51.505 – Foundations of Cybersecurity

Week 8 – Hash & MAC Functions

Created by **Pawel Szalachowski** (2017)
Modified by **Jianying Zhou** (2018)

Last updated: 3 Nov 2018

Recap

- Questions on Week 5’s exercises?

Cryptographic Hash Functions

- H: $\{0,1\}^* \rightarrow \{0,1\}^n$
  - ✓ For an arbitrarily long string produces a <u>fixed-size</u> output.
  - ✓ Output is called **digest**, or **fingerprint**, or just **hash.**
  - ✓ Usually between 128 and 1024 bits.

- Many applications:
  - ✓ integrity of messages
  - ✓ digital signatures
  - ✓ pseudorandom number generator
  - ✓ …

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 people in a room, that the chance 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 $k \approx sqrt(N)$ 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 $sqrt(2^n) = 2^{n/2}$.

Security

- An **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 attack: 2^n steps

  ✓ 2^{nd} pre-image attack: how many steps?

Real Hash Functions

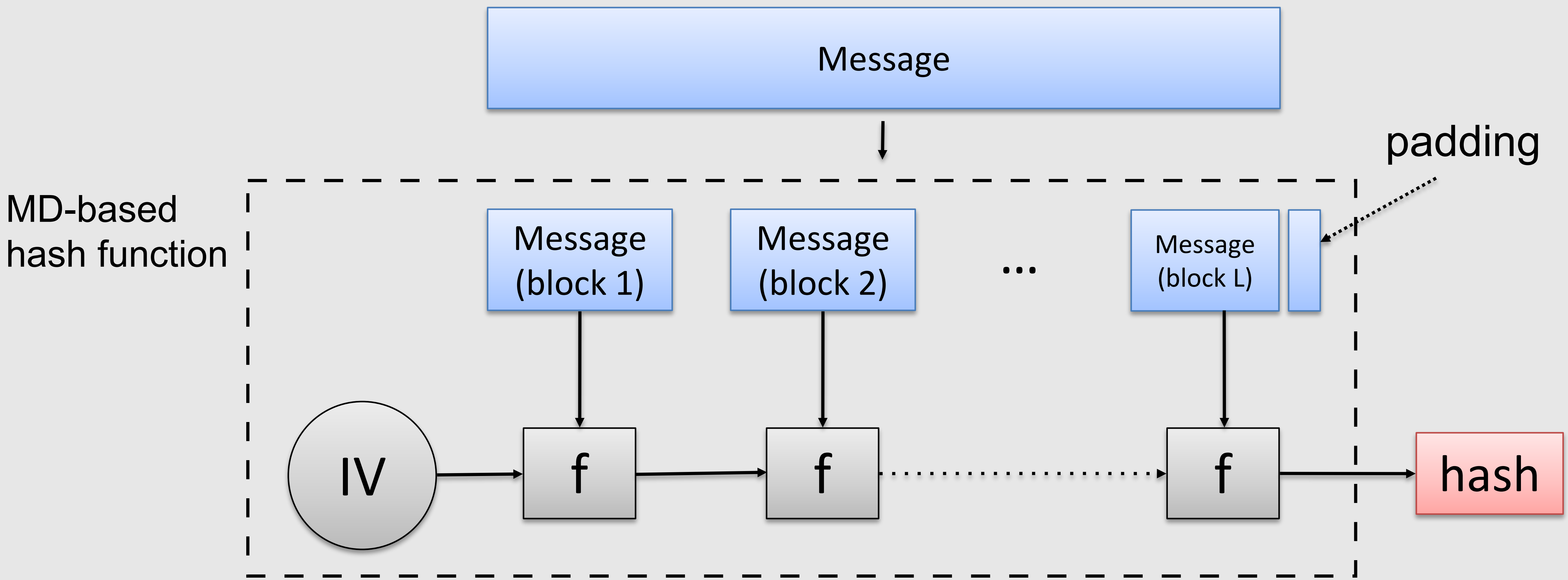
- Should be
  - ✓ deterministic
  - ✓ fast
  - ✓ secure
  - ✓ easy to analyze
- MD5, SHA1, SHA2, SHA3

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 $\boldsymbol{H_k}$ is the hash.
  - ✓ Advantage: Message can be hashed on the fly without ever storing the data.

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).



An Insecure MD Construction

- *Message m is spit into 128-bit blocks* $m_1, m_2, \ldots m_j$.
- $H_i = AES_K(H_{i-1} \oplus m_i)$ *where* $H_0 = 0$. H_j is the hash value of *m*.
- Why is this not a secure hash function?
  - ✓ Let $m = (m_1, m_2)$. $H_1 = AES_K(H_0 \oplus m_1)$, $H_2 = AES_K(H_1 \oplus m_2)$
  - ✓ Let $m' = (m_1', m_2') \neq m$, where $m_1' = m_2 \oplus H_1$, $m_2' = H_2 \oplus m_2 \oplus H_1$
  - ✓ $H_2' = H_2 \rightarrow$ collision !

MD-based Hash Functions

- MD5
  - ✓ 16 byte (128 bit) long hash
  - ✓ insecure, DO NOT USE
- SHA1
  - ✓ 20 byte (160 bit) long hash
  - ✓ insecure, STOP USING
- SHA2
  - ✓ SHA-224, SHA-256, SHA-384, SHA-512
  - ✓ secure

Length Extensions

- Intuition: let's assume $m = m_1,...,m_k$ and $m' = m_1,...,m_k,m_{k+1}$
  - ✓ $H(m') = f(H(m), m_{k+1})$
  - ✓ Length extension attack: $H(m)$ provides direct info about the intermediate state after the first k blocks of m'.
  - ✓ 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.,

✓ $H_{fixed}(m) = H(H(m) \| m)$

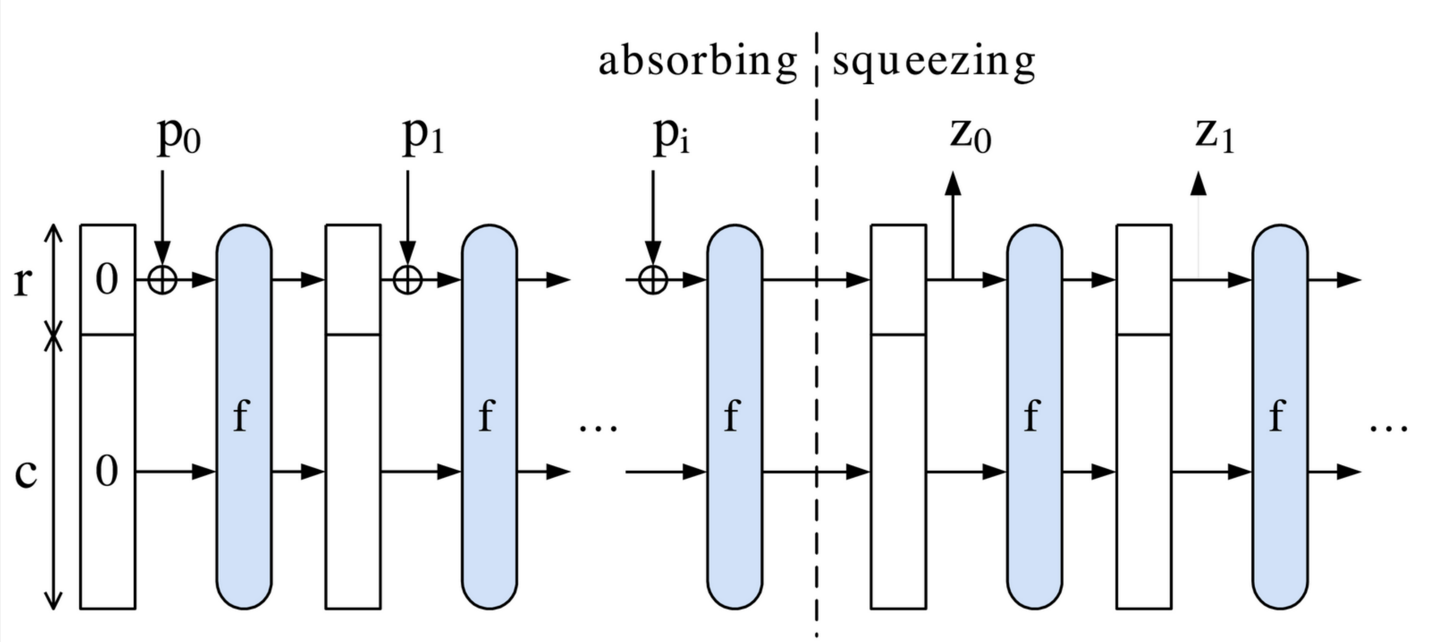
- The iterative hash computations immediately depend on all the bits of the message.
- Disadvantages: Slow, have to hash *m* twice. Whole message *m* to be buffered, cannot hashed on the fly.

✓ Truncate the output

- Only use the first *n*-*s* bits as the hash value.
- Example: SHA-512, drop 256 bits of output, return 256 bit hash value.

SHA3

- Current standard (since 2015)
- New design (sponge function)
  - ✓ A <u>b-bit permutation</u> f, with $b = r + c$
    - – r bits of rate
    - – c bits of capacity (security parameter)
  - ✓ Security level of $2^{c/2}$
- Eliminates problems of MD construction
  - ✓ XOF (eXtendable Output Function): the output can be extended to any length.



Instance	Output size d	rate r = block size	capacity c
SHA3-224(M)	224	1152	448
SHA3-256(M)	256	1088	512
SHA3-384(M)	384	832	768
SHA3-512(M)	512	576	1024

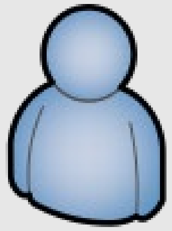
Message Authentication

- *Is a procedure to verify that received message comes from the alleged source and has 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)

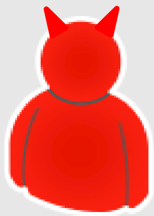
Hash Function as a MAC?

Idea: hash of the entire message serves as its authenticator.

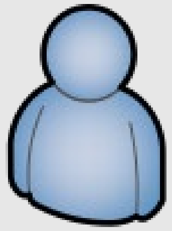
- Provides *integrity*, however does not provide authentication.
- Everyone can compute hash (see the example).



Alice



Mallory



Bob

tag = H("Hello from Alice")

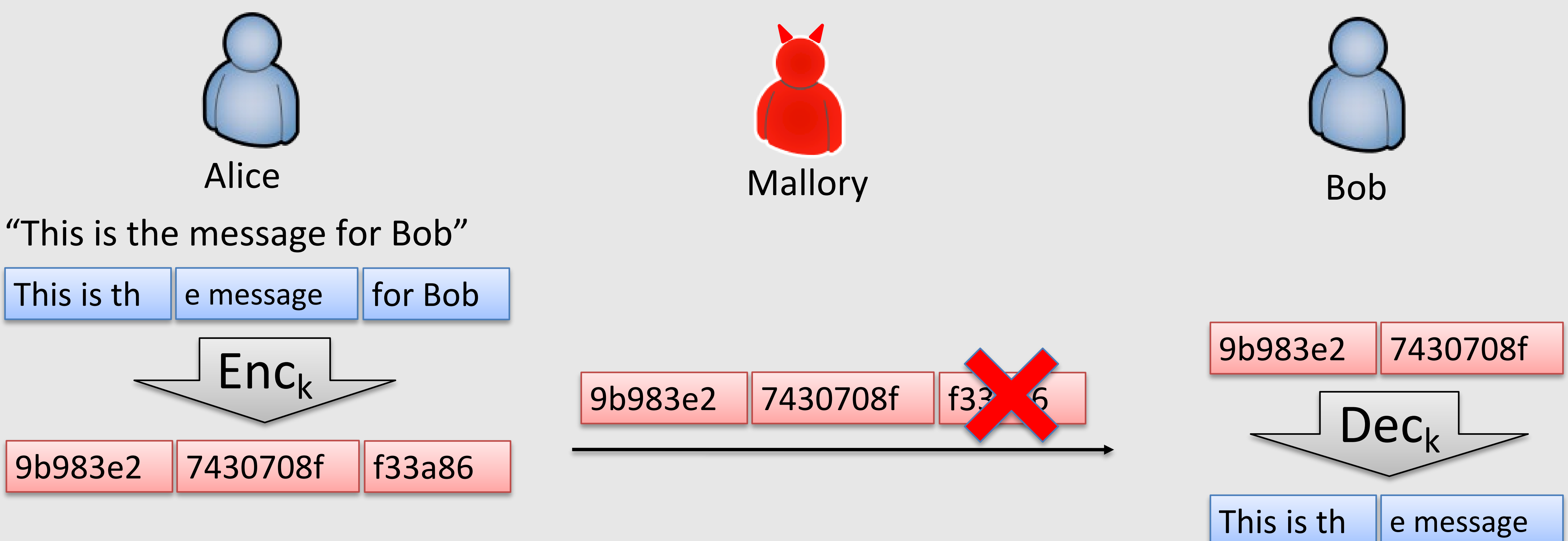
"Hello from Alice", **tag**

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

Symmetric Encryption as a MAC?

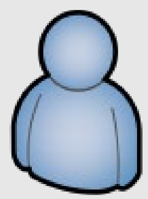
Idea: ciphertext of the entire message serves as its authenticator.

- 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).

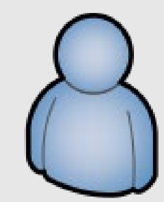


MAC Definition

- **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)***

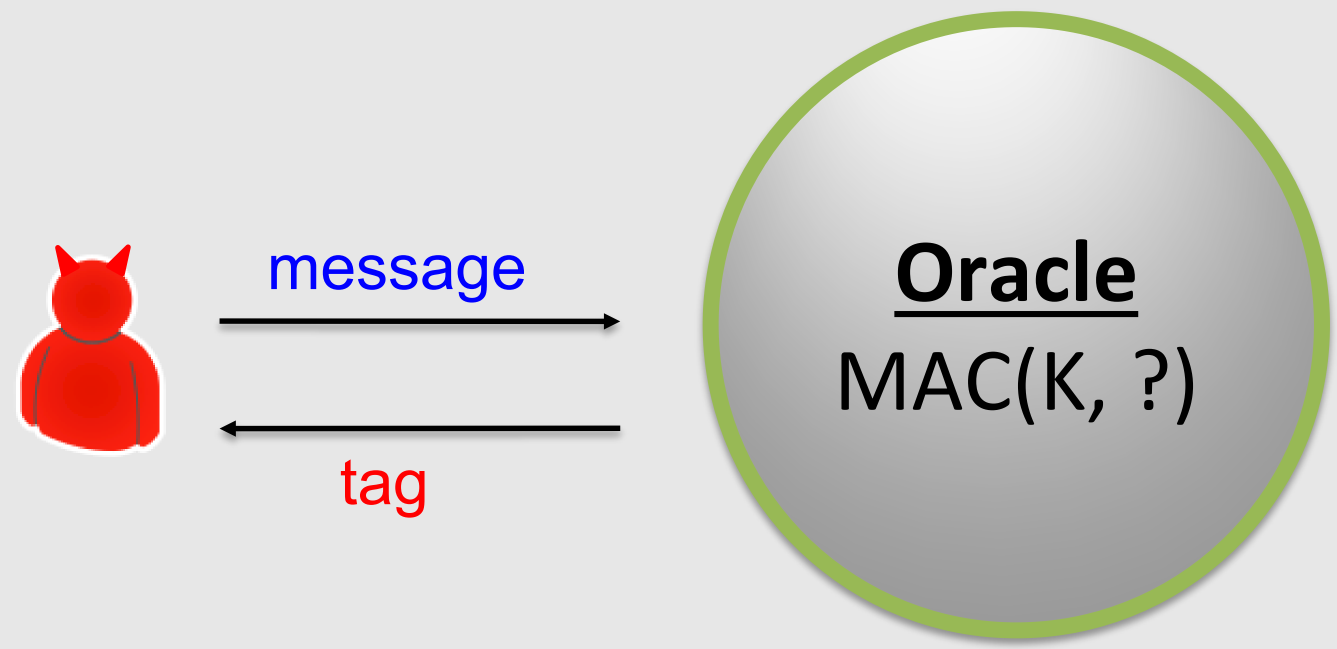
- For any randomly chosen messages ***M*** and ***M'***: $Pr[MAC(K, M) = MAC(K, M')] = 2^{-n}$

- 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 querying) 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

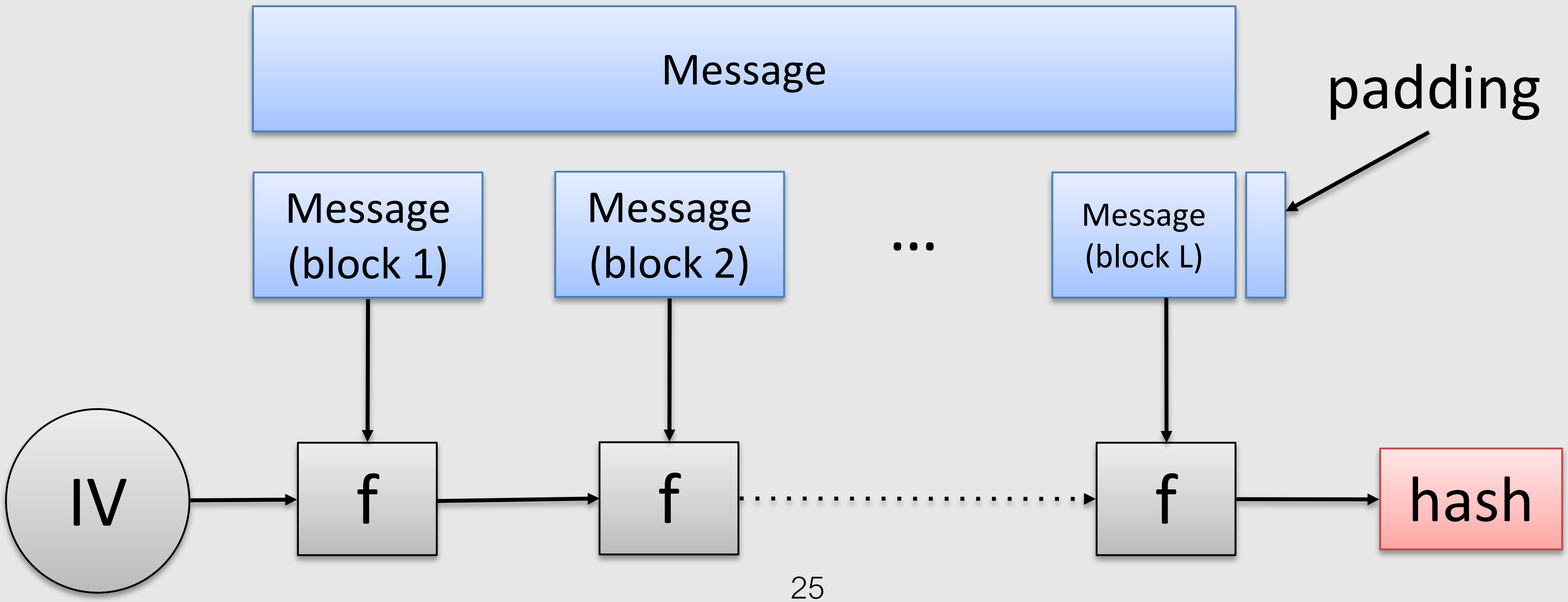
- Mainly based on hash functions and block ciphers
  - ✓ well-known primitives (e.g., SHA2, AES)
  - ✓ library code is widely available (e.g., OpenSSL, NSS)
  - ✓ fast implementations
  - ✓ hardware support (AES-NI)
- Hash functions
  - ✓ naïve constructions, **HMAC**, …
- Block ciphers
  - ✓ CBC-MAC, **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
  - ✓ 2nd 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
    - ✓ Subject to length extension attack.
    - ✓ Merkle-Damgård construction (reminder):



Alternatives

- **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, …)

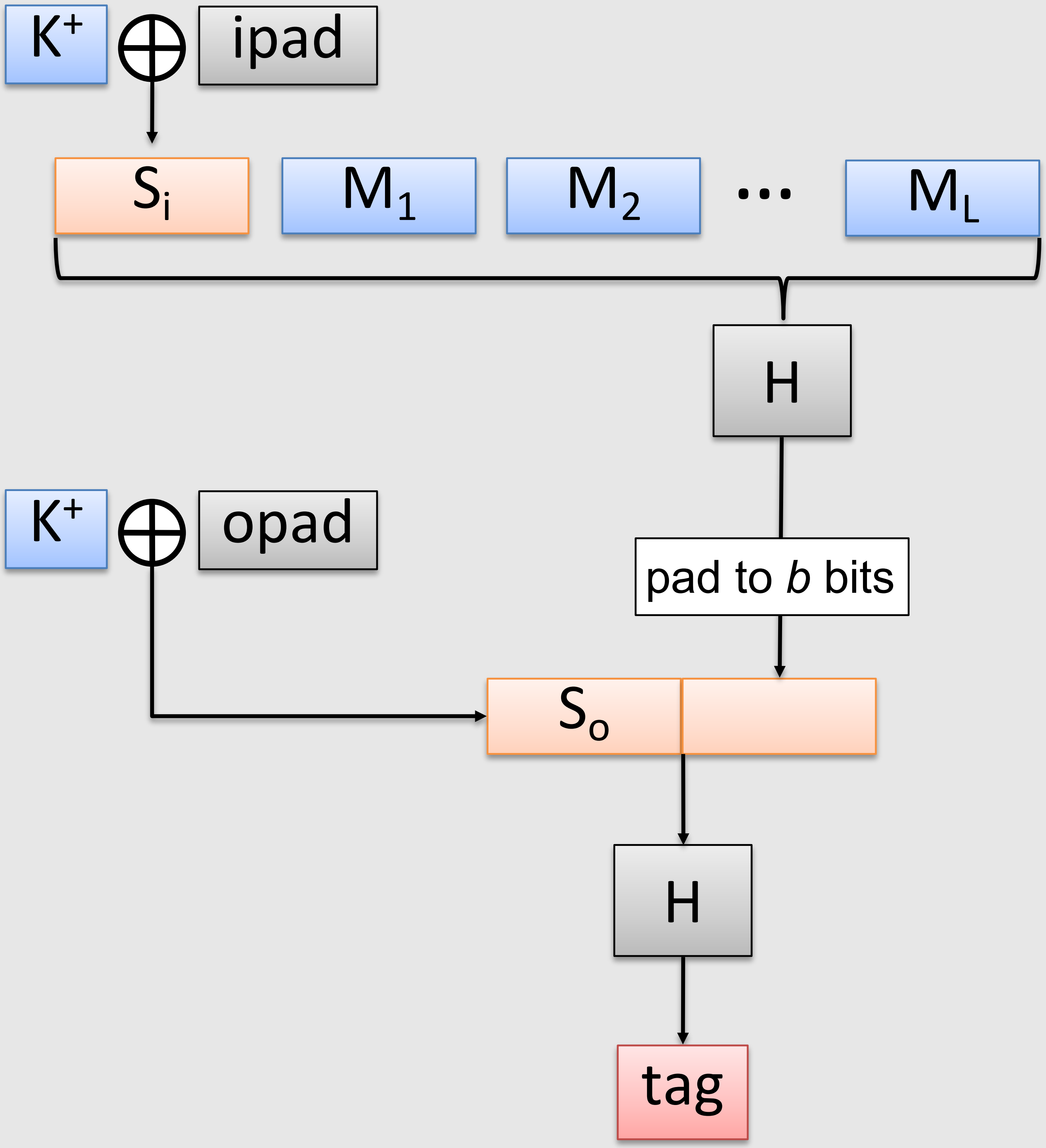
HMAC(K,M) = H[(K$^+ \oplus$opad) || H[(K$^+ \oplus$ipad) || M]]

H: hash function with b-bit output
b: block size (bits)
K: secret key

$$\mathbf{K^+} = \begin{cases} \mathbf{H}(\mathbf{K}) \text{ if } \mathrm{len}(\mathrm{K}) > \mathrm{b} \\ \mathbf{K} \text{ (pad to b bits if } \mathrm{len}(\mathrm{K}) < \mathrm{b}) \end{cases}$$

ipad = 0x36 * (**b**/8) – up to block size
opad = 0x5c * (**b**/8) – up to block size

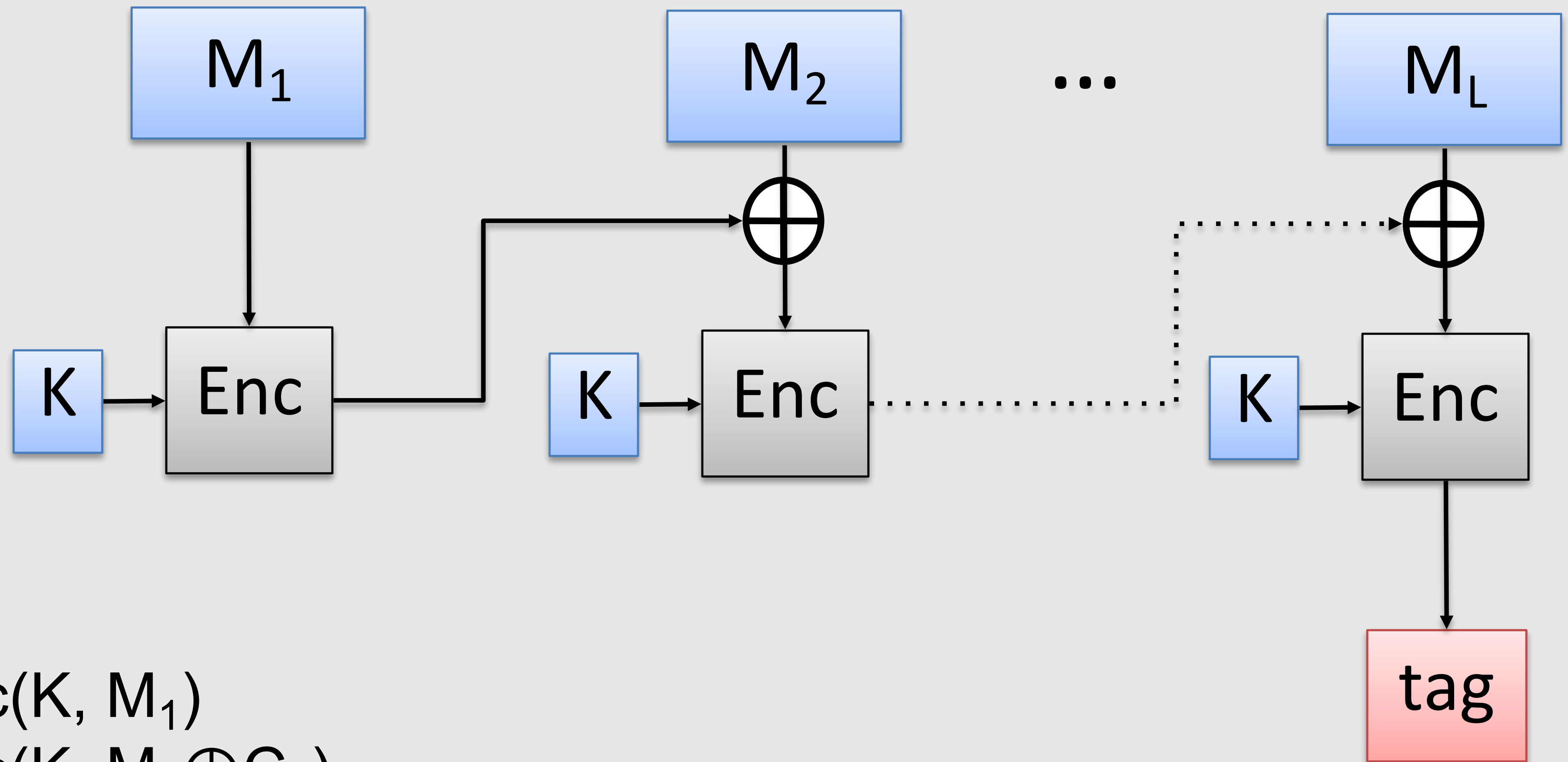
1. XOR $\mathbf{K^+}$ with **ipad** to produce $\mathbf{S_i}$
2. Append **M** to $\mathbf{S_i}$
3. Apply **H** to the stream from step 2
4. XOR $\mathbf{K^+}$ with **opad** to produce $\mathbf{S_o}$
5. Append the hash result from step 3 to $\mathbf{S_o}$
6. Apply **H** to the stream from step 5 and output 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 and unknown to the attacker,
  - ✓ or, the attacker finds collisions in the hash function even when the **IV** is random and secret.
- A typical HMAC construction:
  - ✓ HAMC-SHA-256
  - ✓ Truncating the hash of HAMC-SHA-256 to 128 bits is safe.

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

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

Collision Attack to CBC-MAC

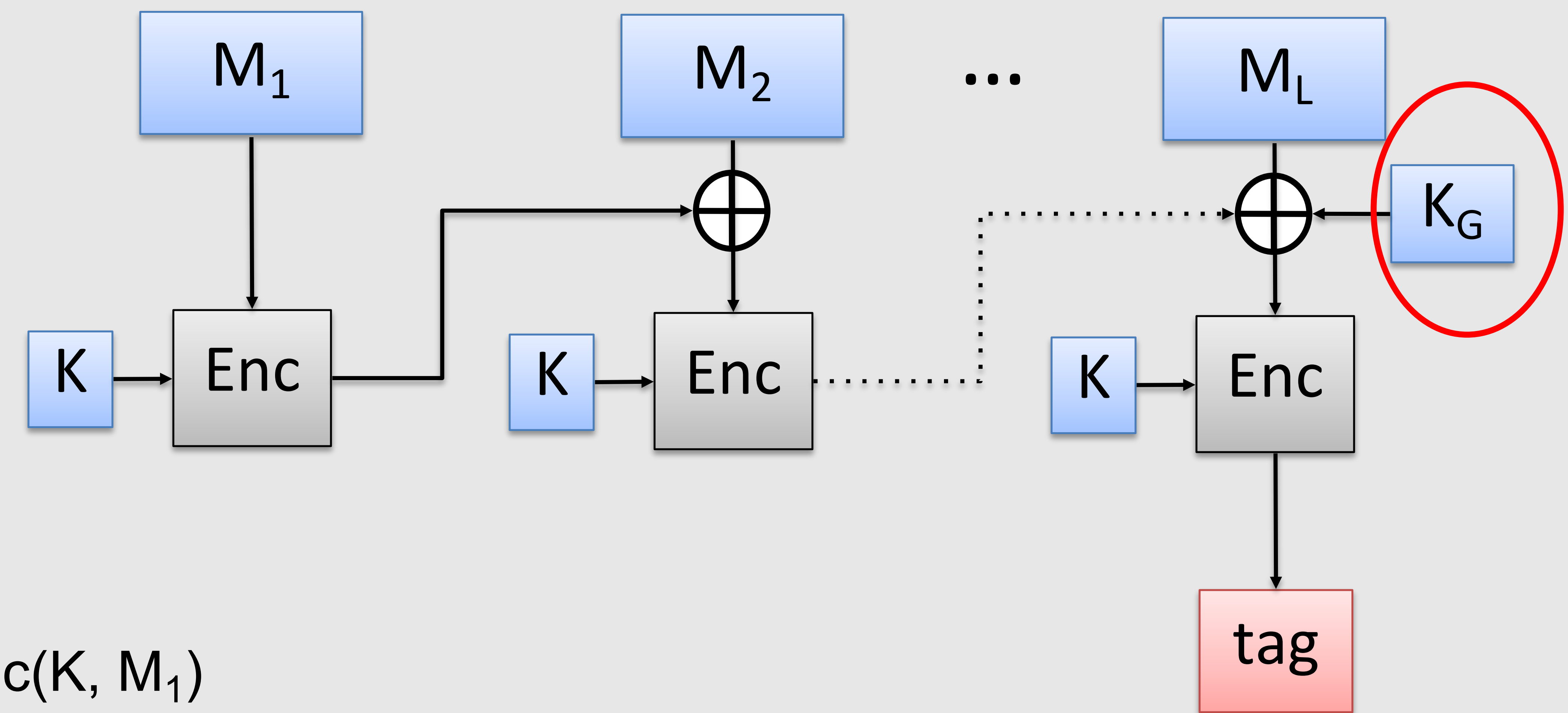
- Suppose *a* and *b* are 2 messages, and an attacker knows the collision $MAC_K(a) = MAC_K(b)$

- Suppose *c* is a single block message.

- Due to the structure of CBC-MAC, we have
  - ✓ $MAC_K(a||c) = Enc(K, c \oplus MAC_K(a))$
  - ✓ $MAC_K(b||c) = Enc(K, c \oplus MAC_K(b)) = Enc(K, c \oplus MAC_K(a)) = MAC_K(a||c)$

- If the attacker can get the sender to authenticate *a*||*c*, he can replace the message with *b*||*c* without changing the MAC value.

CMAC



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

$\mathbf{Z} = \mathbf{Enc}(\mathbf{K}, 0\ldots0)$
$\mathbf{K_G} = \mathbf{Z} \cdot \mathbf{const_1}$ if $\mathbf{M_L}$ is padded
$\mathbf{K_G} = \mathbf{Z} \cdot \mathbf{const_2}$ otherwise

CMAC Properties

- 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

Using MAC for Authentication

- Replay attack:
  - ✓ Alice and Bob use the same key *K* for authentication.
  - ✓ Alice sends *MAC (K,M)* to Bob.
  - ✓ Attack can replay *MAC (K,M)* to Alice.

- <u>Horton Principle</u>: Authenticate what it meant, not what is said.
  - ✓ MAC only authenticates a string of bytes, whereas Alice and Bob want to a message with a specific meaning.
  - ✓ Example: Alice uses MAC to authenticate $m := a||b||c$, where a,b,c are some data fields. Bob should know how to split m into the fields that Alice put in.

Key Points

- Hash functions:
  - ✓ Collision resistance
  - ✓ One-way property
  - ✓ Birthday attack
- MAC:
  - ✓ HMAC
  - ✓ CMAC

Exercises & Reading

- Classwork (Exercise Sheet 8): due on Fri Nov 2, 10:00 PM
- Homework (Exercise Sheet 8): due on Fri Nov 9, 6:59 PM
- Reading: FSK [Ch5, Ch6]

End of Slides for Week 8