Cryptography IV: Message Integrity, Hash Functions, and Authenticated Encryption

CSE 565: Fall 2024

Computer Security

Xiangyu Guo (xiangyug@buffalo.edu)

University at Buffalo

Announcement

- Please sign-up at course Piazza.
- Reminder of Quiz 0 (**Due 09/19**).
    - You must obtain full score of the Quiz.
    - Updated so that you can see exactly where you got wrong.
- Assignment 1 & Project 1 will be released tonight (**Due 09/24**).

Review of Last Week

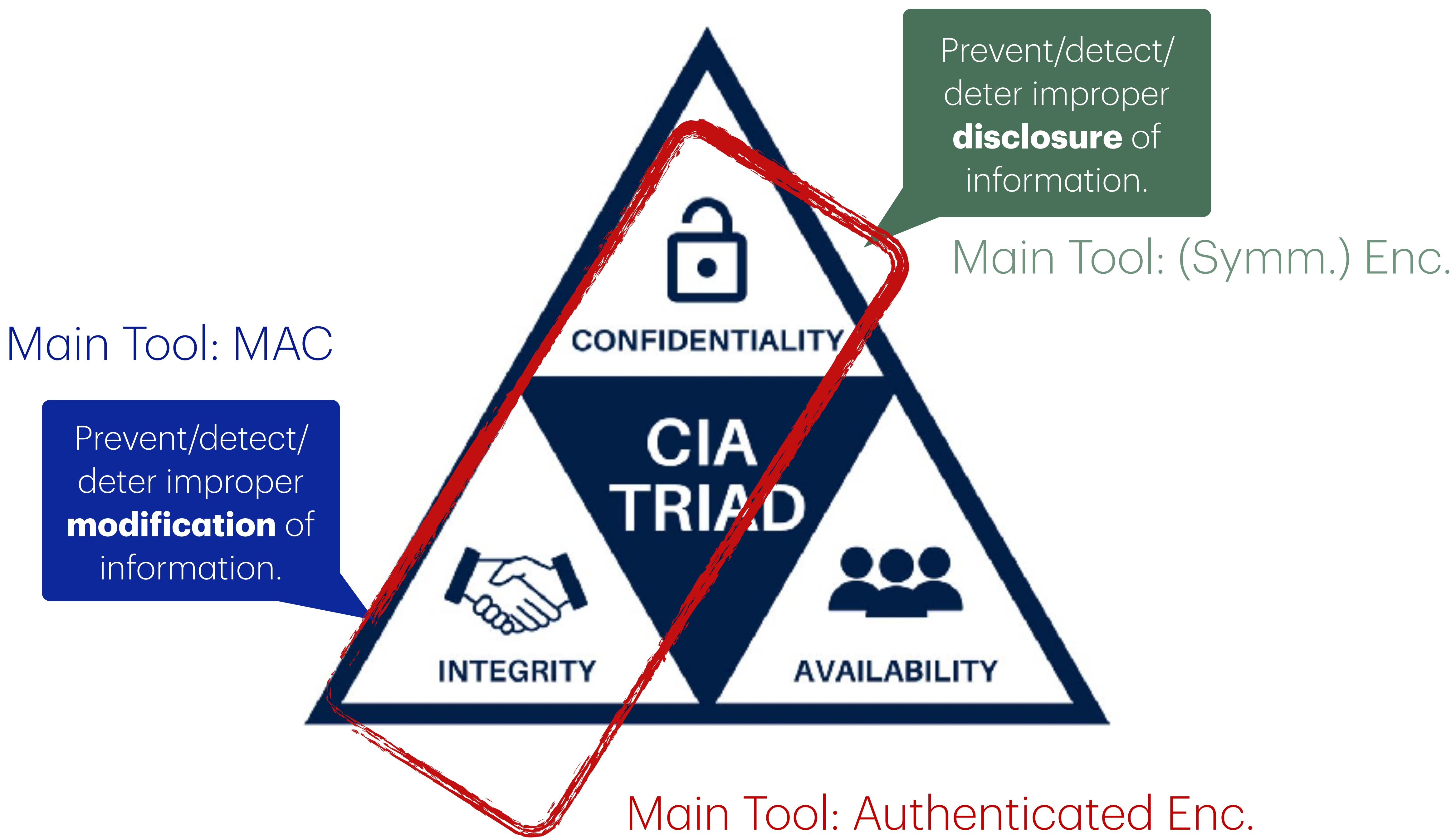
- Symmetric ciphers
  - Stream Ciphers ≈ PRG + OTP
  - Block Ciphers: workhorse for building crypto tools
    - Design principles
    - SPN & Feistel Network
    - DES & AES

Today's Topic

- Message Integrity
  - MAC: Message Authentication Code
  - Hash function
- Authenticated Encryption: confidentiality + integrity
  - Construction
  - Attacks

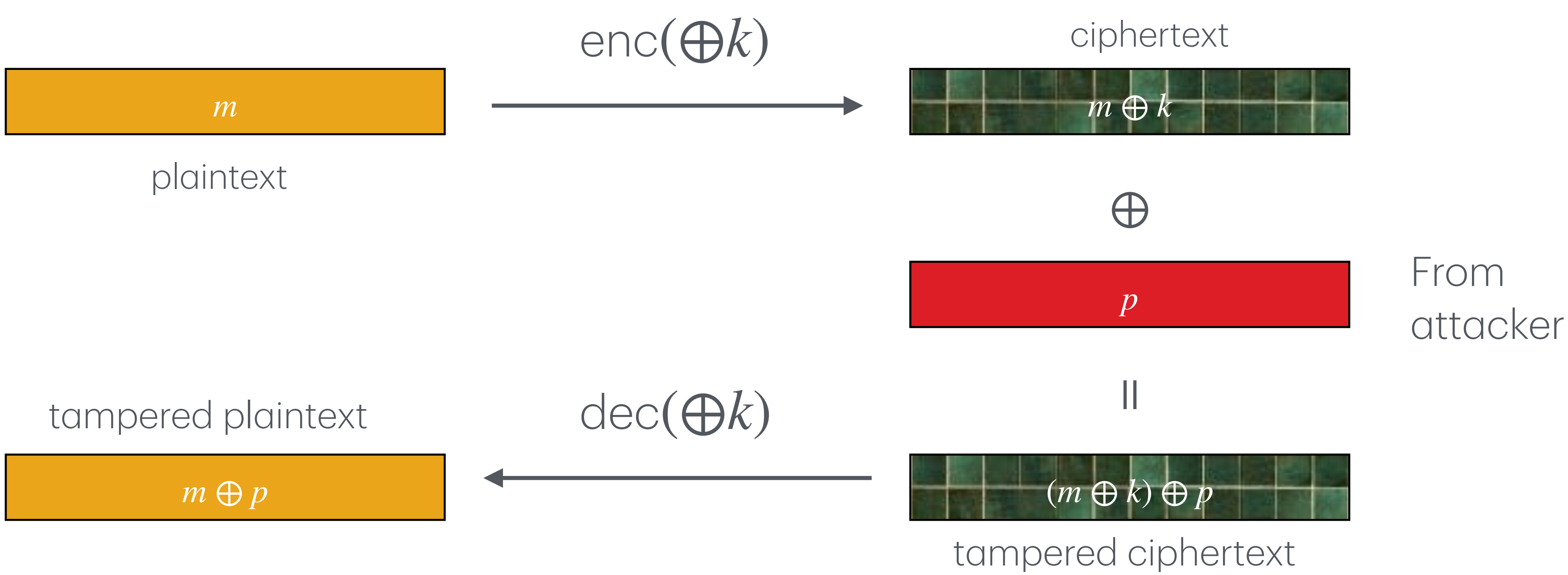
MAC: Message Authentication Code

Message Integrity



Recall: CPA-secure does not offer integrity

- Example: "Secure" stream cipher
  - CPA-secure: a **passive** attacker (eavesdropper) cannot recover plaintext / key
  - Insecure against an **active** attacker



Message Authentication Code (MAC)

MAC: a cryptographic primitive (S, V) over $(\mathscr{K}, \mathscr{M}, \mathscr{T})$ for protecting integrity

- In addition to the message itself, another token that authenticates the message, often called a *tag*, is transmitted.
- It can be used with or without encryption

Security of MAC

MAC constructions *requires* a **shared secret key**

- A MAC cannot be computed (or verified) without the key
- Different from CRC, which only corrects *random* errors.

Attacker goal: Let server output $b = 1$

$m_1, m_2, \ldots, m_q \in X$

$t_i = S(k, m_i), i = 1, \ldots, q$

a new (m, t)

$$\begin{cases} b = 1 \text{ if } V(k, m, t) = \text{true and } (m, t) \notin \{(m_i, t_i), i = 1, \ldots, q\} \\ b = 0 \text{ otherwise} \end{cases}$$

MAC is **secure** if $\Pr[b = 1]$ is negligible

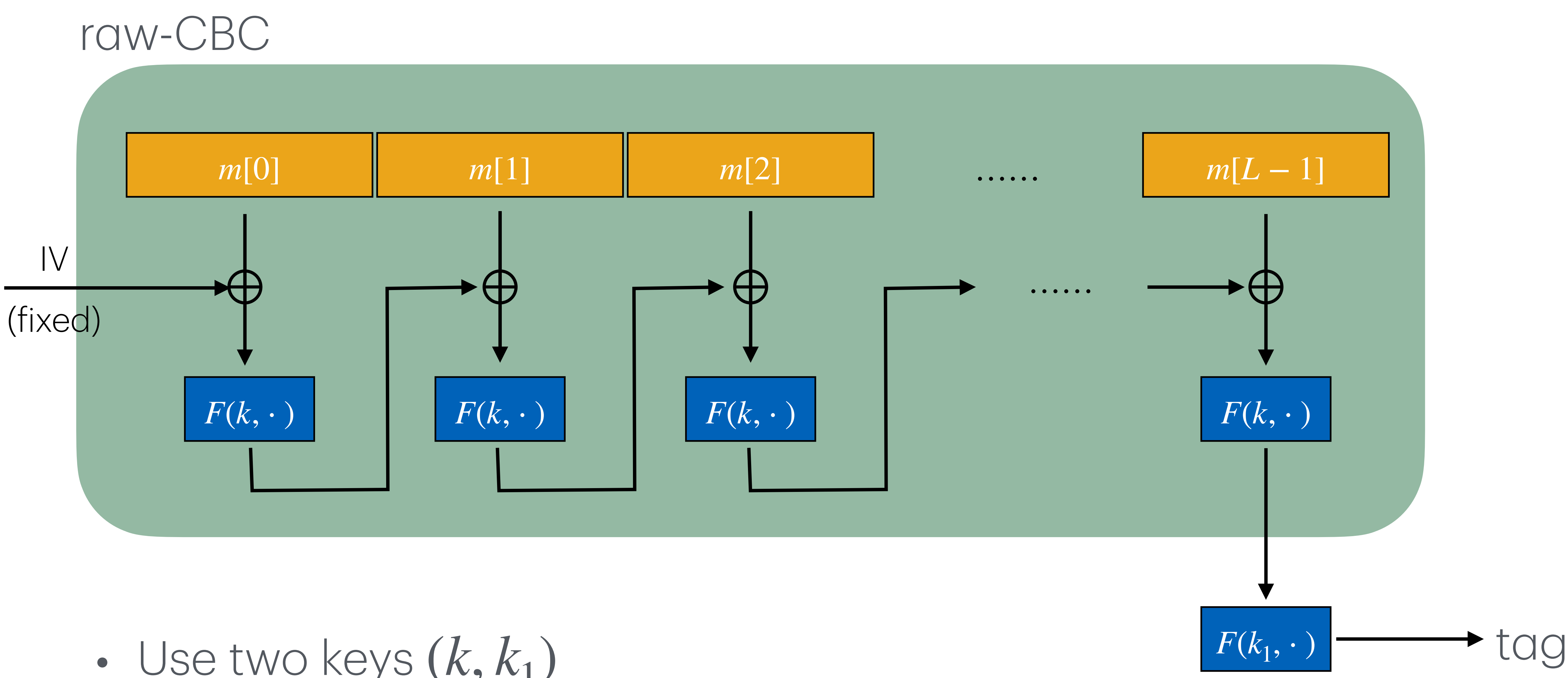
Security of MAC

- **Theorem** (informal): A secure pseudo-random function $F : K \times X \mapsto Y$ is a secure MAC if $|Y|$ is large enough.
  - Just think of F as a secure block cipher: e.g. AES: a MAC for *16-byte* messages
  - $|Y|$ must be large: otherwise the attacker can just guess.
- **Question**: How to convert a small MAC (e.g. AES) to a big MAC for arbitrarily long msg?

MAC Examples

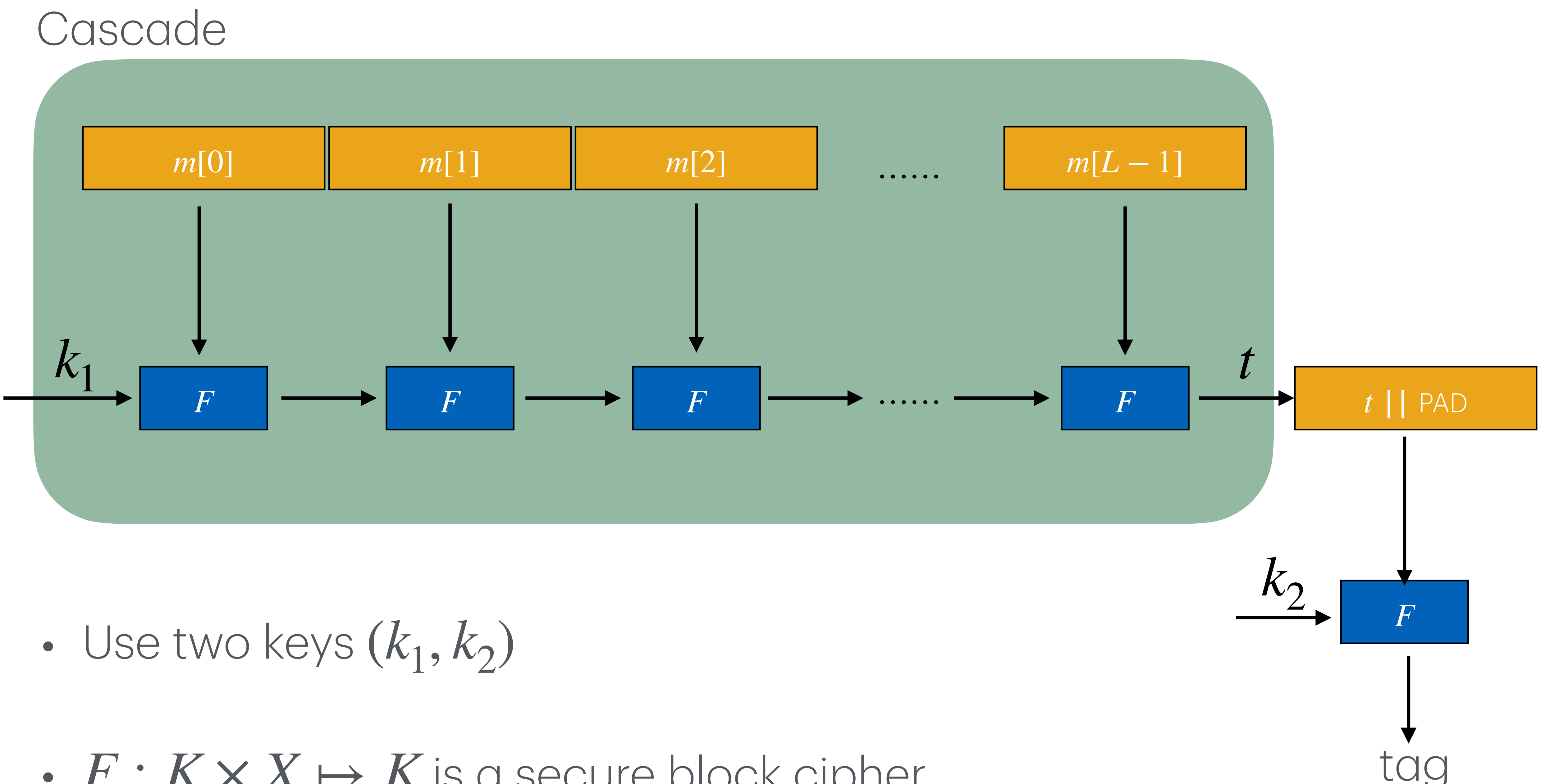
- AES: a MAC for *16-byte* messages
- *Convert small-MAC to big-MAC?*
    - Encrypted CBC-MAC (banking - ANSI X9.9, X9.19, FIPS 186-3)
    - Nested-MAC (NMAC): basis for HMAC.
    - Parallel MAC (PMAC): suitable for parallel comp.
    - CMAC: variant of CBC-MAC. Also NIST standard.
    - HMAC (Internet protocols: SSL, IPsec, SSH): next section.

Encrypted CBC-MAC (ECBC-MAC)



- Use two keys (k, k_1)
- $F : K \times X \mapsto X$ is a secure block cipher

Nested MAC (NMAC)



- Use two keys (k_1, k_2)
- $F : K \times X \mapsto K$ is a secure block cipher

Security of ECBC-MAC & NMAC

- Example: consider AES-128 as the secure block cipher $F : K \times X \mapsto X$, i.e. $K = X = \{0,1\}^{128}$
  - ECBC-MAC: secure as long as #msgs $\ll |X|^{1/2} = 2^{64}$
  - NMAC: secure as long as #msgs $\ll |K|^{1/2} = 2^{64}$
- The bound essentially comes from Birthday attack
- Once a collision occurs for the (AES-based) MAC, it's easy to forge new (m, t) pairs based on the collision

Comparison

- ECBC-MAC is commonly used as an AES-based MAC
  - CCM encryption mode (used in 802.11i)
  - NIST standard called CMAC
- NMAC not usually used with AES or 3DES
  - Main reason: need to change AES key on every block
  - Requires re-computing AES key expansion
- But NMAC is the basis for a popular MAC called HMAC (next)

Hash Function

Collision Resistance

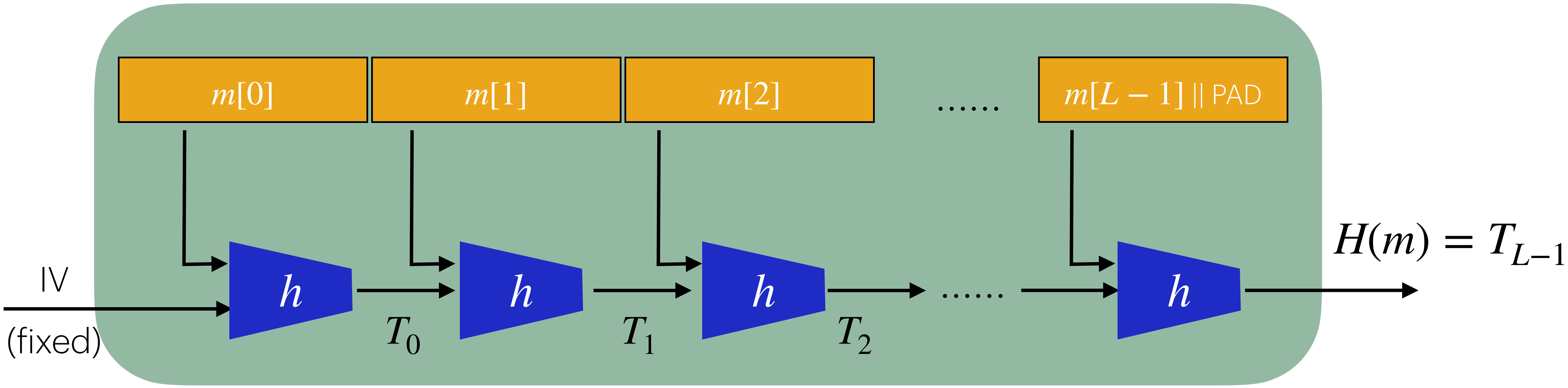
- Let $H : M \mapsto T$ be a (hash) function ($|M| \gg |T|$)
- A *collision* for H is a pair $m_0, m_1 \in M$ such that: $H(m_0) = H(m_1)$ and $m_0 \neq m_1$
- A function H is **collision resistant** if for all computational-bounded adversary A: $\Pr[A$ finds a collision for $H]$ is negligible
- Example: SHA-256 (outputs 256 bits)

MAC from Collision Resistance

- Let $I = (S, V)$ be a MAC for **short** messages over (K, M, T) (e.g. AES)
- Let $H : M^{\text{long}} \mapsto M$ be a collision-resistant hash func
- Define MAC $I^{\text{long}} = \left(S^{\text{long}}, V^{\text{long}}\right)$ over $\left(K, M^{\text{long}}, T\right)$ as
  - $S^{\text{long}}(k, m) = S(k, H(m))$; $V^{\text{long}}(k, m) = V(k, H(m), t)$
- **Theorem**: If I is a secure MAC and H is collision resistant, then I^{long} is a secure MAC
- Example: $S(k, m) = AES_{\text{2-block-CBC}}(k, \text{SHA-256}(m))$ is a secure MAC

The Merkle-Damgard Construction

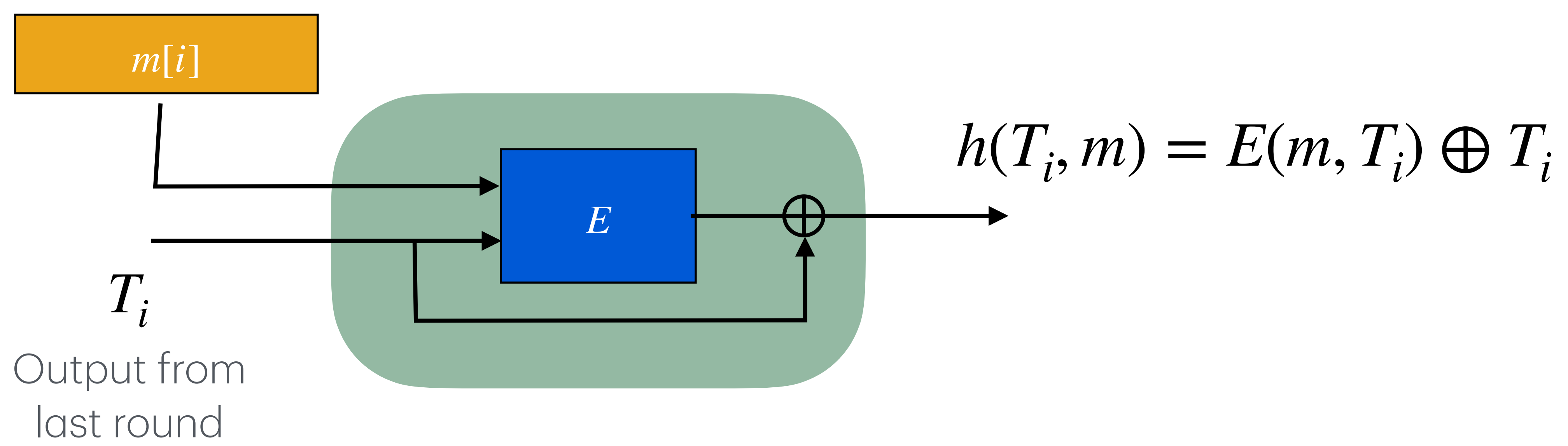
Hash func construction



- $h : T \times X \mapsto T$: a given "compression function". A hash function for short msgs.
- $H : X^{\leq L} \mapsto T$: hash func for arbitrarily long msgs
- **Theorem**: if h is collision resistant then so is H

Davies-Meyer Compression Fun.

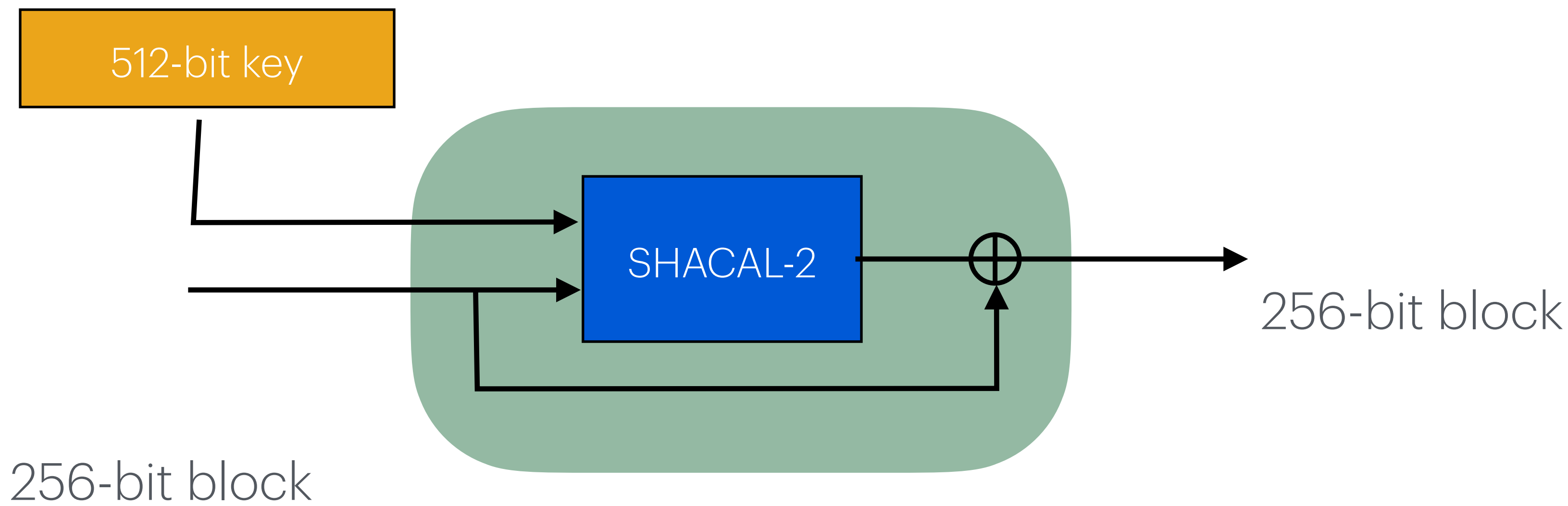
Compression Func. from Block Ciphers



- $E : K \times T \mapsto T$: a given block cipher.
- $h : T \times M \mapsto T$: compression function. Note in the construction the msg block $m[i]$ is used as key for the block cipher.
- **Theorem**: if E is a secure block cipher with $T = \{0,1\}^n$, then finding collision for h takes $\Theta(2^{n/2})$ evaluations of E.

Example: SHA-256

Compression function:



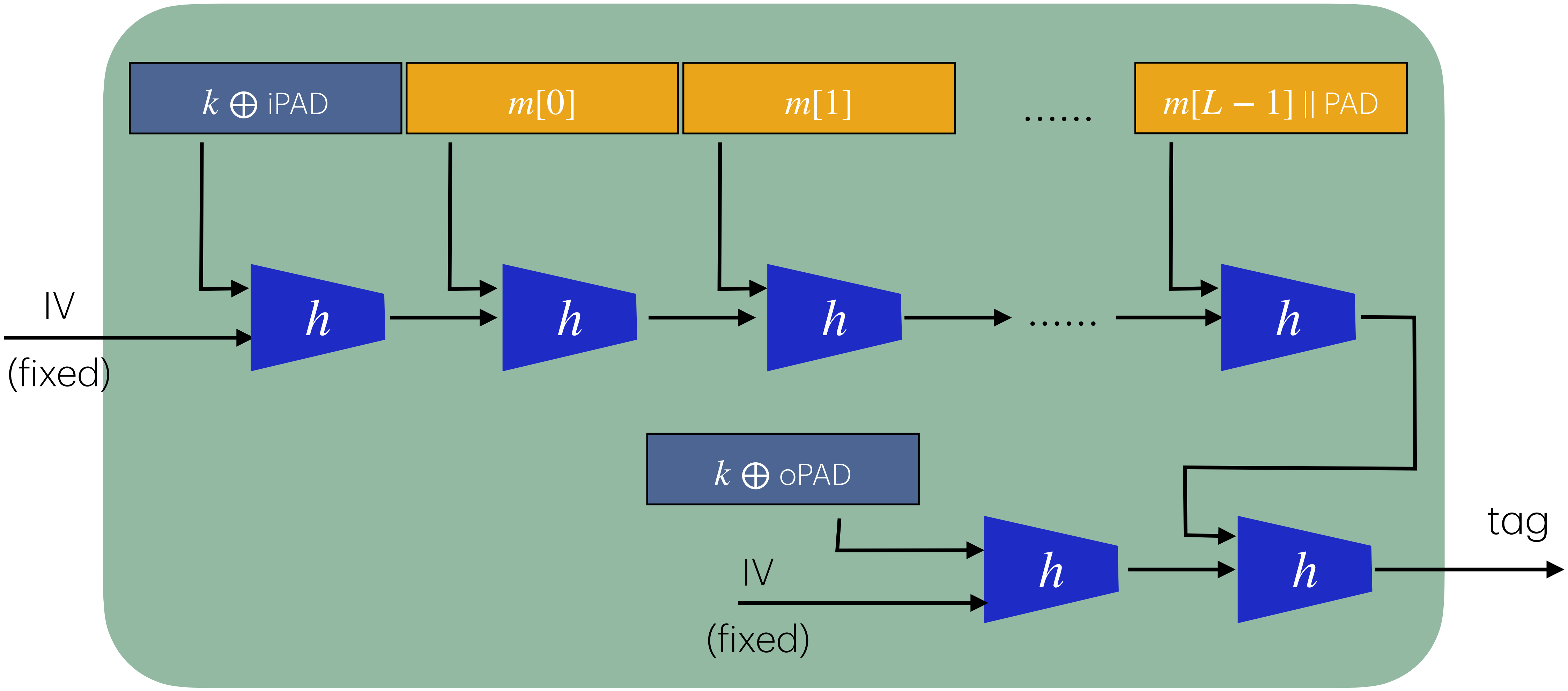
- Merkle-Damgard function
- Davies-Meyer compression function
- Block cipher: SHACAL-2

HMAC: Hash-MAC

- Most widely used MAC on the Internet
- "H": hash functions, e.g. SHA-256
- Building a MAC out of a hash function:
  - $S(k, m) = H\left(k \oplus \text{oPAD} \mid\mid H\left(k \oplus \text{iPAD} \mid\mid m\right)\right)$

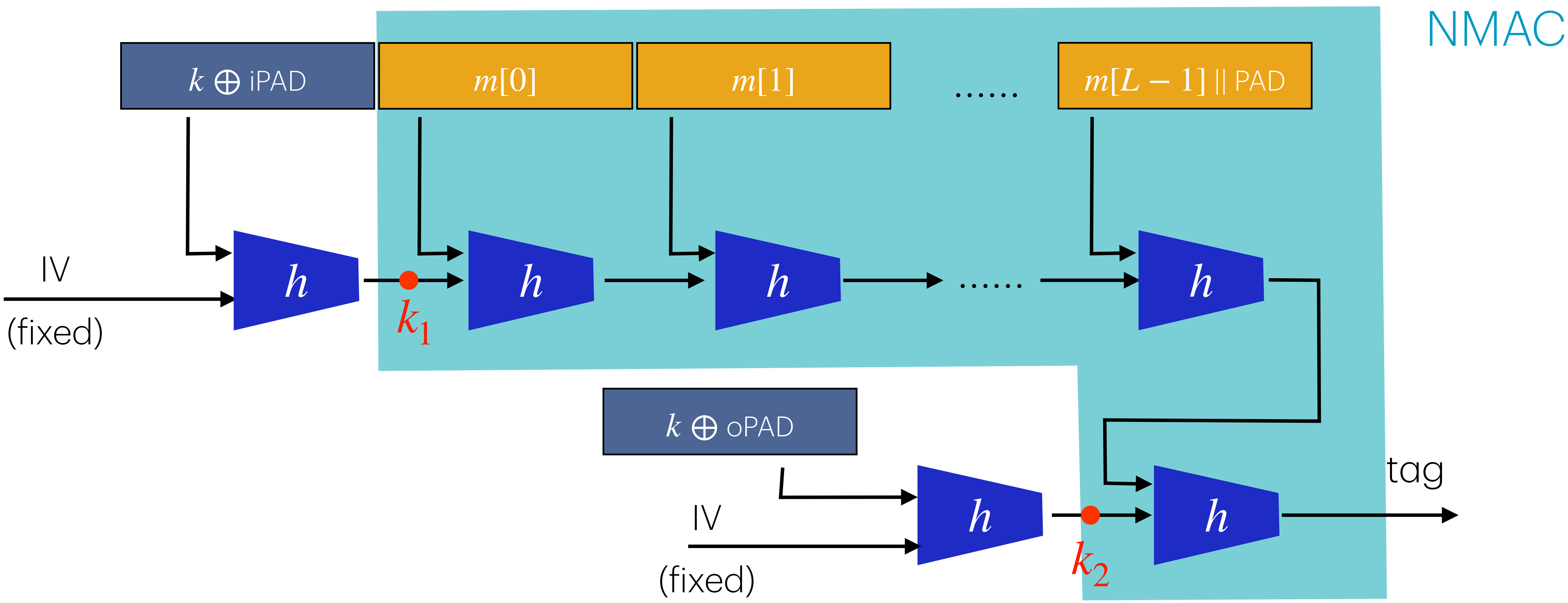
HMAC: Hash-MAC

$$S(k,m) = H\left(k \oplus \text{oPAD} \;||\; H\left(k \oplus \text{iPAD} \;||\; m\right)\right)$$



HMAC: Hash-MAC

$$S(k, m) = H\left(k \oplus \text{oPAD} \;||\; H\left(k \oplus \text{iPAD} \;||\; m\right)\right)$$



Similar to NMAC:

- main difference: the two keys k_1, k_2 are dependent.
- similar security bounds: need #blocks $q \ll |T|^{1/2}$ (e.g. 2^{128} for SHA-256)

Birthday Attack on Hash Functions

Let $H : M \mapsto \{0,1\}^n$ be a collision-resistant hash function

Birthday Attack

1. Choose $2^{n/2}$ *random* elements: $m_1, \ldots, m_{2^{n/2}} \in M$
2. Compute hashes for all: $t_i = H(m_i), i = 1,\ldots,2^{n/2}$
3. Look for collision ($t_i = t_j, i \neq j$). If not found, go to step 1

Expected number of iterations ≈ 2

Running time $O(2^{n/2})$, space $O(2^{n/2})$

Verification Timing Attack

Example: Keyczar crypto library (Python) [simplified]

```
def Verify(key, msg, sig_bytes):
    return HMAC(key, msg) == sig_bytes
```

The problem: '==' implemented as a byte-by-byte comparison

- Comparator returns false when first inequality found

Verification Timing Attack

Timing attack: to compute tag for target message m do:

1. Query server with random tag
2. Loop over all possible first bytes and query server.
   - Stop when verification *takes a little longer* than in step 1
3. repeat for all tag bytes until valid tag found

Authenticated Encryption

Where are we now

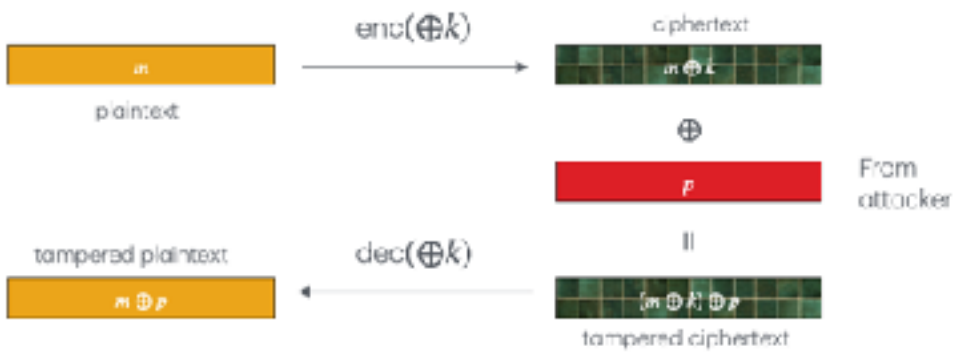
- CPA security cannot guarantee secrecy under active attacks.
- CPA-secure cipher: Confidentiality but no integrity
- MAC: Integrity but no confidentiality
- If message needs both Integrity and Confidentiality?
  - Authenticated Encryption !

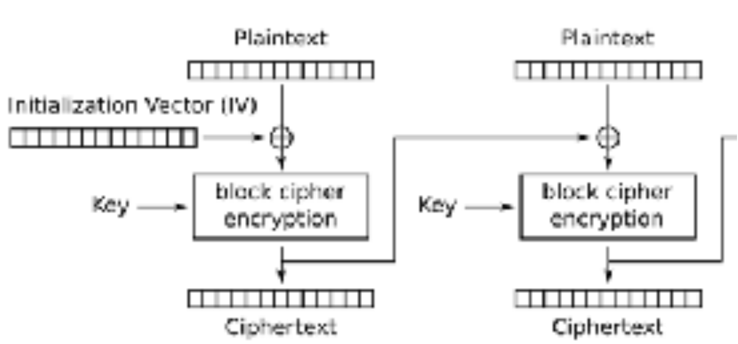
Authenticated Encryption

- An Authenticated Encryption system (E, D) is a cipher where
  - As usual, Enc $E : K \times M \mapsto C$
  - But, Dec $D : K \times C \mapsto M \cup \{ \perp \}$, where "⊥" means the ciphertext is rejected
- Security:
  - **CPA-Secure**, as usual, and
  - **Ciphertext integrity**: attacker cannot create new ciphertexts that decrypt properly

Authenticated Encryption

- **Ciphertext Integrity (C.I.):** attacker cannot fabricate valid ciphertext *that he has not seen before.*
  - An Auth. Enc. system should always decrypt such ciphertexts to $\perp$
  - *Bad examples*: all ciphers we have learned till now
    - Particularly bad: CBC with random IV; Stream cipher.
    - Not only no integrity check, but plaintext can also be tampered in predictable way by XORing ciphertext with desired pattern.





Authenticated Encryption

- **Ciphertext Integrity (C.I.):** attacker cannot fabricate valid ciphertext *that he has not seen before.*
  - Implications:
    - Authenticity: Attacker cannot fool the receiver that the ciphertext is from some particular sender.
    - Security against Chosen Ciphertext Attacks (CCA)
  - Limitations:
    - Does not prevent *replay attacks*
    - Does not account for *side channels* (timing)

Chosen-Ciphertext Attack (CCA)

- Adversary's power: both CPA and CCA
  - Can obtain the encryption of arbitrary messages of his choice
  - Can decrypt any ciphertext of his choice, *other than the ones he has already submitted*.
- (conservative modeling of real life)

Chosen-Ciphertext Attack (CCA)

CCA-Security

b **?**

(1) CPA queries:

$m_0^i, m_1^i \in M : |m_0^i| = |m_1^i|, i = 1, \ldots, q$

$c_i = E(k, m_b^i), i = 1, \ldots, q$

(2) CCA queries:

$c_i \in C : c_i \notin \{c_1, \ldots, c_q\}$

$m_i = D(k, c_i), i \notin \{1, \ldots, q\}$

Server has
a secret bit
$b \in \{0,1\}$

CCA-secure:
Attacker cannot infer
b significantly better
than random guess
after multiple rounds
of interaction.

Auth. Enc. system is CCA-secure

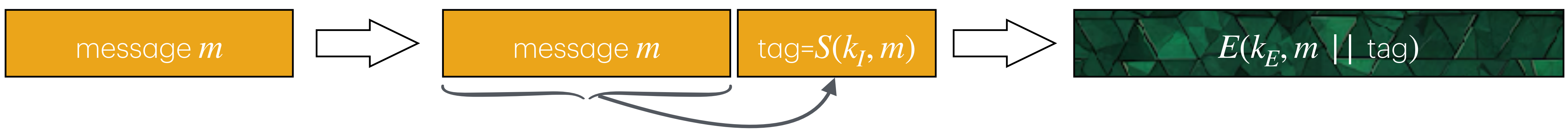
History of Auth. Enc.

- Authenticated Encryption (AE): introduced in 2000 [KY'00, BN'00]
- Crypto APIs before then: (e.g. MS-CAPI)
  - Provide API for CPA-secure encryption (e.g. CBC with rand. IV)
  - Provide API for MAC (e.g. HMAC)
- Every project had to combine the two itself without a well defined goal
  - Not all combinations provide AE …

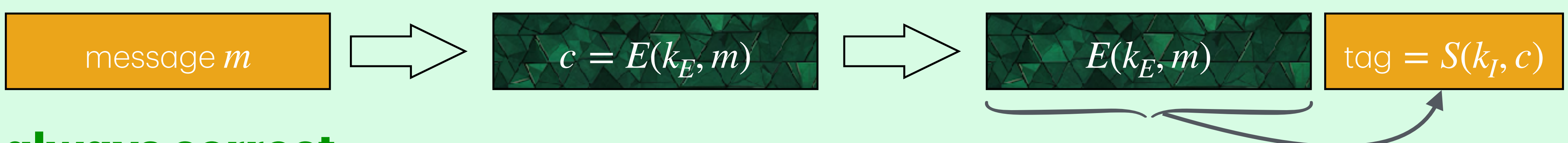
Combining MAC and Encryption

Given Encryption key k_E, MAC key k_I

Option 1: MAC-then-Enc (SSL)

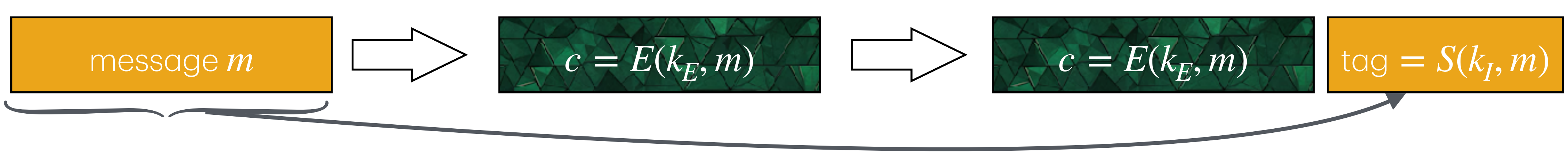


Option 2: Enc-then-MAC (IPsec)



always correct

Option 3: Enc-and-MAC (SSH)



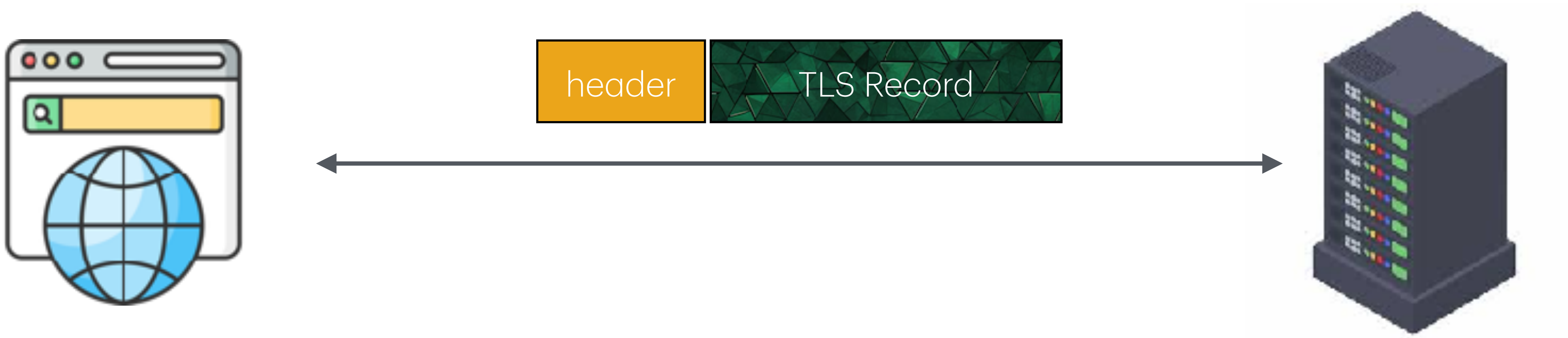
Combining MAC and Encryption

- Let (E, D) be CPA secure cipher and (S, V) secure MAC. Then:
- **Encrypt-then-MAC**: always provides Authenticated Enc.
- **MAC-then-Encrypt**: may be insecure against CCA attacks
  - However: when (E, D) is rand-CTR mode or rand-CBC, MAC-then-Enc provides Auth. Enc.

Standards

- GCM: CTR mode encryption then CW-MAC (not covered in this lecture)
  - Accelerated via Intel's PCLMULQDQ instruction
- CCM: CBC-MAC then CTR mode encryption (802.11i)
- EAX: CTR mode encryption then CMAC
- All support AEAD: (Auth. Enc. with Associated Data). All are nonce-based.

Case Study: TLS 1.1 Record Layer



shared sk $k_{b\to s}$, $k_{s\to b}$

shared sk $k_{b\to s}$, $k_{s\to b}$

Unidirection keys: $k_{b\to s}$ and $k_{s\to b}$

- Stateful encryption
  - Each side maintain two 64-bit counters: $\mathrm{ctr}_{b\to s}$, $\mathrm{ctr}_{s\to b}$
  - Initialized to 0 when session started. Ctr++ for every record.
  - Purpose: defend against replay attack

Case Study: TLS 1.1 Record Layer

Encryption: CBC AES-128, HMAC-SHA1

$k_{b \to s} = (k_{\text{MAC}}, k_{\text{ENC}})$

Browser side: ENC $\left(k_{b \to s}, \text{data}, \text{ctr}_{b \to s}\right)$

1. tag $\leftarrow S\left(k_{\text{MAC}},\ [++\,\text{ctr}_{b \to s} \mid\mid \text{header} \mid\mid \text{data}]\right)$

2. Pad [header || data || tag] to AES block size (Note ctr is not transmitted)

3. CBC encrypt with k_{ENC} and new random IV.

4. Prepend header & IV

Case Study: TLS 1.1 Record Layer

Decryption: CBC AES-128, HMAC-SHA1

$k_{b \to s} = \left(k_{\mathrm{MAC}}, k_{\mathrm{ENC}}\right)$

Server side: DEC $\left(k_{b \to s}, \text{data}, \text{ctr}_{b \to s}\right)$

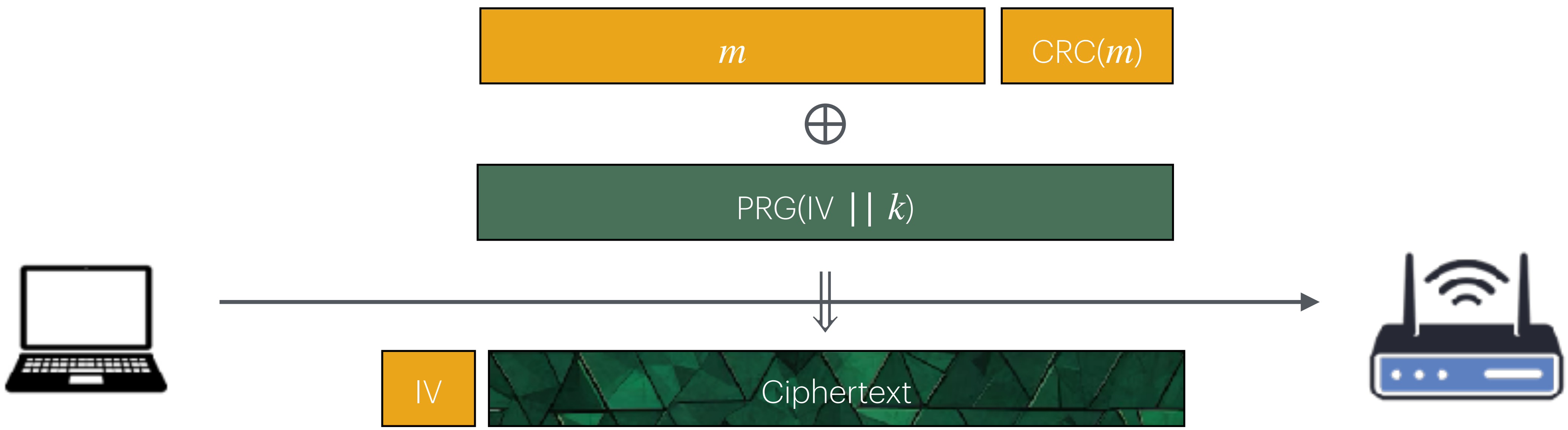
1. CBC decrypt with k_{ENC}.
2. Check pad format: send decryption_failed if invalid
3. Verify tag $V\left(k_{\mathrm{MAC}},\ \left[++\,\text{ctr}_{b \to s} \ || \ \text{header} \ || \ \text{data}\right], \text{tag}\right)$; send bad_record_mac if invalid

The two different failure return value leaks plaintext info! (see next sec.)

Attacks on Authenticated Encryption

Attack insecure MACs

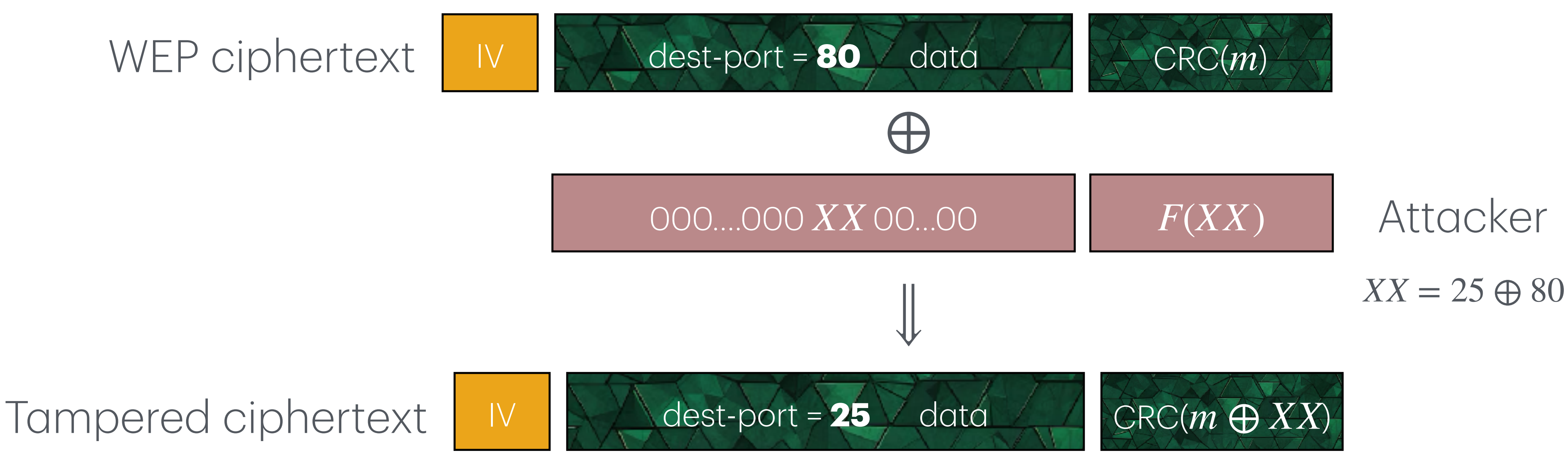
802.11b WEP



- Recall: Encryption using RC4 stream cipher
- Problem: CRC is ***not*** a cryptographic MAC
  - $\forall m, p$, $\text{CRC}(m \oplus p) = \text{CRC}(m) \oplus F(p)$
  - F is a public & easily computed function

Attack insecure MACs

802.11b WEP

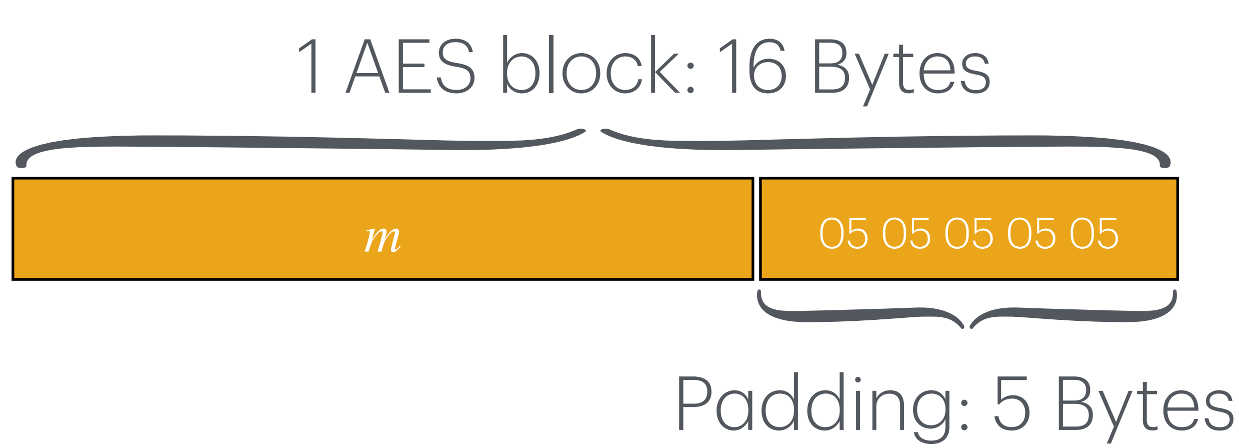


Upon decryption: CRC is valid but Ciphertext is changed.

Attack using Padding Oracle

The TLS (1.1) record protocol (CBC Encryption)

- Recall **AES-CBC Padding**
  - Main purpose: make msg length an integral multiple of block length.
  - Format:
    - When padding $i > 0$ bytes, fill each of the i bytes with value i.
    - If msg len is already a multiple of 16 bytes: add one dummy block with all byte value 16



Attack using Padding Oracle

The TLS (1.1) record protocol (CBC Encryption)

- Recall TLS 1.1 decryption: DEC $\left(k_{b \to s}, \text{data}, \text{ctr}_{b \to s}\right)$

   1. CBC decrypt with k_{ENC}.

   2. Check pad format: send decryption_failed if invalid

   3. Verify tag $V\left(k_{\text{MAC}}, \left[++\,\text{ctr}_{b \to s} \mid\mid \text{header} \mid\mid \text{data}\right], \text{tag}\right)$;
   send bad_record_mac if invalid

- **Padding oracle**: attacker submits ciphertext and learns if last bytes of plaintext are a valid pad

Attack using Padding Oracle

Padding Oracle from Timing

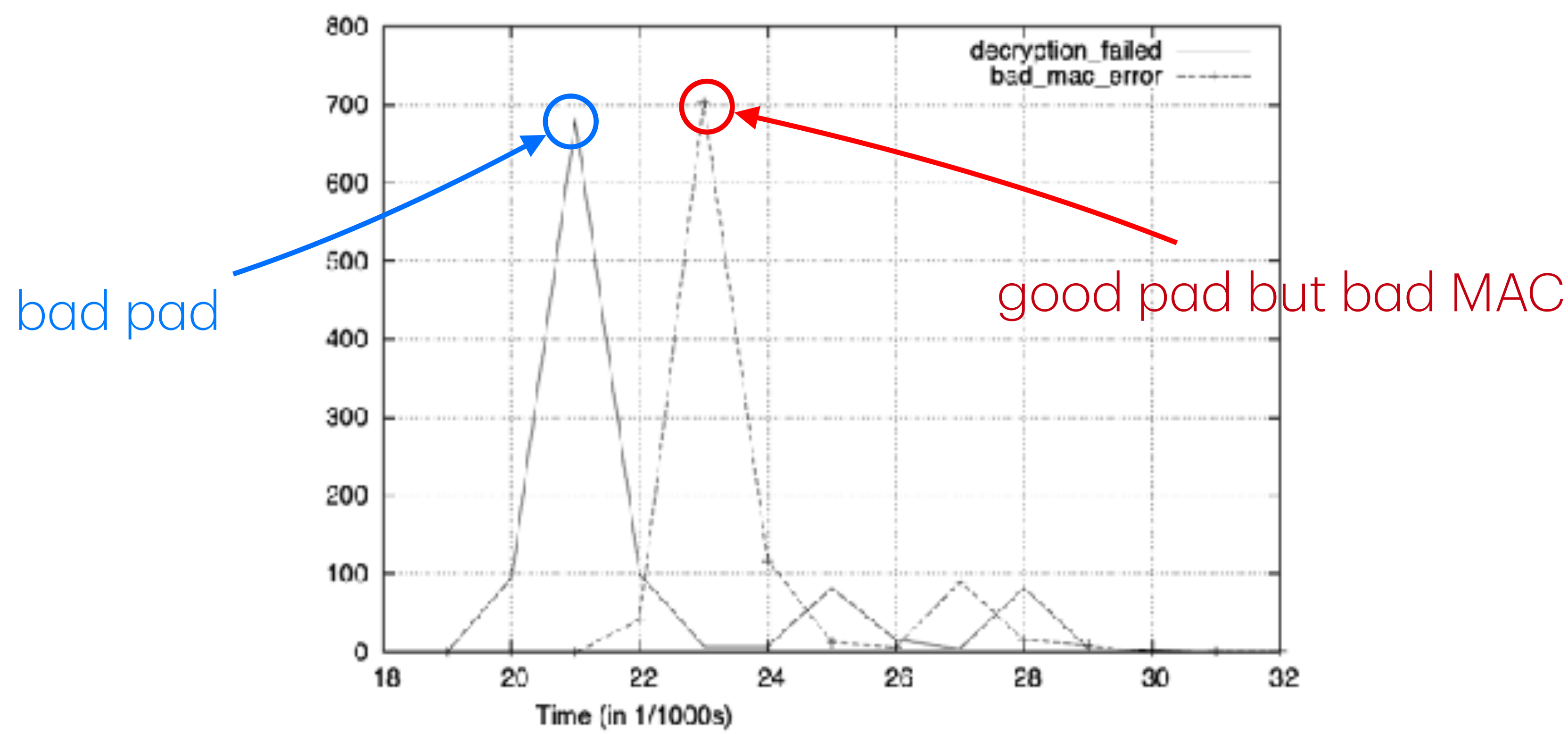


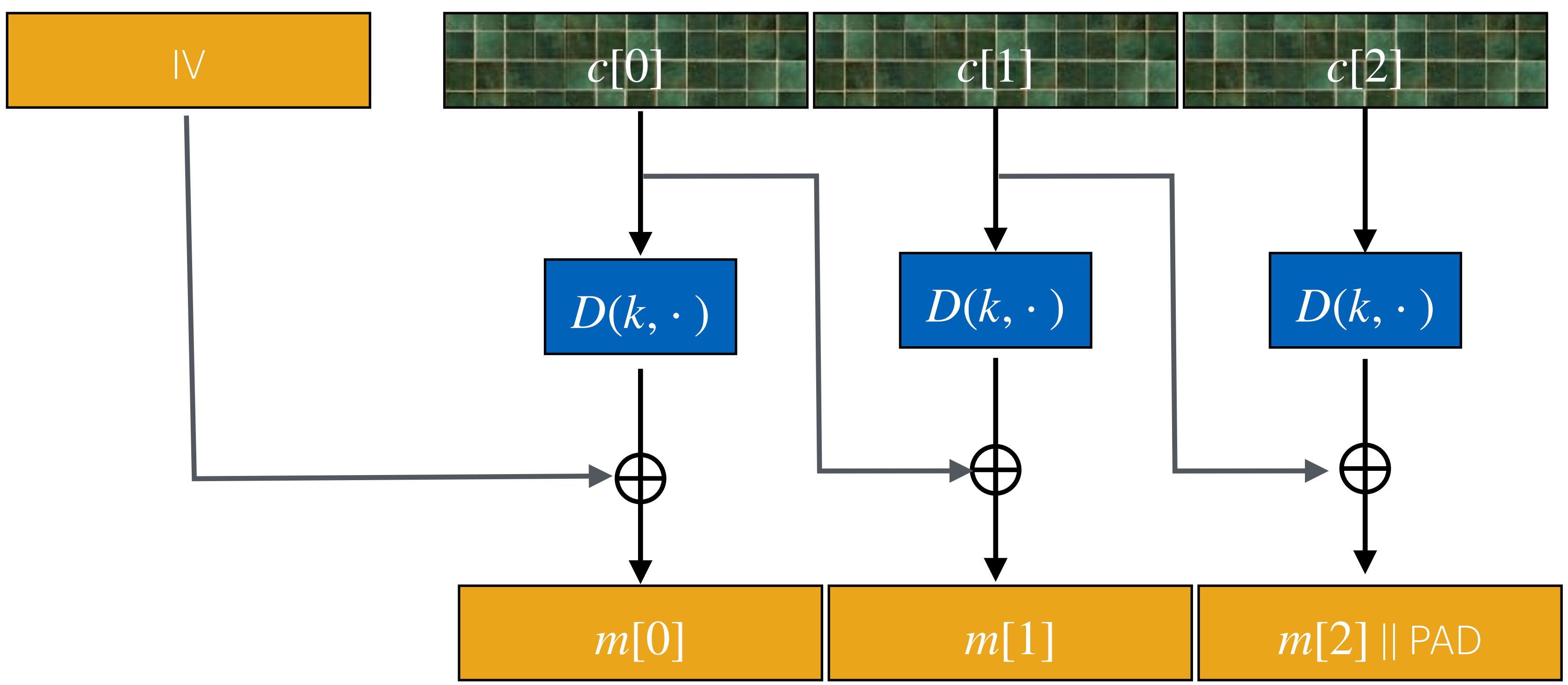
Fig. 3. Distribution of the number of decryption failed and bad mac error error messages with respect to time.

- Even with a same return value, Padding oracle can be obtained via measuring response time. [Canvel-Hiltgen-Vaudenay-Vuagnoux'2003]
- Fixed in OpenSSL 0.9.7a

Attack using Padding Oracle

Using the Padding Oracle (against CBC encryption)

Recall the decryption procedure of CBC mode: $c[i-1]$ is XORed with $D(k, c[i])$ to get $m[i]$

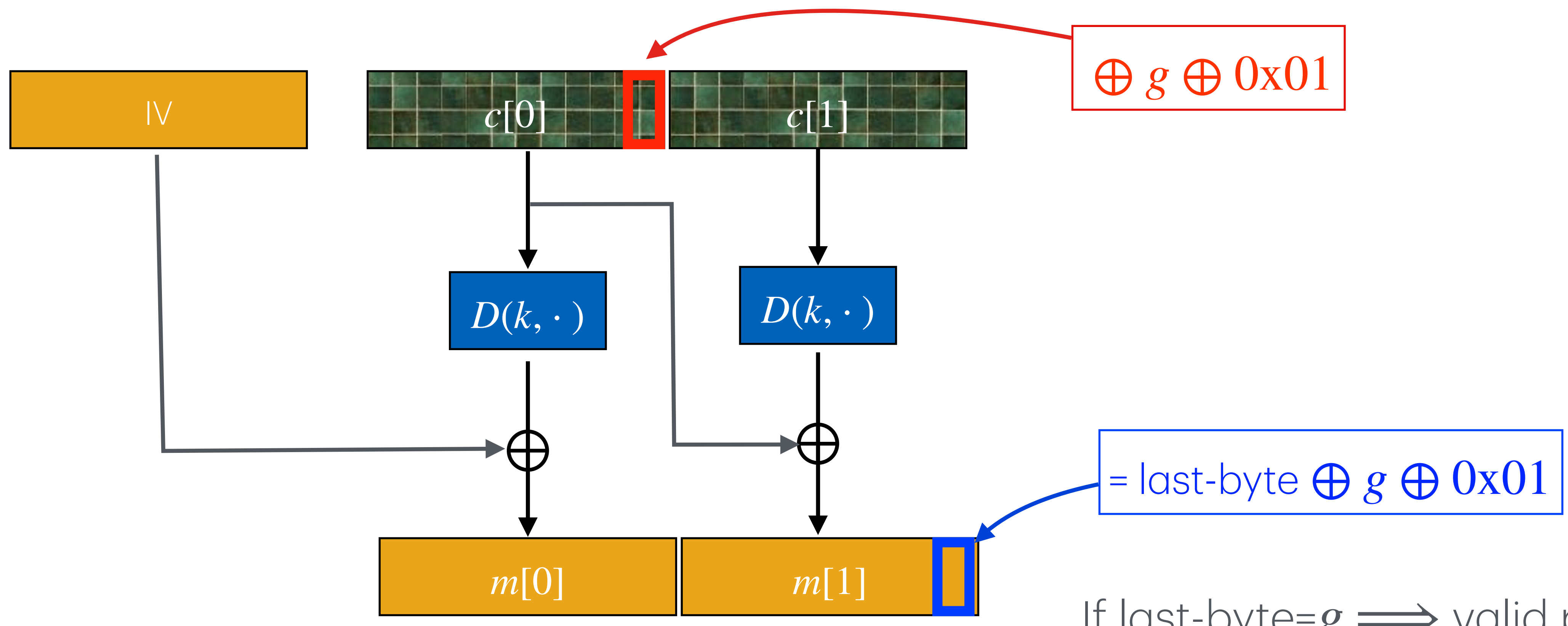


Suppose attacker has $(c[0], c[1], c[2])$ and wants $m[1]$

Attack using Padding Oracle

Using the Padding Oracle (against CBC encryption)

Let g be attacker's guess for the last byte of $m[1]$



If last-byte=$g \Longrightarrow$ valid pad;
Otherwise, invalid pad.

- Repeat with $g = 0,1,\ldots,255$ to learn $m[1]$'s last byte.

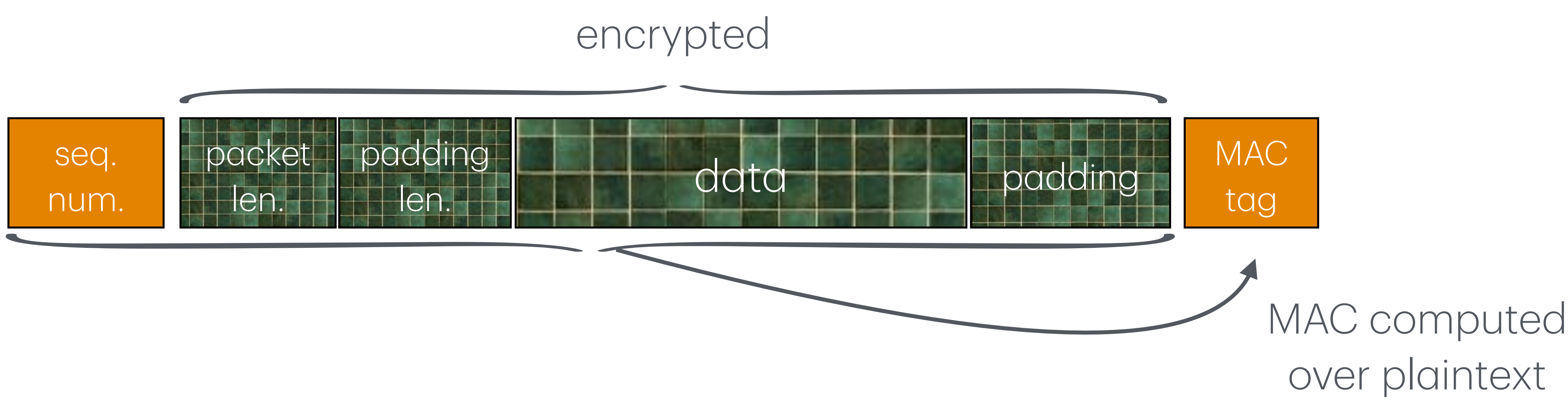
- Then use a $(0x02, 0x02)$ pad to learn the next byte …

Lessons

- Encrypt-then-MAC will avoid this problem completely
  - MAC checked first and ciphertext discarded if invalid
- MAC-then-(CBC)Enc provides Auth. Enc., but padding oracle destroys it.
- MAC-the-(CTR)Enc can avoid the padding oracle: ‘cause it needs no padding.

Attack on Non-Atomic Decryption

SSH Binary Packet Protocol



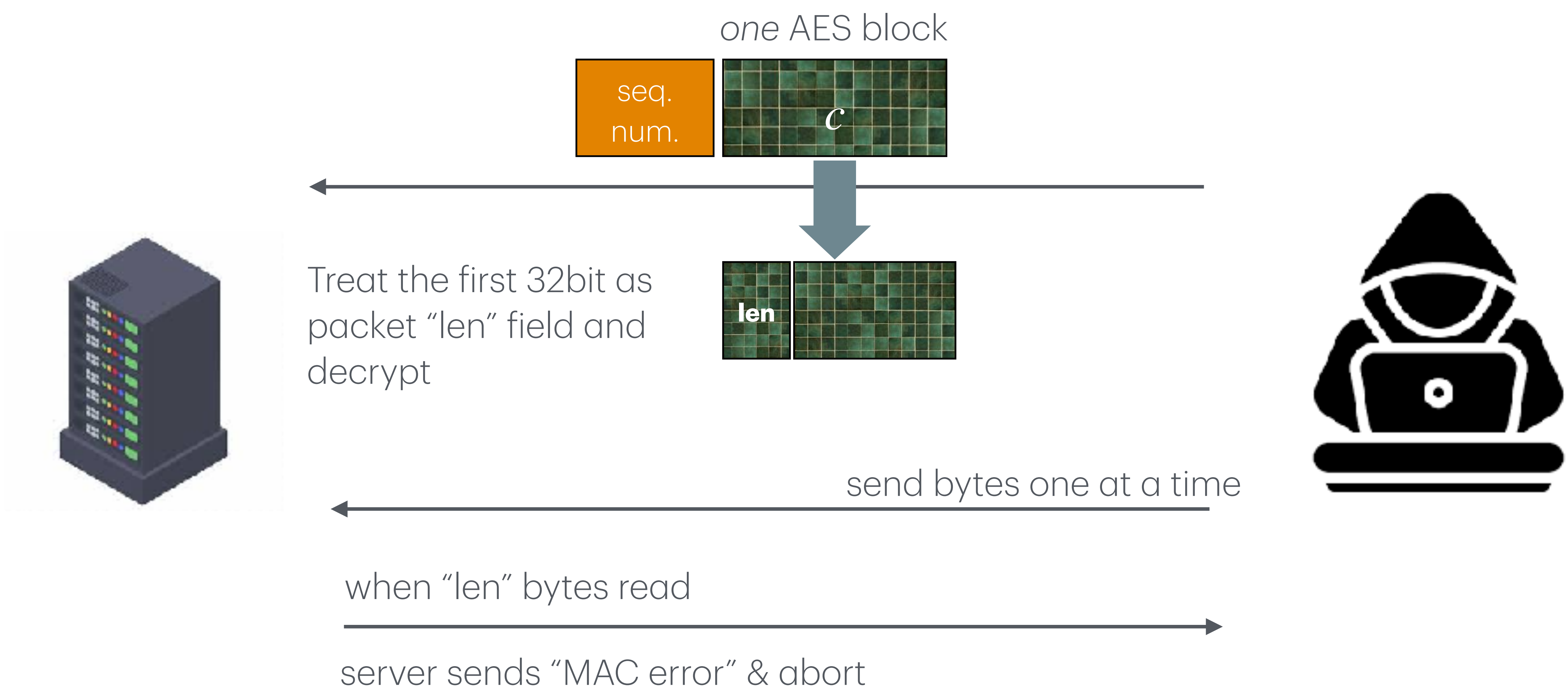
Decryption:

1. Decrypt packet length field only (!)
2. Read as many packets as length specifies
3. Decrypt remaining ciphertext blocks
4. Check MAC tag and send error response if invalid

Attack on Non-Atomic Decryption

SSH Binary Packet Protocol

Attacker has ***one*** ciphertext block $c = \text{AES}(k, m)$ and wants to recover m



$\Longrightarrow$ Attacker learns the highest 32 bits of m

Lessons

- Problems
  - Non-atomic decrypt
  - “len” field decrypted and used *before* it is authenticated
- What could be done better?
  - Send the length field unencrypted (but MAC-ed)
  - Add a MAC of (seq-num, length) right after the length field

Summary

- MAC: protects integrity (but not confidentiality)
- Hash function: collision resistance
- Authenticated Encryption
  - Encrypt-then-MAC (recommend)
  - MAC-then-Encrypt
  - Attacks
    - Padding Oracle
    - Non-atomic decrypt
    - Do not implement A.E. by yourself! Use a standard if possible.

Acknowledgement

- The slides of this lecture is developed heavily based on
  - Slides from Prof Dan Boneh's lecture on Cryptography (https://crypto.stanford.edu/~dabo/courses/OnlineCrypto/)
  - Slides from Prof Ziming Zhao's lecture on Computer Security (https://zzm7000.github.io/teaching/2023springcse410565/index.html)

Questions?