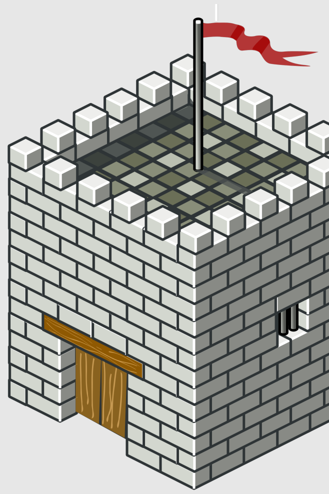
Foundations of Cybersecurity

VII - Hash and MAC functions



Paweł Szałachowski

2017

Cryptographic Hash Functions

- $H: \{0,1\}^* \rightarrow \{0,1\}^n$
  - for an arbitrarily long string produces a fixed-size output
    - output is called **digest**, or **fingerprint**, or just **hash**
      - usually between 128 and 1024 bits
- Many applications
  - integrity of messages
  - digital signatures
  - ...

Requirements

- Collision resistance
  - it is hard to find $m_1 \neq m_2$ such that $H(m_1) = H(m_2)$
- Pre-image resistance (one-way property)
  - given a hash value x it should be difficult to find any message m such that $x = H(m)$
- 2nd pre-image resistance
  - given an input m_1 it should be difficult to find different input m_2 such that $H(m_1) = H(m_2)$

Birthday Attack

- Generic attack against hash functions
  - *What is the minimum number of of people in a room, that the chance that two of them will have the same birthday exceeds 50%?*
    - *23*
  - N different values, choose k elements, then there are k(k-1)/2 pairs of elements, each of which has 1/N chance of being a pair of equal values
    - chance of finding a collision is close to k(k-1)/2N, and when k~= sqrt(N) this is close to 50%
- For a hash function that outputs *n* bits it is possible to find a collision in about $2^{n/2}$ steps as $\mathrm{sqrt}(2^n) = 2^{n/2}$

Security

- The **ideal hash function** behaves like a random mapping from all possible input values to the set of all possible output values
- An attack on a hash function is a non-generic method of distinguishing the hash function from an ideal hash function
- Security
  - Collision attack: $2^{n/2}$ steps
  - Pre-image attacks: 2^n steps

Real Hash Functions

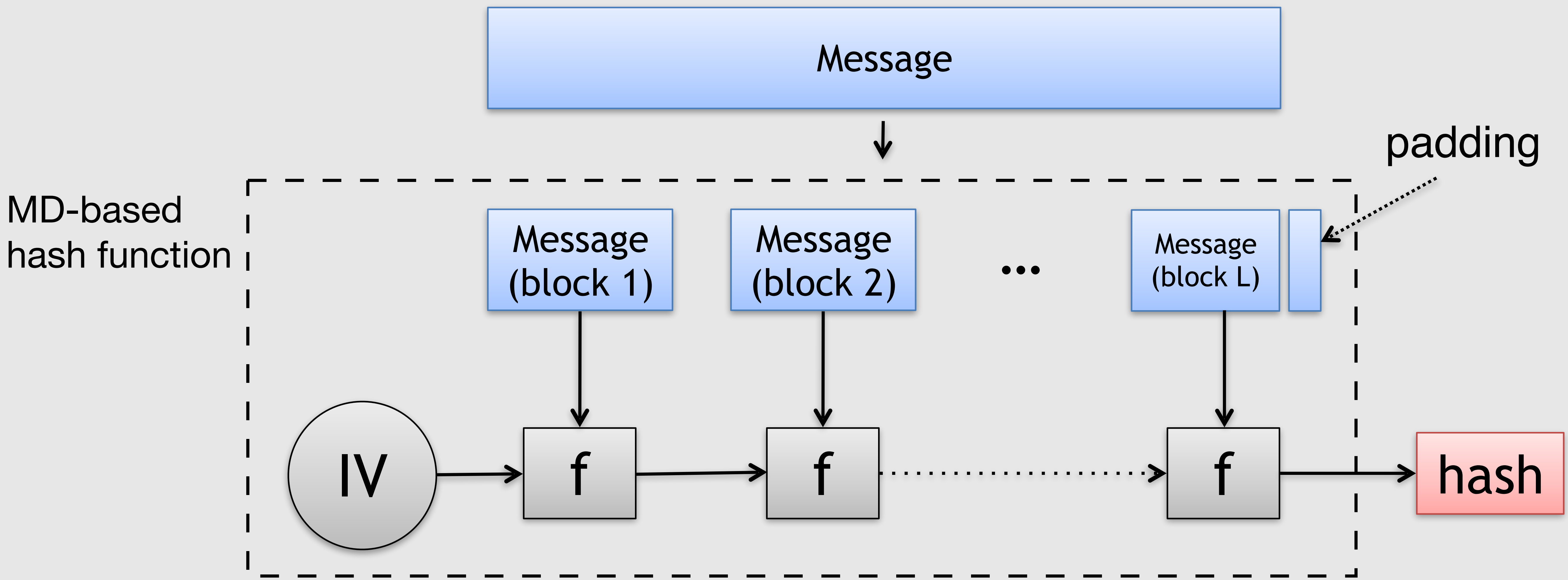
- Should be
  - deterministic
  - fast
  - secure
  - easy to analyze
- MD5, SHA1, SHA2, SHA3

Iterative Hash Functions (Merkle-Damgard construction)

- Split the input into fixed-size blocks m_1, ..., m_k
  - usually block size is 512-1024 bits
- Pad the last block
  - usually padding contains size of the input
- Process the message blocks in order, using a **compression function** *f()* and a fixed-size intermediate state.
  - $H_i = f(H_{i-1}, m_i)$ where H_0 is a fixed value (IV) and H_k is the hash

Merkle-Damgard

- iterative hash function
- IV is an initial state (known)
- if one-way compression function **f** is collision resistant, then so is the hash function
- padding is necessary (always added)



MD-based Hash Functions

- MD5
  - 16 byte long hash
  - insecure, DO NOT USE
- SHA1
  - 20 byte long hash
  - insecure, STOP USING
- SHA2
  - 28, 32, 48, or 64 byte long hash
  - secure

Length Extensions

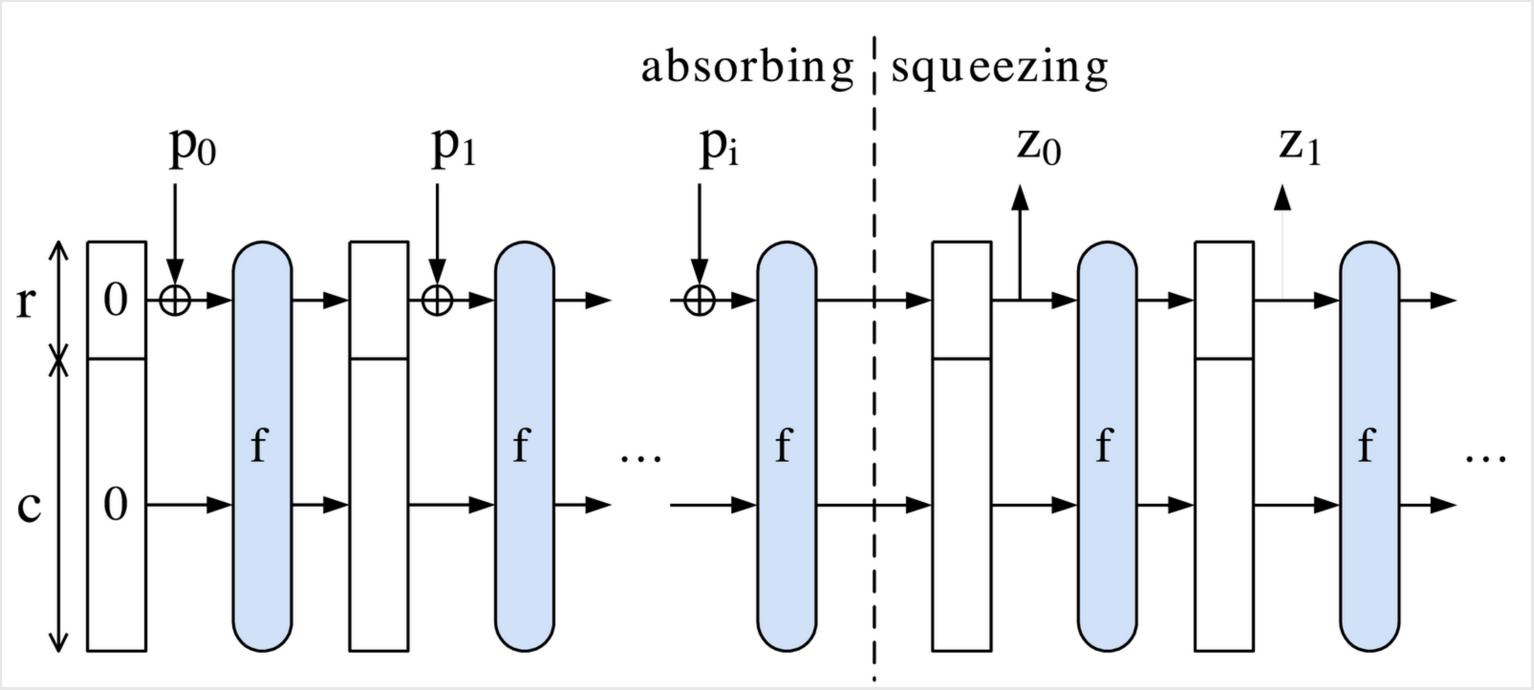
- Intuition: let's assume $m=m_1,...,m_k$ and $m'=m_1,...,m_k,m_{k+1}$
    - $H(m') = f(H(m), m_{k+1})$
        - m_k *and/or* m_{k+1} have to be prepared such that it contains correct padding, however the padding scheme is known
- Consequences
    - from one collision it is trivial to generate infinite number of collisions

Length Extension: Fixes

- Special processing is needed at the end of the process, e.g.,:
  - $H_{fixed} = H(H(m) \,||\, m)$
  - Truncate the output
  - ...

SHA3

- Current standard (since 2015)
- New design (sponge function)
  - eliminates problems of MD construction



Message Authentication Codes

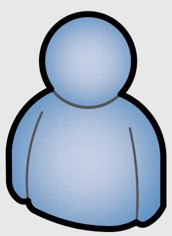
Message Authentication

- *is a procedure to verify that received message come from the alleged source and have not been altered.*
- Low-level primitive that produces an **authenticator**: a value to be used to authenticate a message
  - Hash function
  - Message encryption
  - **Message authentication code** (MAC)

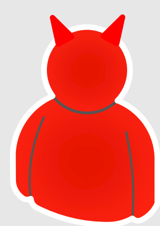
Hash function as a MAC?

Idea: hash of the entire message serves as its authenticator

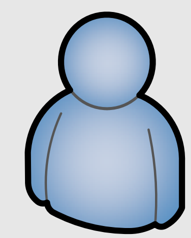
- Provides *integrity*, however does not provide authentication
  - everyone can compute hash (see the example)



Alice



Mallory



Bob

tag = H(“Hello from Alice”)

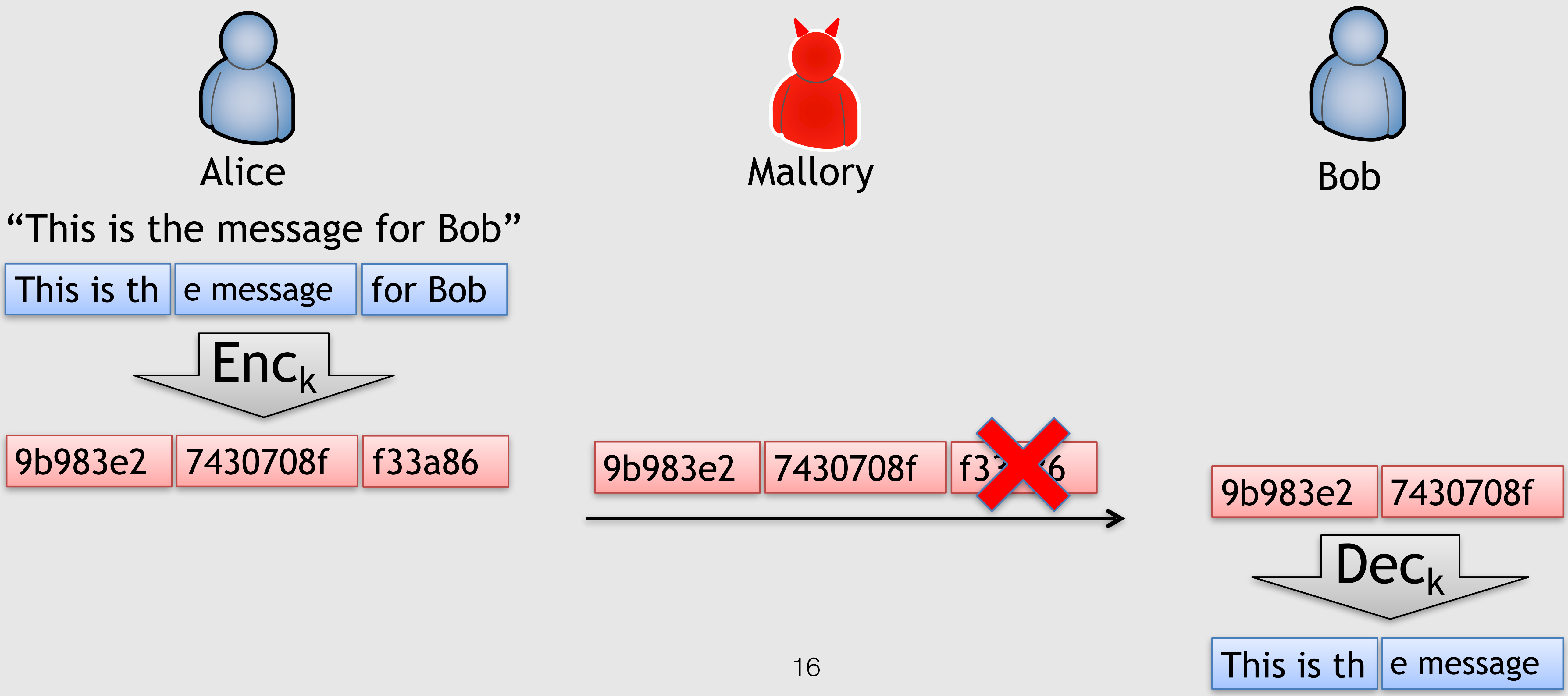
“Hello from Alice”, **tag** →

if H(“Hello from Alice”) ≠ tag:
    return FAIL

Symmetric encryption as a MAC?

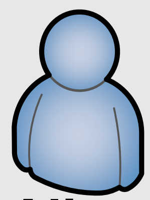
Idea: ciphertext of the entire message serves as its authenticator

- Not every information can be encrypted (e.g., packet headers)
- Symmetric encryption provides *confidentiality* but does not provide *integrity*
  - The message can be modified undetected (see the example)

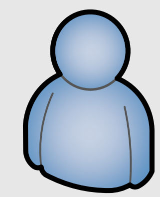


MAC: definition

- **MAC**: $\{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^n$
  - function that for shared secret key **K** and input message **M** generates a small fixed-size block of data known as a **tag** (or MAC or cryptographic checksum)



Alice



Bob

tag = MAC(K, **msg**)

msg, **tag** →

if tag ≠ MAC(K, **msg**):
return FAIL

1. Bob is assured that the message has not been altered: without **K** it is impossible to find correct tag for an altered message.
2. Bob is assured that the message is from Alice (only she knows **K** that is required to produce valid tags).
3. A sequence number or timestamp can additionally provide freshness.

Applications

- Often combined with encryption
    - Authenticated encryption
- Some data is (or can be) sent only in plaintext
    - Packet headers (are read by intermediate routers)
    - Non-sensitive information (sensor networks…)
- Authenticated *tickets*
    - Stateless access control and capabilities
    - HTTP(s) APIs
- …

Requirements

- Adversary knowing **M** and **MAC(K, M)** cannot compute **M'≠M** such that:
  MAC(K, M') = MAC(K, M)

- For any randomly chosen messages **M** and **M'**:
  Pr[MAC(K, M) = MAC(K, M')] = 2^{-n}

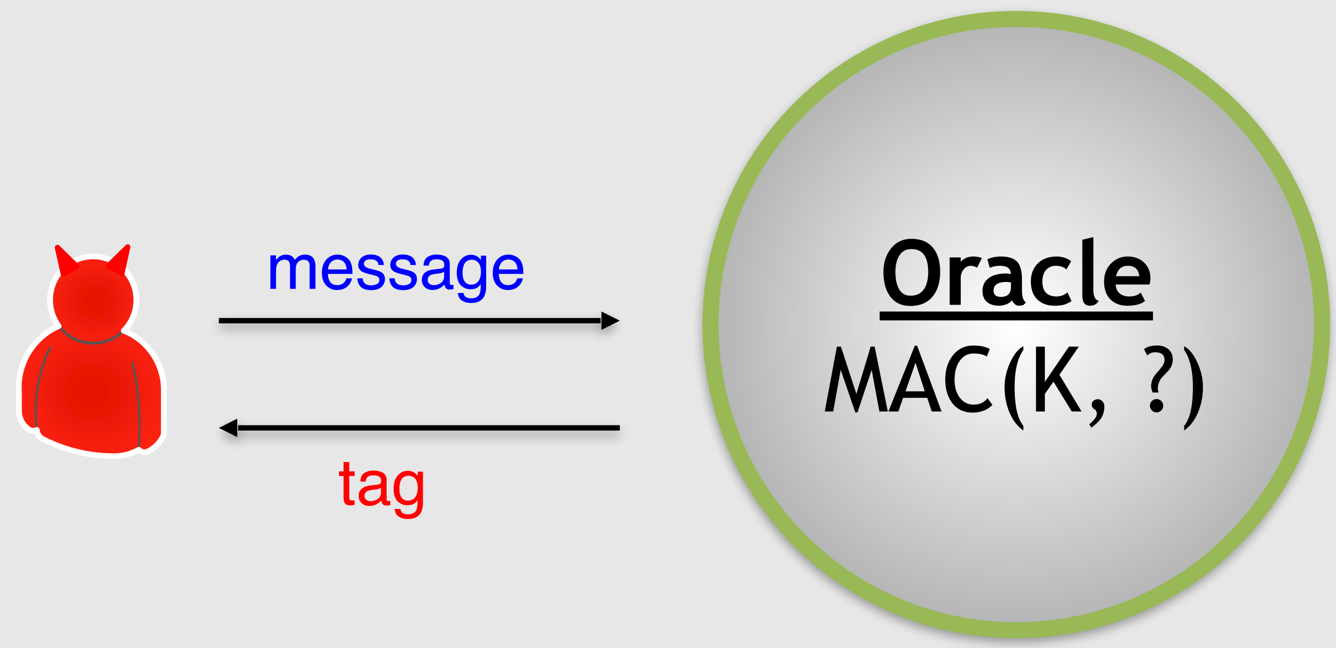
- For **M' = f(M)**, where **f** is some known transformation (e.g., inverting bits):
  Pr[MAC(K, M) = MAC(K, M')] = 2^{-n}

Security Property

- Computation resistance:

  Given one or more text-MAC pairs **[x_i , MAC(K, x_i)]**, *it is computationally infeasible to compute any text-MAC pair* **[x, MAC(K, x)]** *for any new input* **$x \neq x_i$**

Queries:
{„Hello world” ➔ d80c9d…,
„Hello world2” ➔ 828c82...,
„I’m Alice” ➔ bdbb07…,
…}



Can adversary (after quering) generate a new message and its valid tag?

Security: Brute-Force Attacks

Let's assume: k-bit long key, n-bit long tag, and an adversary has a valid (message, tag) pair

- Attack on the key (offline)

```
for key in {0,1}^k:
  if MAC(key, message) == tag:
    return key
```

  - $O(2^k)$ operations & possible collisions (more pairs needed)

- Attack on the tag (online)
  - Find other message for a given tag: $O(2^n)$ operations
  - Find a valid tag for a given message: $O(2^n)$ operations

The level of effort for brute-force attacks is $\min(2^k, 2^n)$

Realizations of MACs

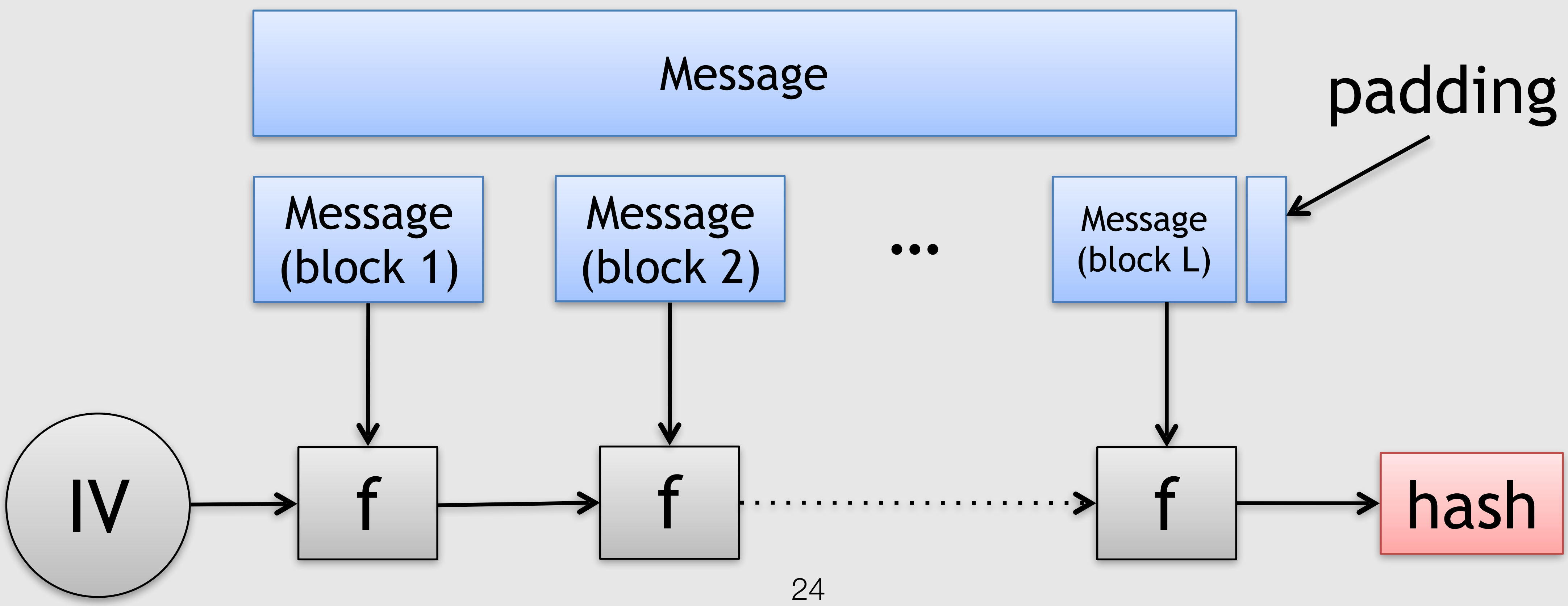
- Mainly based on hash functions and block ciphers
  - well-known primitives (e.g., SHA2, AES)
  - library code is widely available (e.g., OpenSSL, NSS)
  - fast implementations
  - hardware support (AES-NI)
- Hash functions
  - naïve constructions, **HMAC**, …
- Block ciphers
  - CBC-MAC, **CMAC,** GMAC, …

Hash-based MACs

- Hash functions are good candidates for MACs
- Need to merge a secret key
    - Why do not just hash a concatenated key and message?
- Security properties of hash function
    - Pre-image resistance
    - 2^{nd} pre-image resistance
    - Collision resistance

Hash-based MACs

- First intuition: define MAC(K, M) as H(K|M)
  - Unfortunately, insecure for MD-based hash functions
    - MD5, SHA1, SHA2, …
    - Merkle-Damgård construction (reminder):



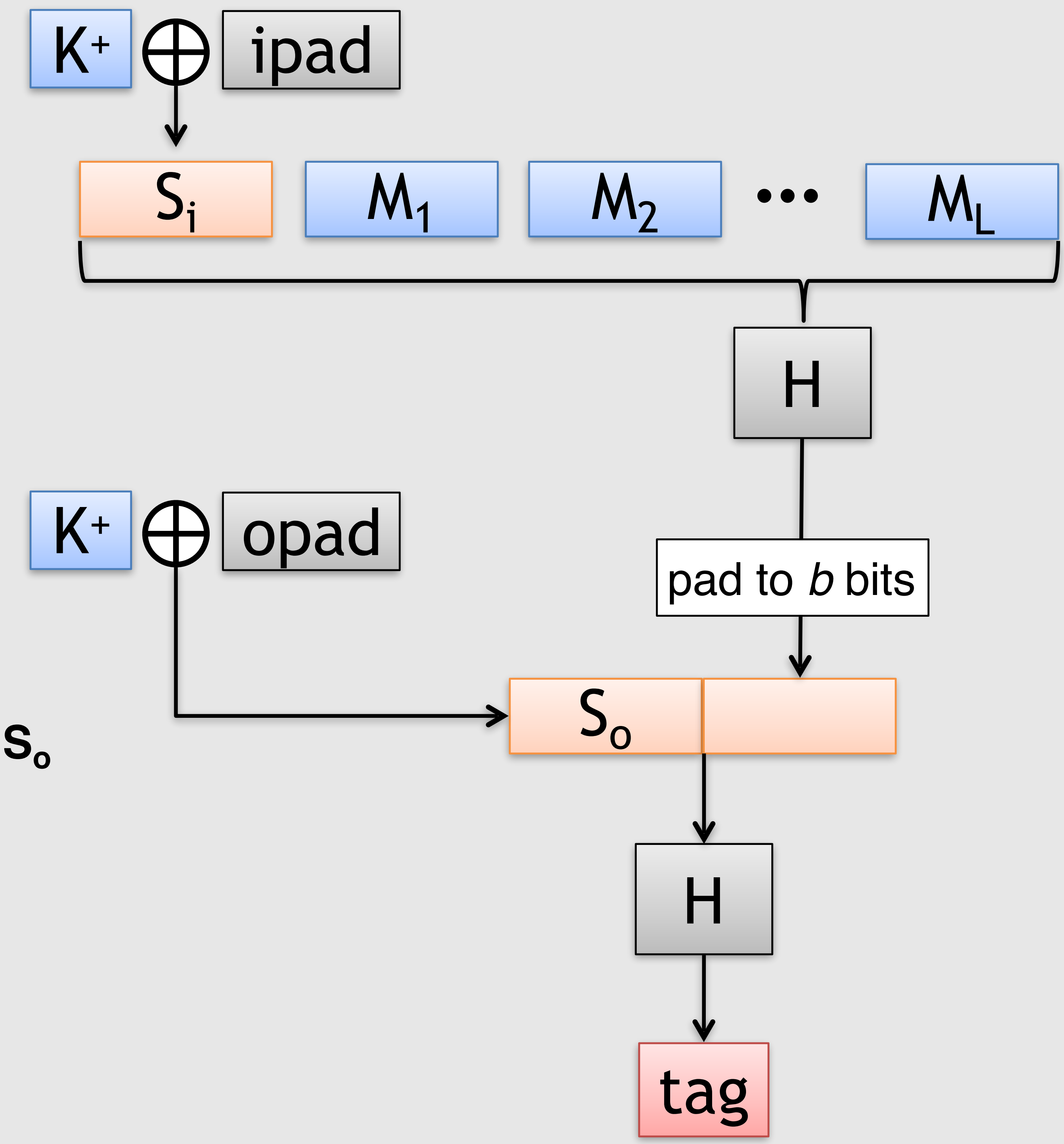
Alternatives

- H(M|K), H(K|M|K), ...
- HMAC: Keyed-Hashing for Message Authentication
  - Use available hash functions (usually hash functions have fast implementation)
  - Ease replaceability of the embedded hash function
  - Preserve the original performance of the hash function
  - Use and handle keys in a simple way
  - Well understood cryptographic analysis (provable security guarantees)
  - Standard (RFC2104, FIPS 198, IPsec, SSL/TLS, …)

HMAC(K,M) = H[($K^+ \oplus$opad) | H[($K^+ \oplus$ipad) | M]]

H: hash function that produces **n**-bit long hashes
b: number of bits in a block
K: secret key, recomm. length ≥ **n**
if len(**K**) > **b** then **K** = **H**(**K**)
ipad = 0x36*(**b**/8)
opad = 0x5c*(**b**/8)

1. Append zeros to the left end of **K** to create a **b**-bit string $\mathbf{K^+}$
2. XOR $\mathbf{K^+}$ with **ipad** to produce $\mathbf{S_i}$
3. Append **M** to $\mathbf{S_i}$
4. Apply **H** to the stream from step 3
5. XOR $\mathbf{K^+}$ with **opad** to produce $\mathbf{S_o}$
6. Append the hash result from step 4 to $\mathbf{S_o}$
7. Apply **H** to the stream from step 6 and output the result



HMAC: properties

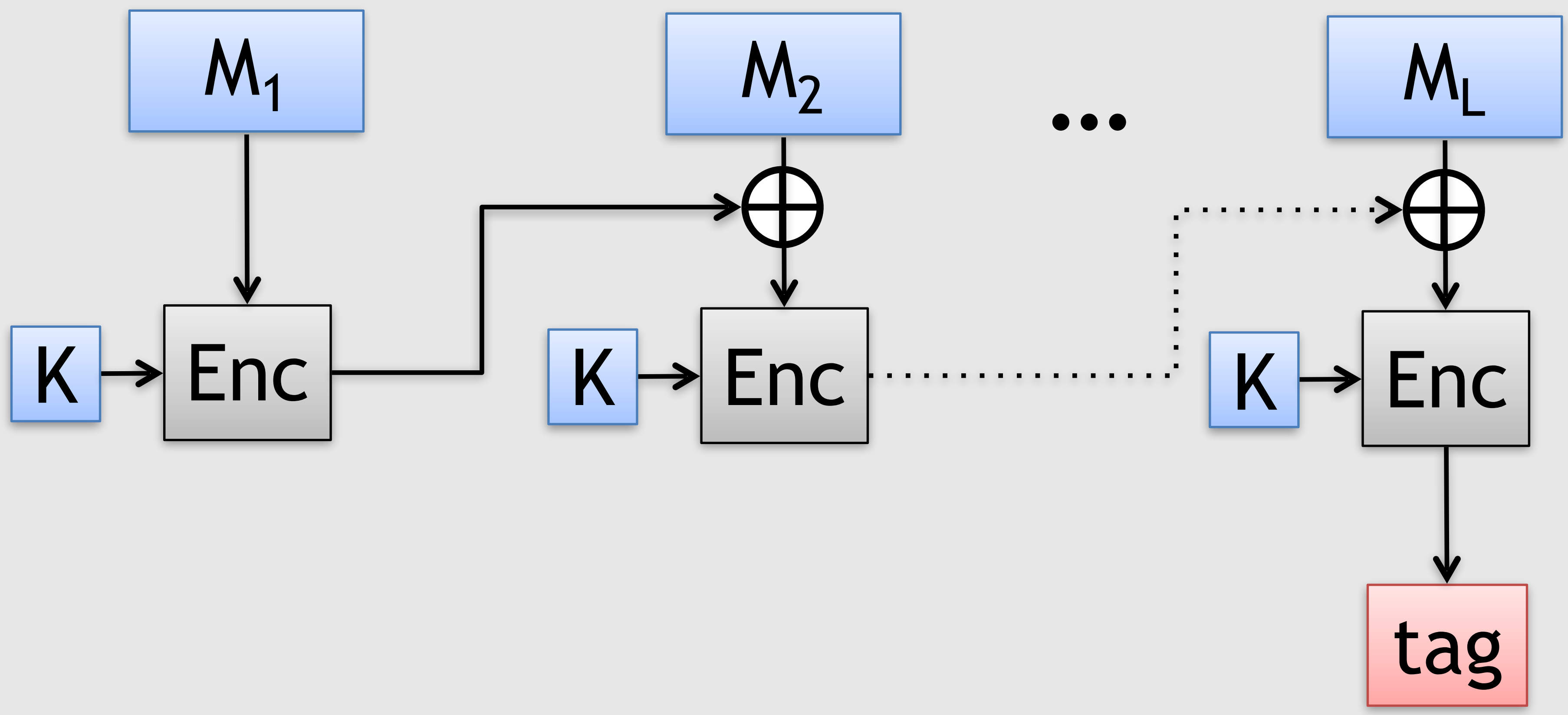
- HMAC can be attacked iff:
  - the attacker is able to compute an output of the compression function even with an **IV** that is random, secret, and unknown to the attacker
  - or, the attacker finds collisions in the hash function even when the **IV** is random and secret.

MACs based on block ciphers

- Block cipher
  - Pseudorandom permutation
- CBC-MAC / DAA
- CMAC

CBC-MAC

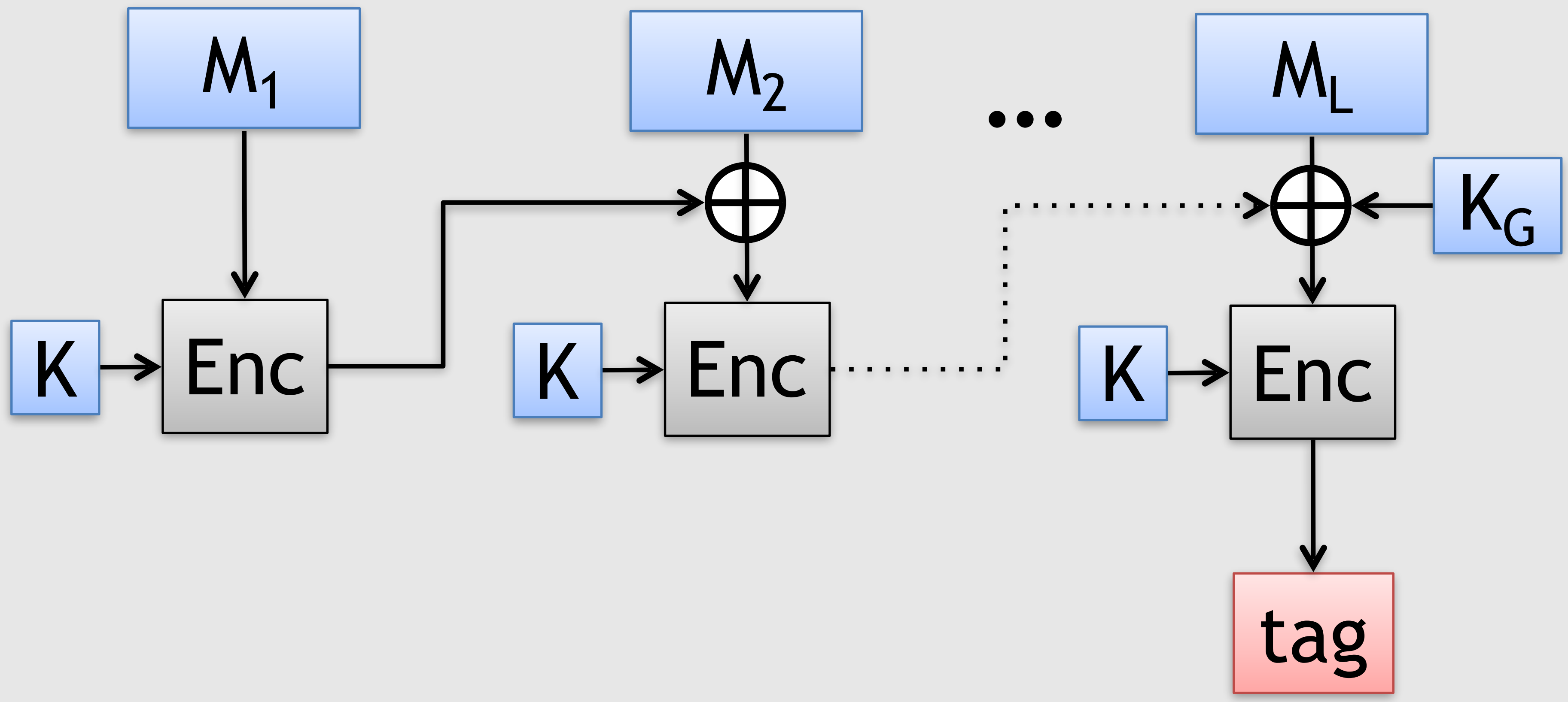
C_1 = Enc(K, M_1)
C_2 = Enc(K, $M_2 \oplus C_1$)
C_3 = Enc(K, $M_3 \oplus C_2$)
...
C_L = Enc(K, $M_L \oplus C_{L-1}$)
tag = C_L



- **M_L** can be padded as specified by the cipher
- If **Enc** is DES then it is DAA (an obsolete standard)
- **Insecure for variable-size messages**

CMAC

C_1 = Enc(K, M_1)
C_2 = Enc(K, $M_2 \oplus C_1$)
C_3 = Enc(K, $M_3 \oplus C_2$)
...
C_L = Enc(K, $M_L \oplus C_{L-1} \oplus K_G$)
tag = C_L



Z = **Enc**(**K**, 0…0)
$\mathbf{K_G}$ = **Z** • $\mathbf{const_1}$ if $\mathbf{M_L}$ is padded (by 10…0)
$\mathbf{K_G}$ = **Z** • $\mathbf{const_2}$ otherwise

CMAC

- Secure for variable-size messages
    - Different keys used for the padded and unpadded last block
    - Security proof
- Fast (small overheads)
- Standard (RFCs 4493&4494, NIST SP 800-38B)
- SSL/TLS

Exercises & Classwork