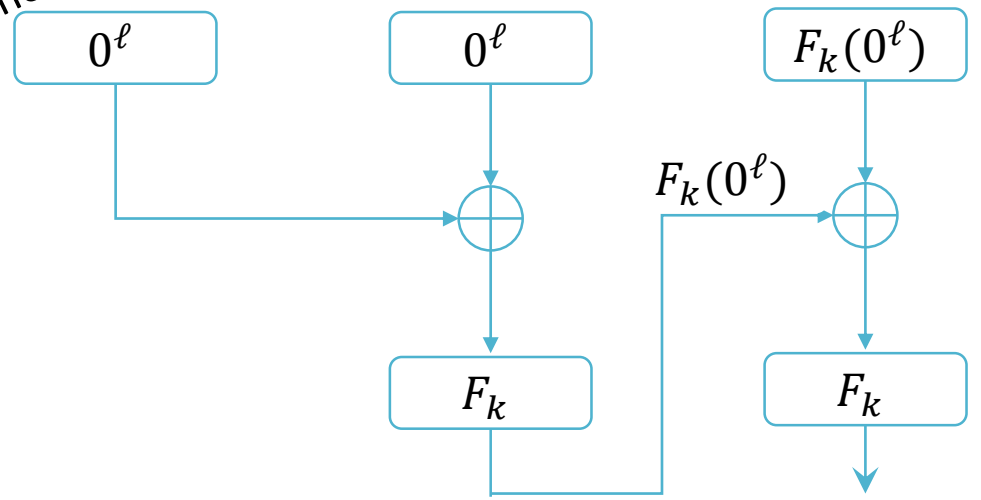
First Query:



$MAC_k(0^\ell) = F_k(0^\ell)$

Adversary outputs $0^\ell F_k(0^\ell), F_k(0^\ell)$

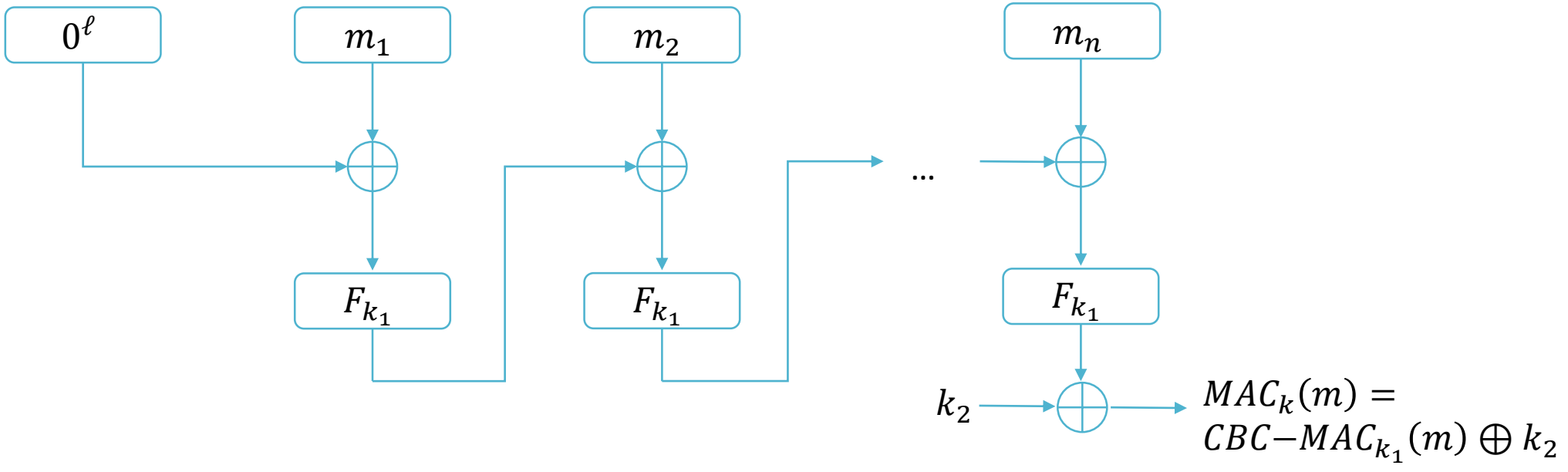
Correctness:



$MAC_k(0^\ell F_k(0^\ell)) = F_k(0^\ell)$

Modified CBC-MAC?

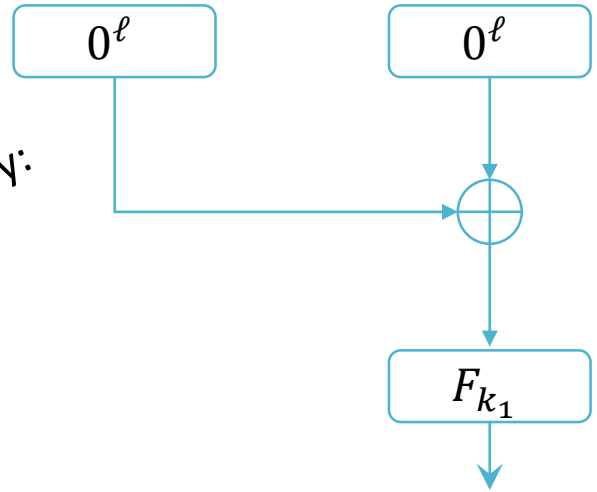
- Start with the all zero seed
- Given a message consisting of n blocks, $m = m_1, m_2, \ldots, m_n$
- Apply CBC mode encryption (using the secret key k_1), then XOR with k_2
- (Here the key $k = k_1, k_2$)



Modified CBC-MAC?

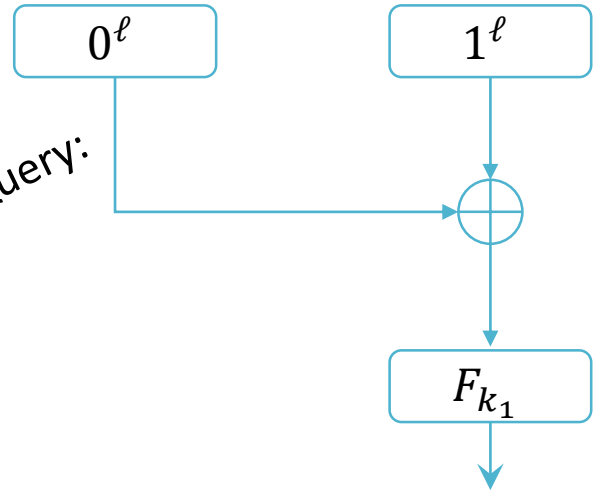
- Not secure for messages of variable length

First Query:



$$MAC_k(0^\ell) = CBC\text{-}MAC_{k_1}(0^\ell) \oplus k_2$$
$$= F_{k_1}(0^\ell) \oplus k_2 = t_0$$

Second Query:

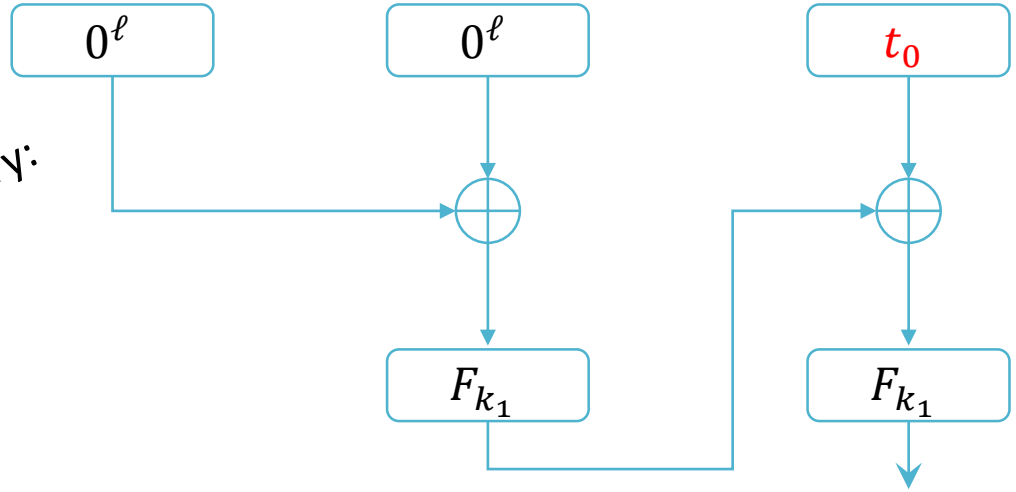


$$MAC_k(1^\ell) = CBC\text{-}MAC_{k_1}(1^\ell) \oplus k_2$$
$$= F_{k_1}(1^\ell) \oplus k_2 = t_1$$

Modified CBC-MAC?

- Not secure for messages of variable length

Third Query:



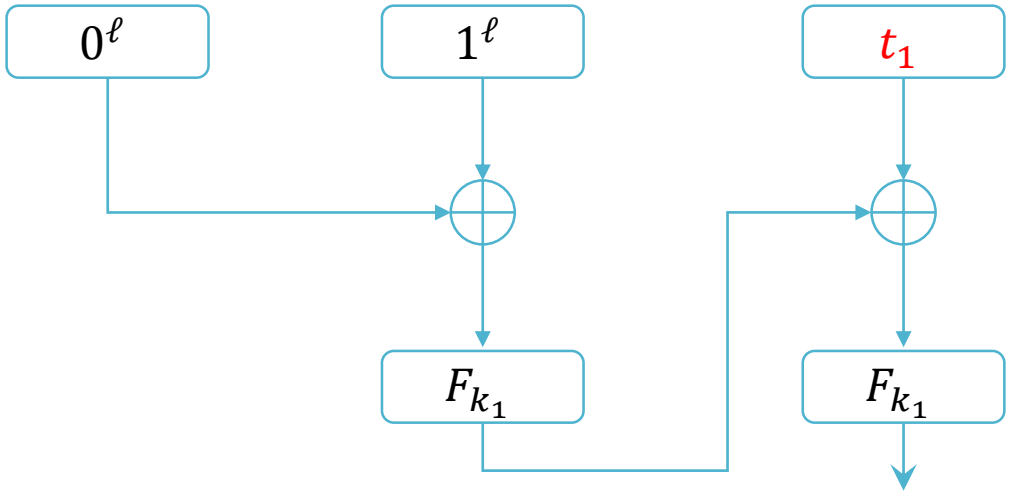
$$MAC_k(0^\ell t_0) = CBC\text{-}MAC_{k_1}(0^\ell t_0) \oplus k_2$$
$$= F_{k_1}(F_{k_1}(0^\ell) \oplus t_0) \oplus k_2$$
$$= F_{k_1}(F_{k_1}(0^\ell) \oplus F_{k_1}(0^\ell) \oplus k_2) \oplus k_2$$
$$= F_{k_1}(k_2) \oplus k_2$$

Modified CBC-MAC?

- Not secure for messages of variable length

Adversary outputs $1^\ell t_1, F_{k_1}(k_2) \oplus k_2$

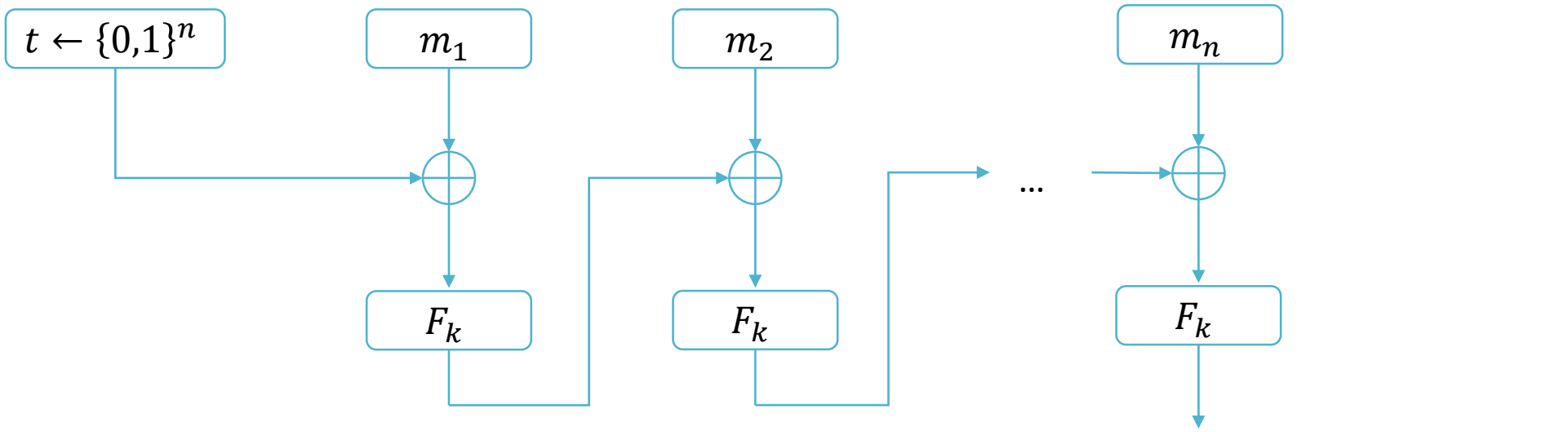
Correctness:



$$MAC_k(1^\ell t_1) = CBC\text{-}MAC_{k_1}(1^\ell t_1) \oplus k_2$$
$$= F_{k_1}(F_{k_1}(1^\ell) \oplus t_1) \oplus k_2$$
$$= F_{k_1}(F_{k_1}(1^\ell) \oplus F_{k_1}(1^\ell) \oplus k_2) \oplus k_2$$
$$= F_{k_1}(k_2) \oplus k_2$$

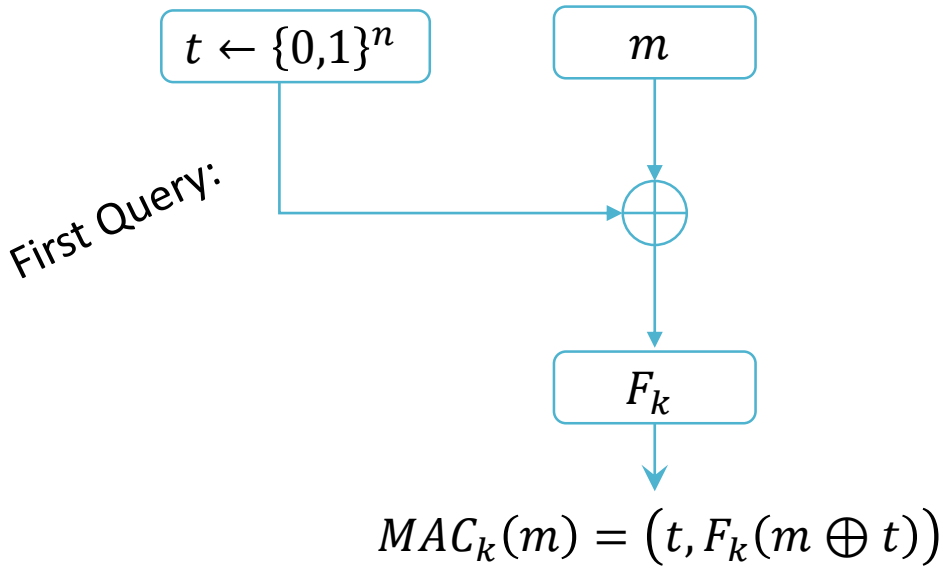
Random IV CBC-MAC?

- Start with a random seed
- Given a message consisting of n blocks, $m = m_1, m_2, \ldots, m_n$
- Apply CBC mode encryption (using the secret key k)

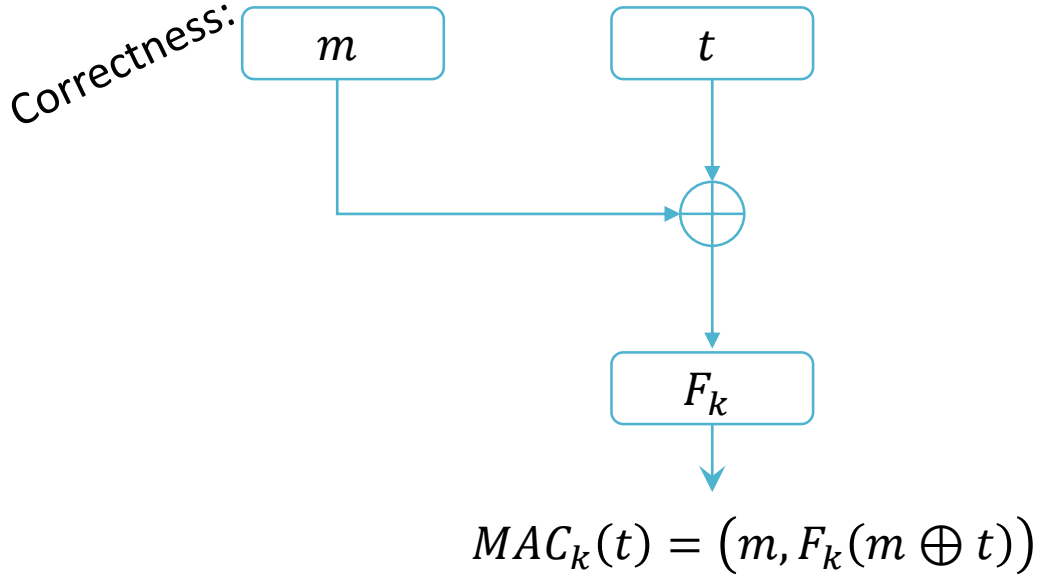


Random IV CBC-MAC?

- Not secure for messages of fixed length



Adversary outputs $t, (m, F_k(m \oplus t))$



Collision-Resistant Hash Functions (CRH)

- Functions that take arbitrary-length strings and compress them
- Goal: find as few collisions as possible ($x \neq x'$ such that $h(x) = h(x')$)
- How do we formalize this requirement?
  - A function h is collision-resistant if it is infeasible for any PPT algorithm to find a collision in h

Definition. A hash function $h: \{0,1\}^{n+s} \to \{0,1\}^n$ is **ε-collision resistant** if for any PPT adversary A:

$$\Pr_{A}[A(h) = x, x' \text{ s.t. } x \neq x' \text{and } h(x) = h(x')] < \varepsilon$$

Collision-Resistant Hash Functions (CRH)

- **Pre-image resistance**: h is preimage resistant if for any PPT adversary that is given an input y, it is hard to find x s.t. $h(x) = y$

- **Second pre-image resistance**: h is second preimage resistant if for any PPT adversary that is given an input x, it is hard to find $x' \neq x$ s.t. $h(x) = h(x')$

- **Collision resistance:** h is collision resistant if for any PPT adversary, it is hard to find any pair $x' \neq x$ s.t. $h(x) = h(x')$

Collision-Resistant Hash Functions (CRH)

- Can consider a family of keyed hash functions
- A family of hash functions indexed by a key k
- The requirement is that it is hard to find a collision in h_k for a randomly-chosen key k
- k is not a secret key

Definition. A family of hash functions $\{h_k\}$ is **ε-collision resistant** if for any PPT adversary A:

$$\Pr_{A,k}[A(h_k) = x, x' \text{ s.t. } x \neq x' \text{and } h(x) = h(x')] < \varepsilon$$

Collision-Resistant Hash Functions (CRH)

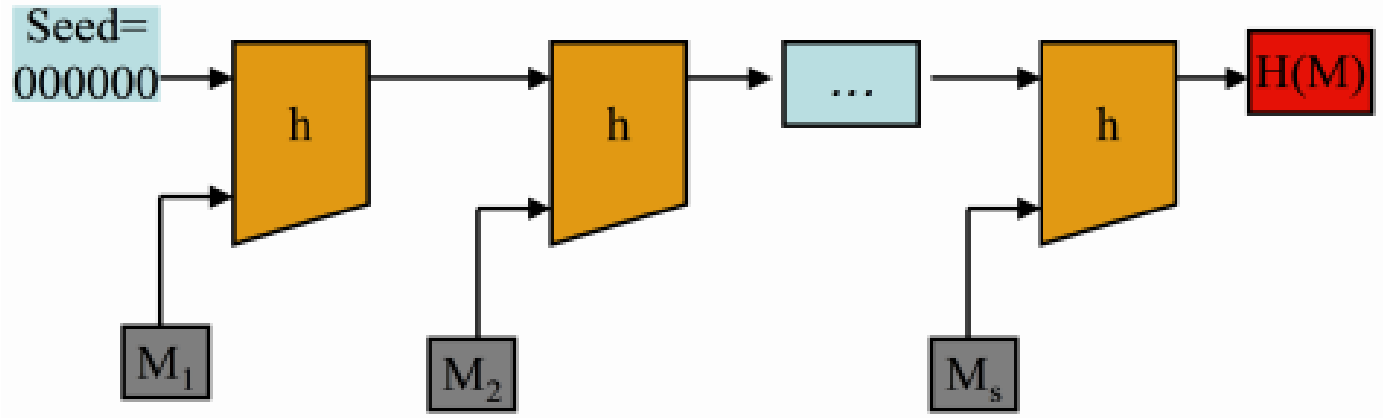
Example. Let h be a CRH. Define $H(s) = h(h(s))$. Prove that H is a CRH.

- Assume H is not a CRH
- There exists a PPT adversary A_H such that
  $\Pr_{A_H}[A_H(H) = x, x' \text{ s.t. } x \neq x' \text{and } H(x) = H(x')] > \varepsilon$
- Construct the following PPT adversary A_h:

1. The adversary A_h runs $x, x' \leftarrow A_H(H)$
2. If indeed $x \neq x'$ and $H(x) = H(x')$ (A_H succeeded)
   - If $h(x) \neq h(x')$, return $h(x), h(x')$
   - Otherwise, return x, x'

Yet Another Bad MAC

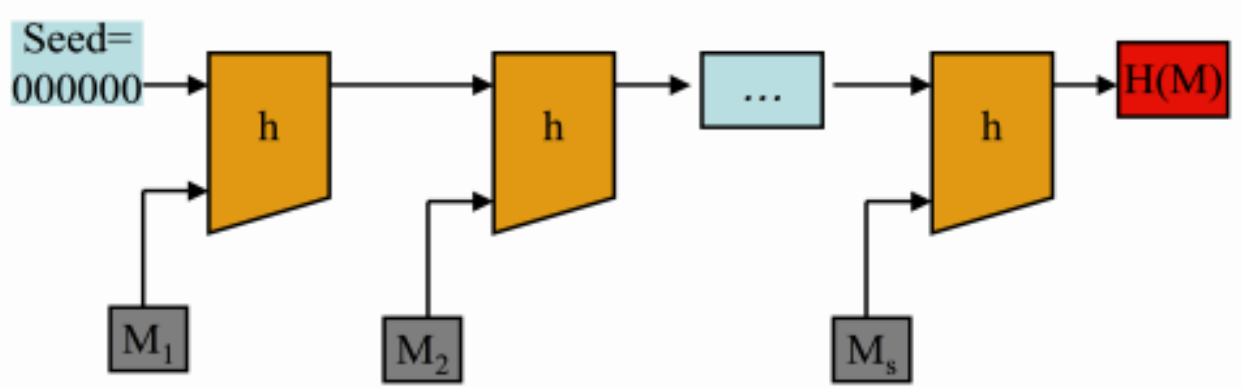
- Let $H: \{0,1\}^* \rightarrow \{0,1\}^n$ be the following hash function, where $h: \{0,1\}^{2n} \rightarrow \{0,1\}^n$
- H is a CRH



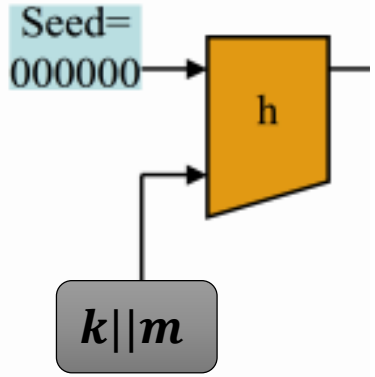
- Define $MAC_k(m) = H(k||m)$

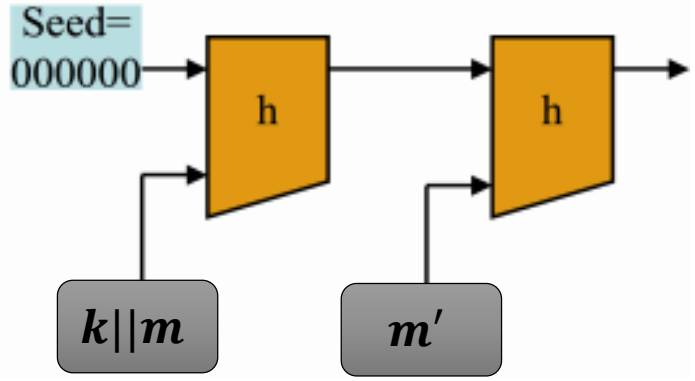
Yet Another Bad MAC

- Define $MAC_k(m) = H(k||m)$



- Adversary A:
  - Chooses $m \in \{0,1\}^{2n-|k|-\ell}$
  - Requests $MAC_k(m) = t = h(0^\ell \,||\, k \,||\, m)$
  - Chooses $m \neq m' \in \{0,1\}^n$
  - Computes $h(t \,||\, m') = h(h(0^\ell \,||\, k \,||\, m)||m')$
  - Outputs $m \,||\, m', h(t \,||\, m')$
- Correctness:
  - $MAC_k(m||m') = H(k||m||m')$
    $= h(h(0^\ell \,||\, k \,||\, m)||m')$



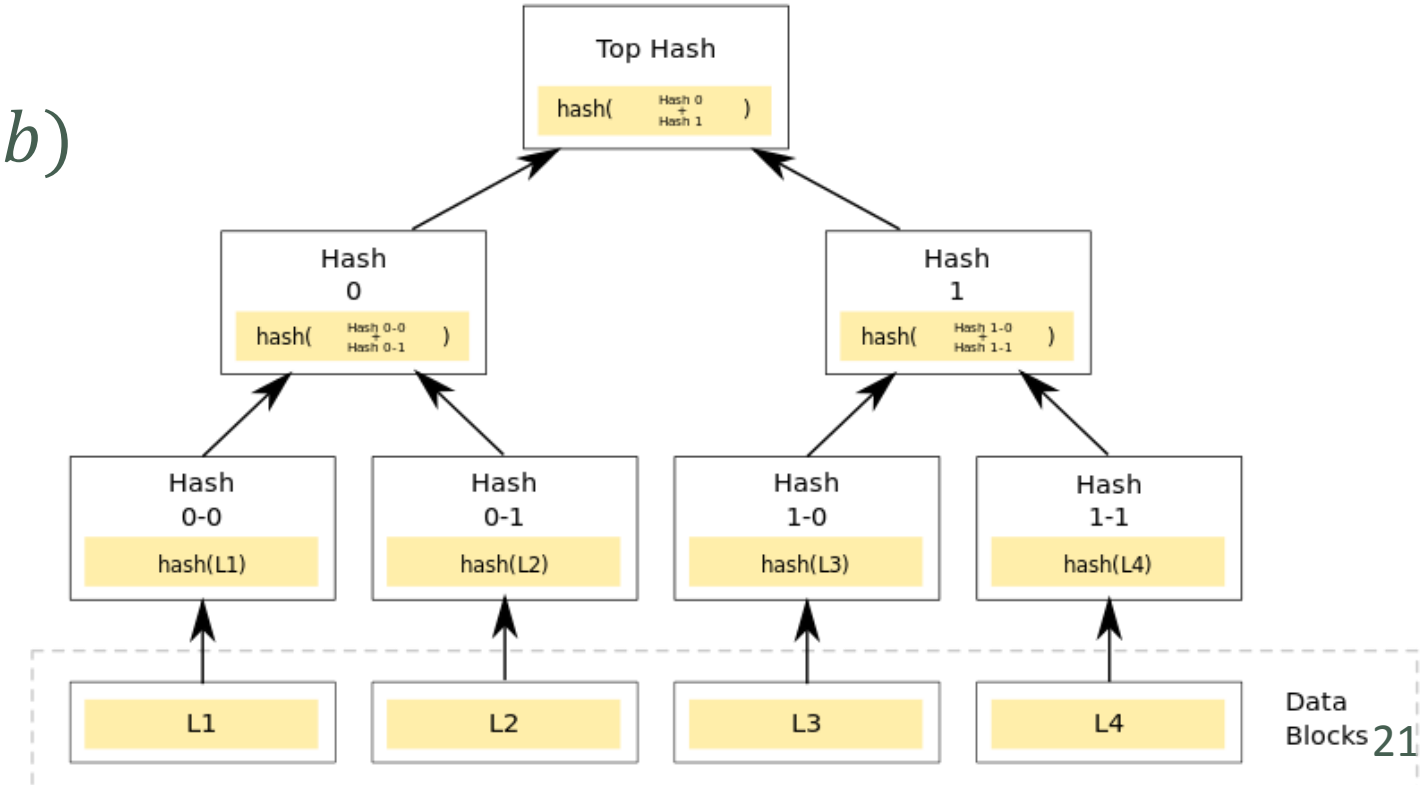


CRH- Motivation

- Say we want to store a database DB in the cloud
- When requesting for information (entry), we want to verify its correctness
- Naïve solution:
  - $d \in \{0,1\}^N$
  - Keep hash of the DB $h(d)$
  - Verifying correctness of a single entry is very expensive

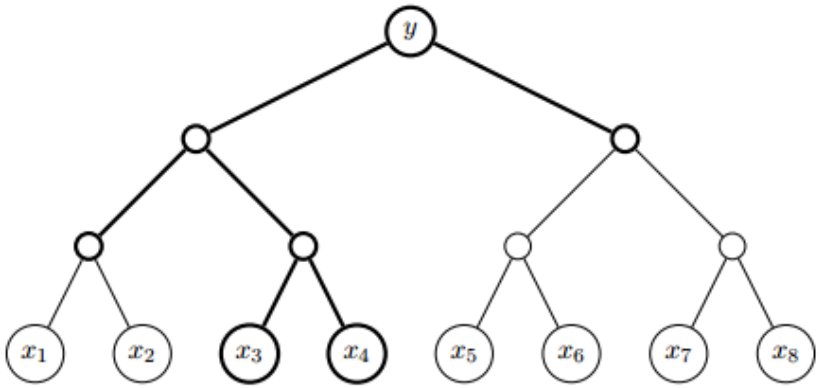
Merkle Tree

- We start with a hash function $h: \{0,1\}^{2n} \to \{0,1\}^n$
- Consider the following complete binary tree of depth
- All n-bit blocks $x_1, \ldots, x_{2^h}$ are stored in the leaves
- We define the parent of two nodes a, b to have value $h(a, b)$
- The root node has value $y = H(x_1, \ldots, x_{2^h})$ where $H: \{0,1\}^{2^h} \to \{0,1\}^n$



Merkle Tree

- What if we want to retrieve a single block x_i?
- How will we verify its correctness?
  - Only need to send the ith block x_i along with the hash outputs corresponding to all the sibling nodes along the path from the root of the tree to the ith block
  - We can then use these values to re-compute the value associated with the root of the tree and verify that this matches y



Merkle Tree

- Proof of security is similar to this of the construction seen in class