CRYPTOGRAPHY

(Lecture 4)

Literature:

"**Handbook of Applied Cryptography**" **(ch 9.0,9.5, 9.5.1, 9.75**)
"Lecture Notes on Cryptography" by S. Goldwasser and M. Bellare (**ch 9.0,9.1,9.2, 9.8.1**)
"A Graduate Course in Applied Cryptography" by D. Boneh and V. Shoup (ch 6, 6.1, **9.0**, 9.3, **9.7**)

Module 1: Agenda

Commitment Schemes

Hash Functions

Blockchain Technology

OTP & Perfect Secrecy

Randomness in Cryptography

Semantic Security + Proof

Block Ciphers

Modes of Operation

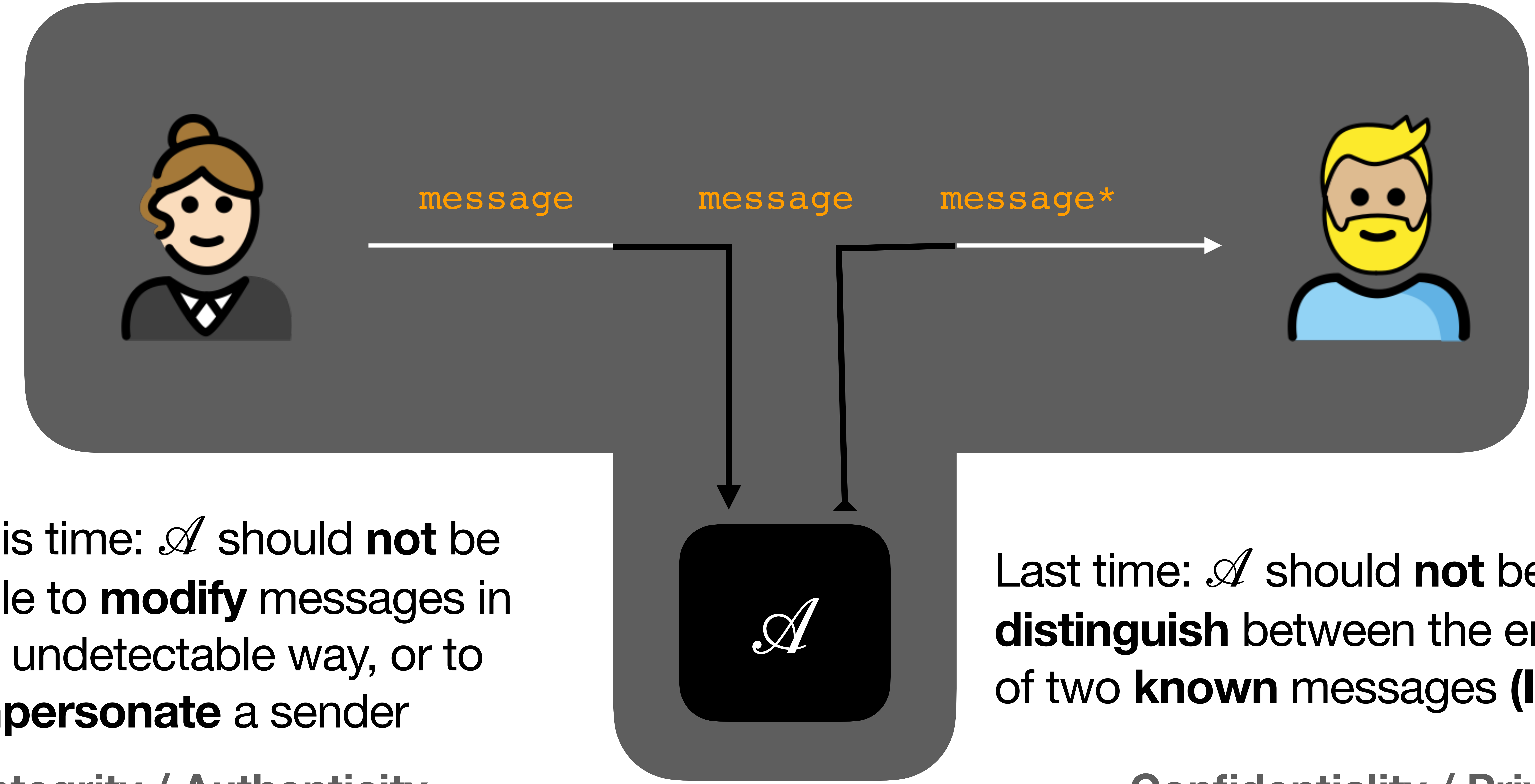
Message Authentication Codes (MAC)

- What's the Problem?
- Definition (Syntax)
- Adversary's Goals & Powers
- Security Notion
- A Construction: HMAC

Authenticated Encryption

- GCM

Secure Communication Over an Insecure Channel



This time: $\mathscr{A}$ should **not** be able to **modify** messages in an undetectable way, or to **impersonate** a sender

Integrity / Authenticity

Last time: $\mathscr{A}$ should **not** be able to **distinguish** between the encryption of two **known** messages **(IND-CPA)**

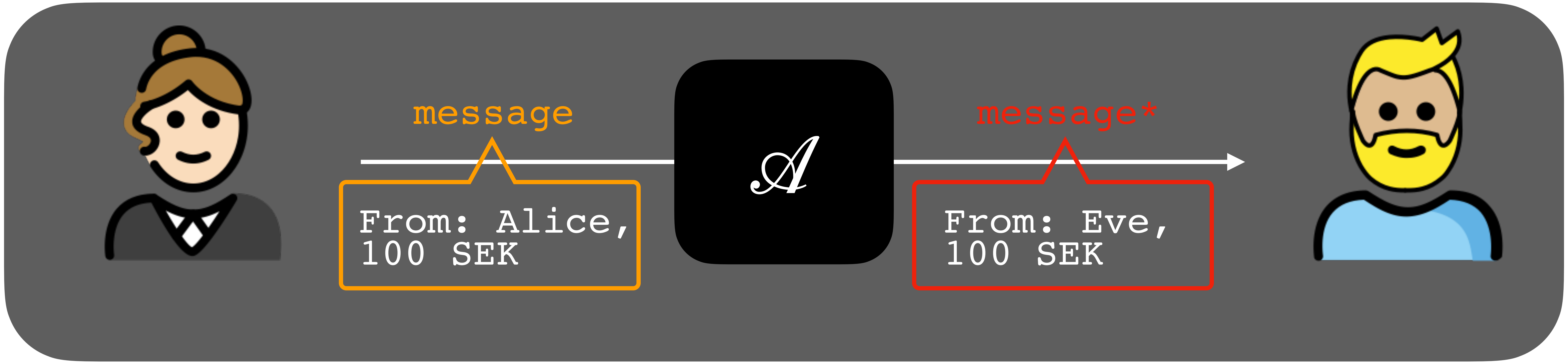
Confidentiality / Privacy

Why Does Integrity Matter?

A motivating example

Fact1: files sent over a network have well-known, **predictable headers**. A typical example is emails, which have sender (`From:`) and receiver (`To:`) info, as well as date, subject and others.

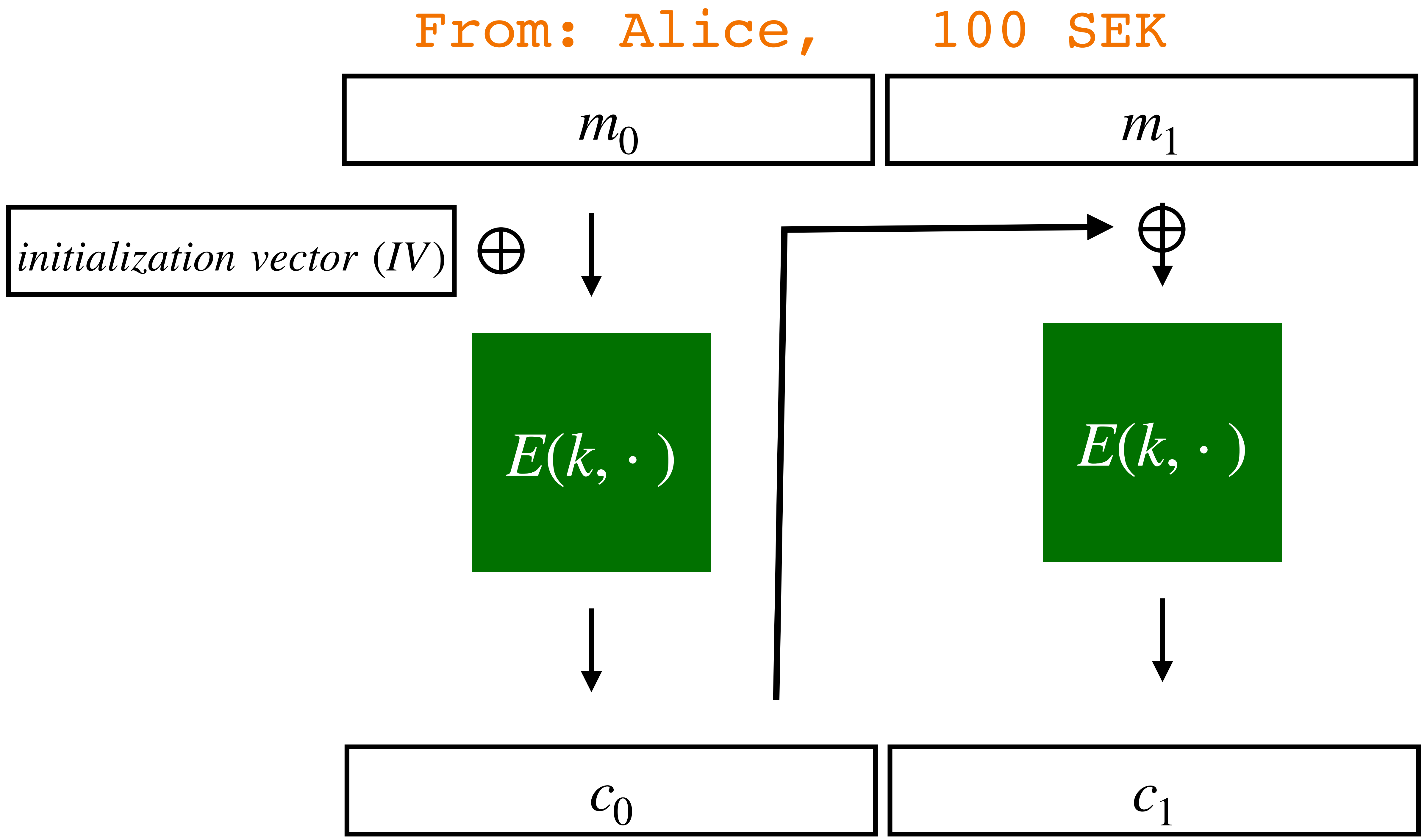
Fact2: Files are often **encrypted** in transit, so this information is not readable to the eavesdropping adversary.



This attack is trivial against AES (or any block cipher) in **CBC mode**

The adversary that launches this attack will succeed with 100% probability AND without knowing the secret key

Cipher Block Chaining Mode (CBC)



`ciphertext` = (IV, c_0, c_1)

$$c_0 = E(k, m_0 \oplus IV)$$
$$c_i = E(k, m_i \oplus c_{i-1}) \text{ for } i > 0$$

$$m_0 = D(k, c_0) \oplus IV$$
$$m_i = D(k, c_i) \oplus c_{i-1} \text{ for } i > 0$$

The Attack: $IV^* = IV \oplus \texttt{From:Alice} \oplus \texttt{From:Eve}$ `ciphertext*` = (IV^*, c_0, c_1)

🧐 *Encryption alone cannot detect the change, but Bob could. Can you see how?*

Integrity Matters. But Even More So Does Authenticating the Source of a Message

Encryption is not enough!
We need a new cryptographic primitive

Think Halloween

Message Authentication Code (MAC)

Definition: MAC

A Message Authentication Code (MAC in short) is a pair of efficient algorithms (MAC, Ver) with the following syntax:

- $MAC : \mathscr{K} \times \mathscr{M} \rightarrow \mathscr{T}$ is a probabilistic algorithm that takes in input a key k, a message m and outputs a tag t.
- $Ver : \mathscr{K} \times \mathscr{M} \times \mathscr{T} \rightarrow \{0,1\}$ is a deterministic algorithm that takes in input a key k, a message m and a tag t, and returns 1 (accept) or 0 (reject).

And satisfying the **correctness** condition:

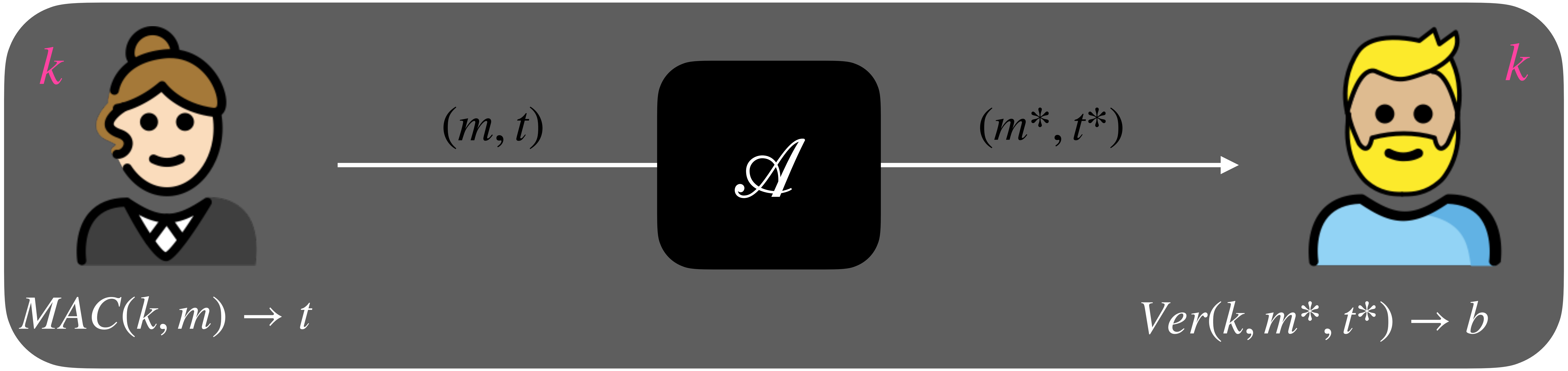
$Pr[Ver(k, m, MAC(k, m)) = 1] = 1$ for all $k \in \mathscr{K}, m \in \mathscr{M}$

Protecting Communications Over an Insecure Channel

Goals:

Encryption = prevent any third party from **understanding** the content of the communication

MAC = prevent any third party (or the channel) from **altering** the communication



$b = 1 \; if \; m^* = m \; and \; t = t^*$

$b = 0 \; if \; m^* \neq m$

*A tag t is **valid** for a message m against the key k, if $Ver(k, m, t) = 1$*

Aim: quantify the $\mathscr{A}$'s likelihood in forging a valid tag t^* for a ***new*** (different) message m^*

🧐 *What about replay attacks?*

Towards a Security Definition

$\mathscr{A}$

Adversary's Goal

~~To decrypt the communication~~ *Here we do not care about secrecy, only about integrity*

~~To recover the secret key~~ *Too strong requirement, damage can be done with less*

~~To modify the content of the communication~~ *Vague, everyone can "flip bits"*

To *produce* a *tag* for a *known message* that the receiver will deem **authentic** and that is *different* from what has been sent during the communication

In crypto jargon: **Unforgeability under chosen message attack**

Towards a Security Definition

$\mathscr{A}$

Adversary's Goal

To *produce* a tag that certifies the **authenticity** of a *known* message that is different from what has been sent during the communication
In crypto jargon: **Unforgeability under chosen message attack**

Adversary's Power

passive adversary

Efficient algorithm (probabilistic, and runs in polynomial time $< 2^{60}$)

$\mathscr{A}$ can see everything transmitted over the communication channel

$\mathscr{A}$ knows all details of the MAC scheme except for the secret key (*Kerckhoffs' principle)*

active adversary

$\mathscr{A}$ can **drop**, **replace** and **inject** information into the communication channel

Towards a Security Definition

$\mathcal{A}$

Adversary's Goal

To *produce* a tag that certifies the **authenticity** of a *known* message that is different from what has been sent during the communication
In crypto jargon: **Unforgeability under chosen message attack**

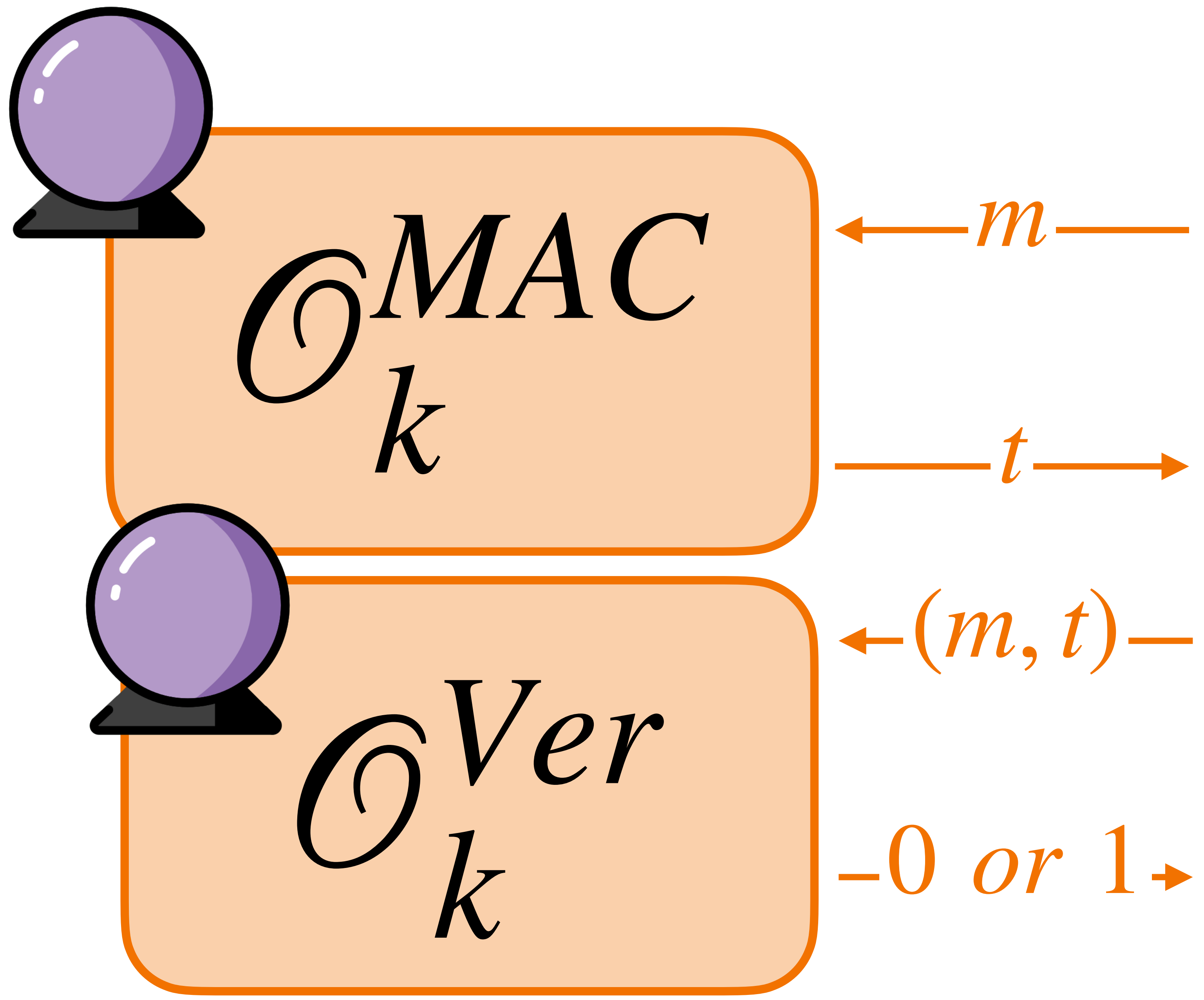
Adversary's Power

$\mathcal{A}$ can **drop**, **replace** and **inject** information into the communication channel (***active*** *adversary)*

Adversary's Resources

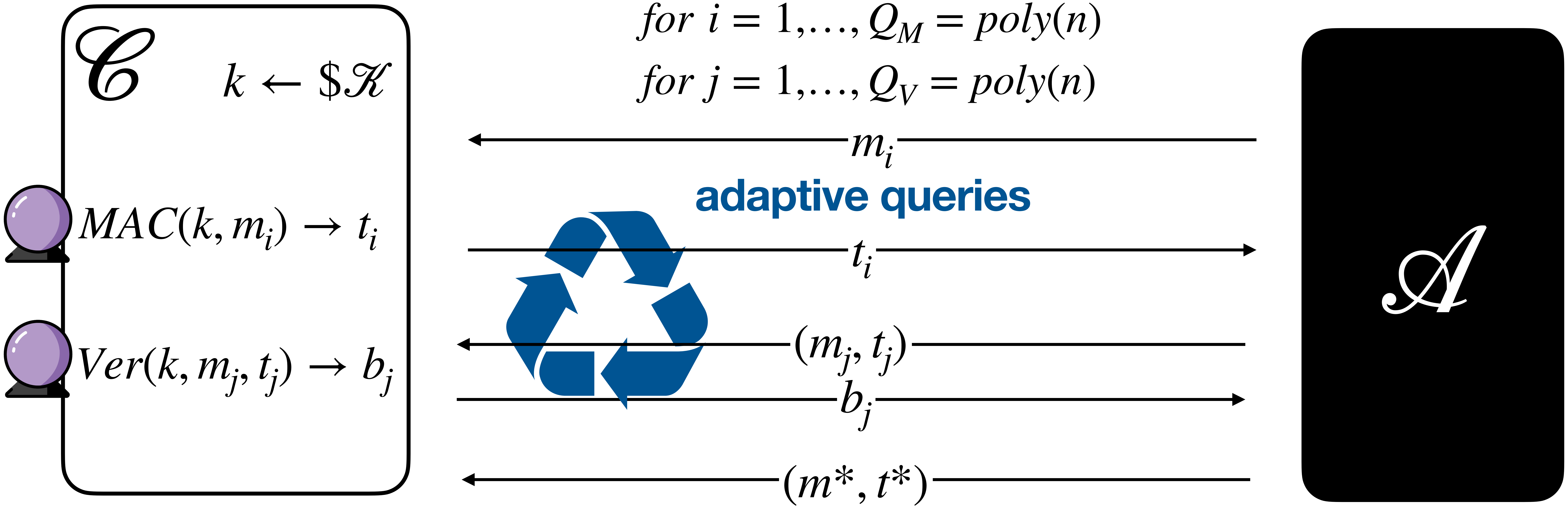
Access to the communication channel

Access to oracles



Security for MACs

Aim: quantify the $\mathscr{A}$'s likelihood in forging a valid tag t^* for a ***new*** (different) message m^*



$\mathscr{A}$ wins the security game iff:

$Ver(k, m^*, t^*) = 1$ **AND** $m^* \notin \{m_1, \ldots, m_{Q_M}\}$

This security game is called: **Unforgeability under Chosen Message Attack**

Secure MAC

A Message Authentication Code is said to be **secure** (unforgeable under chosen message attack) if **for all efficient** adversaries the probability that $\mathscr{A}$ **wins** the security game is **negligible**. Formally,

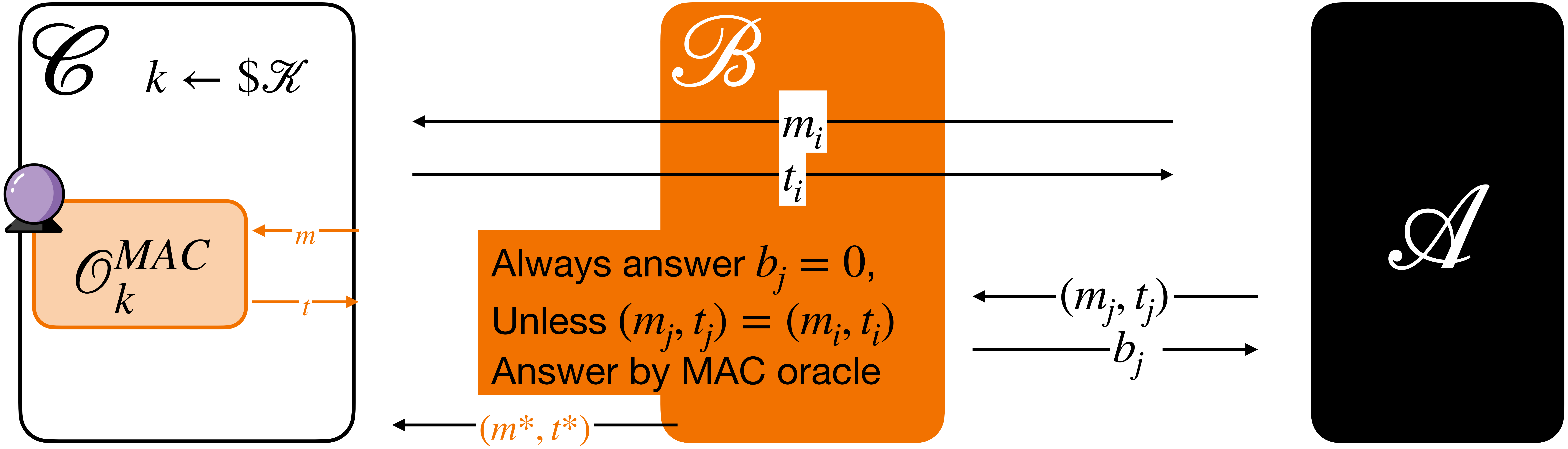
$$Pr[Ver(k, m^*, t^*) = 1 \,|\, (m^*, t^*) \leftarrow \mathscr{A}^{\mathcal{O}_k^{MAC}, \mathcal{O}_k^{Ver}} \wedge m^* \notin \{m_i\}_{i=1}^{Q_M}] \leq negl(n)$$

In this case n is the size of the key space $\mathscr{K} = \{0,1\}^n$

Verification Queries Do Not Help!

For every $\mathscr{A}$ that plays the unforgeability game *with* verification oracle, we can construct a new adversary $\mathscr{B}$ that plays the unforgeability game ***without*** verification oracle and **Prob[$\mathscr{B}$ wins] = Prob[$\mathscr{A}$ wins]/Q_V.**

Wlog, we can assume that $\mathscr{A}$ submits its final forgery as a query to the Verification oracle during the game.



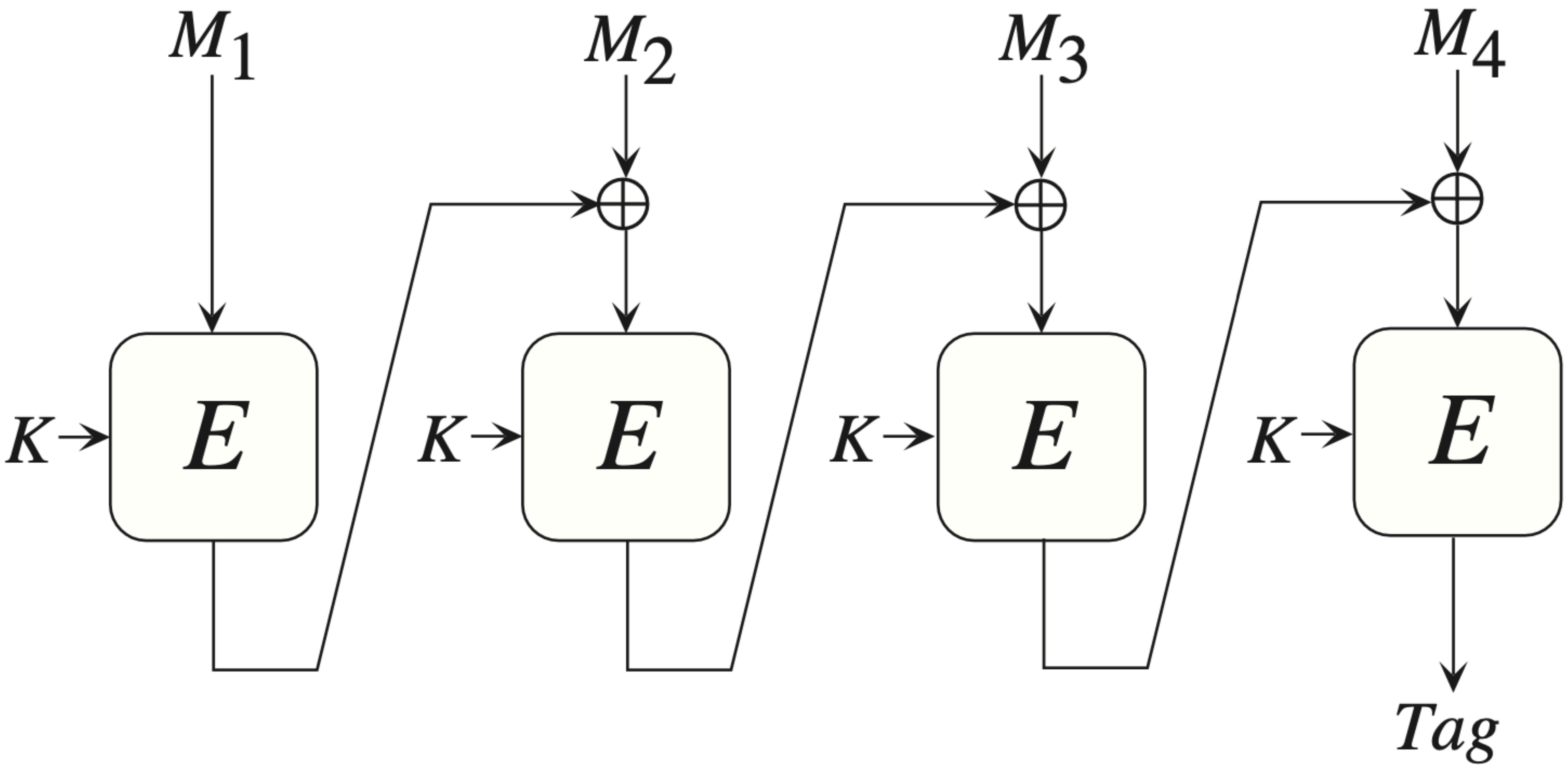
$$(m^*, t^*) \leftarrow \$\{(m_j, t_j)\}_{j=1}^{Q_V} \backslash \{(m_i, t_i)\}_{j=1}^{Q_S}$$

By always returning $b_j = 0$, $\mathscr{B}$ might be giving the wrong answer to $\mathscr{A}$ *some time* (precisely when $\mathscr{A}$ produces a forgery). But $\mathscr{B}$ wouldn't know, so it will pick one of $\mathscr{A}$'s queries as its forgery. This guess will be correct with probability $1/Q_V$.

For more details, read Theorem 6.1 in BonehShoup

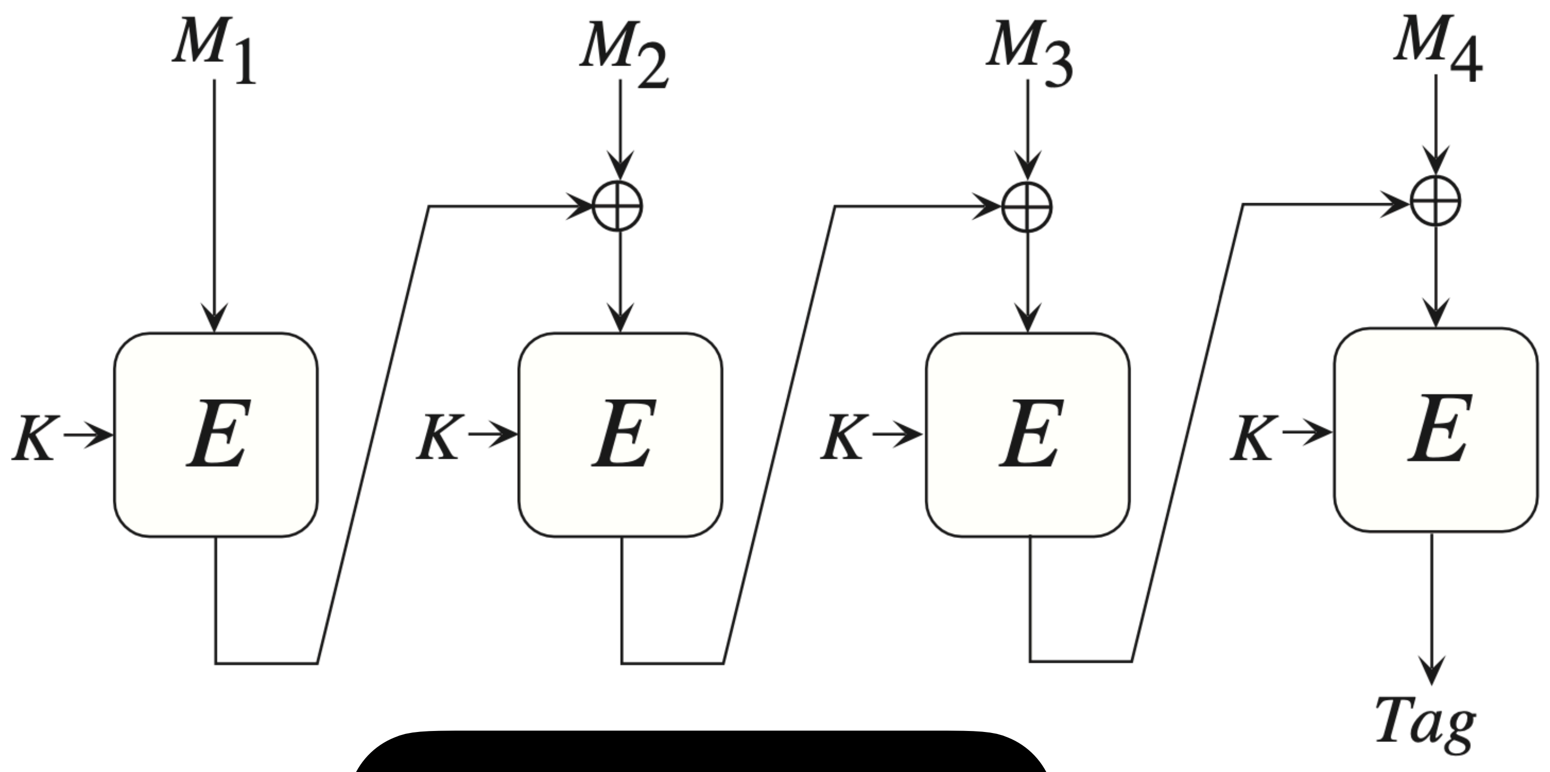
CBC-MAC

🧐 *This RAW version of CBC-MAC is NOT unforgeable. Can you see why?*



The random IV in CBC encryption mode serves to prevent a dictionary attack on the first ciphertext block. Confidentiality is not a concern for MACs, so IV=0 is good enough. The 'Tag' is only one block long (so usually shorter than a message, that can be multiple blocks long… + padding)

Raw CBC-MAC Message Extension Attack



$\mathscr{C}$

$\mathscr{A}$

$$m_1 = (M_0 || M_1)$$

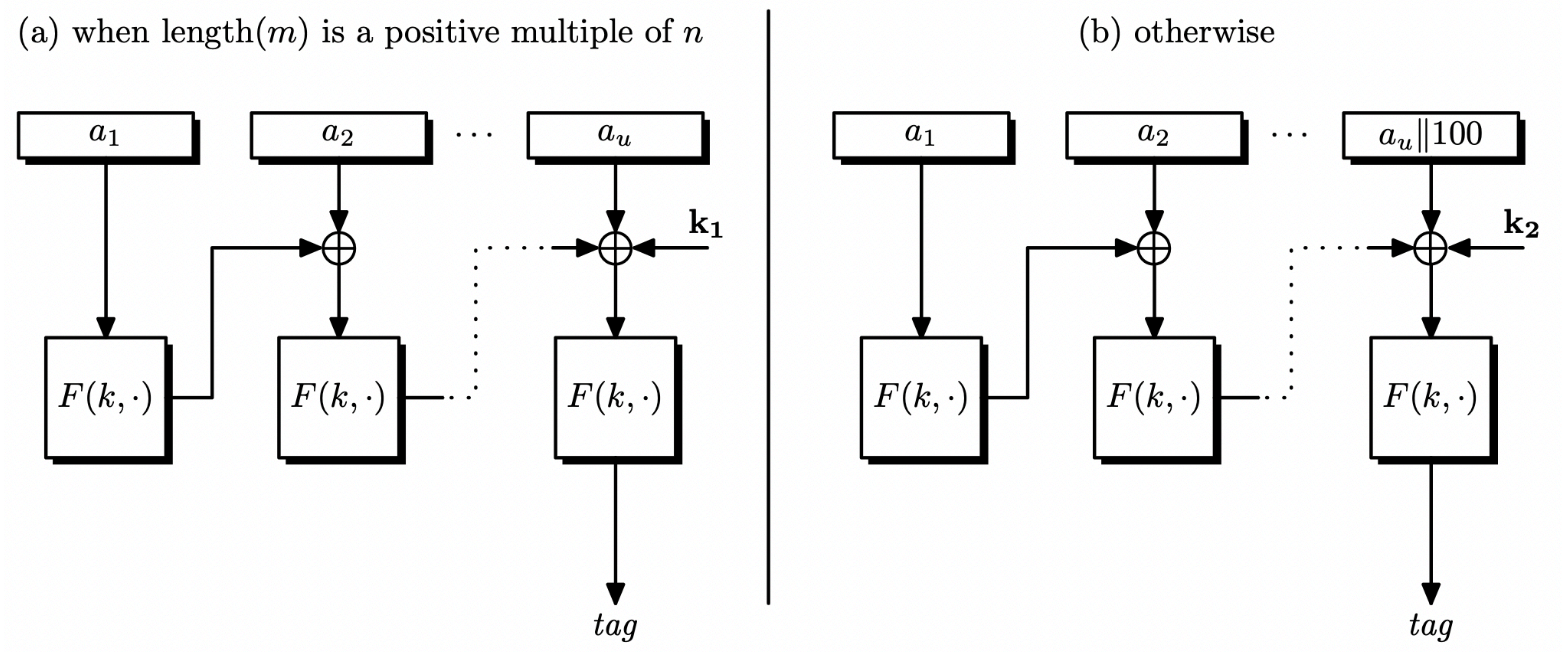
$$t_1 = E(k, E(k, M_0) \oplus M_1)$$

$$m_2 = (M_0' || M_1')$$

$$t_2 = E(k, E(k, M_0') \oplus M_1')$$

$$(m^*, t^*) \quad = ((M_0 || M_1 || M_0' \oplus t_1 || M_1'), t_2)$$

ANSI CBC-MAC



(a) when length(m) is a positive multiple of n
a_1
a_2
a_u
$\mathbf{k_1}$
$F(k,\cdot)$
$F(k,\cdot)$
$F(k,\cdot)$
tag
(b) otherwise
a_1
a_2
$a_u \| 100$
$\mathbf{k_2}$
$F(k,\cdot)$
$F(k,\cdot)$
$F(k,\cdot)$
tag

MACing Using Block Ciphers VS Hash Functions

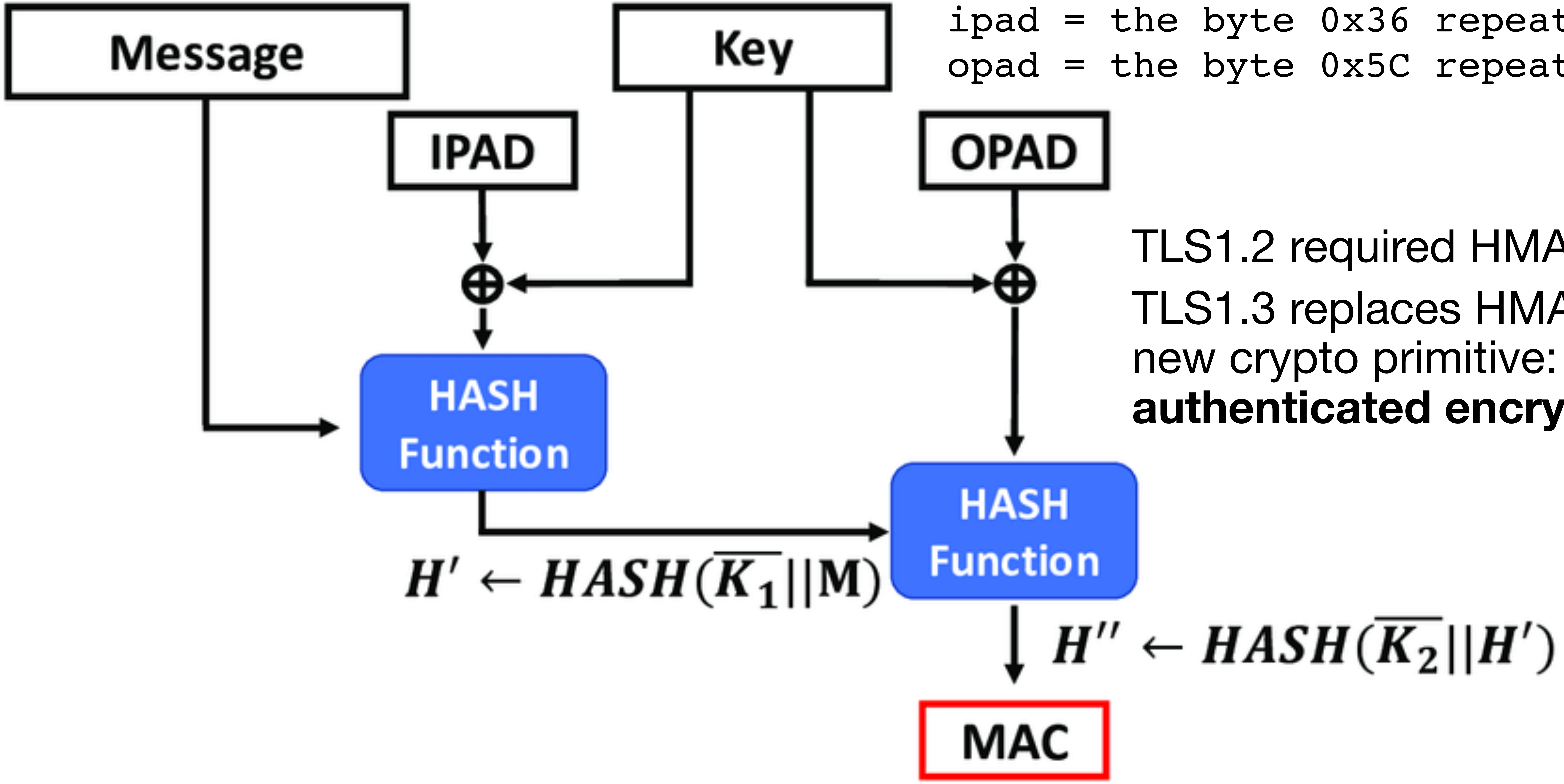
- Cryptographic hash functions are usually faster to compute than block ciphers, in software implementations
- The code that implements many hash functions is free, ready to use and can “cross borders” [USA used to restrict the export of cryptographic technologies and devices until 1992!]

BUT

- Hash functions are not designed for message authentication, and usually do not have keys! How to go about this?
- **HMAC mandatory MAC for internet security protocols (TLS, SSH)**

HMAC

$$HMAC(k, text) = H(k \oplus \texttt{opad} \,||\, H(k \oplus \texttt{ipad} \,||\, text))$$




```
ipad = the byte 0x36 repeated 64 times
opad = the byte 0x5C repeated 64 times.
```

TLS1.2 required HMAC-SHA1-96

TLS1.3 replaces HMACs with a new crypto primitive: **authenticated encryption** (AEAD)

Module 1: Agenda

Commitment Schemes

Hash Functions

Blockchain Technology

OTP & Perfect Secrecy

Randomness in Cryptography

Semantic Security + Proof

Block Ciphers

Modes of Operation

Message Authentication Codes (MAC)

- What's the Problem?
- Definition (Syntax)
- Adversary's Goals & Powers
- Security Notion
- A Construction: HMAC

Authenticated Encryption

- GCM

Authenticated Encryption (With Associated Data)

CONFIDENTIALITY

&

INTEGRITY

at the same time

Authenticated Encryption via Generic Composition

- c_1 *may leak information about* m
- *decryption happens before the integrity check*

This is the most secure way to compose the two primitives
It is used in TLS1.2, IPsec, GCM

Encrypt-and-MAC:

AE.Encrypt

Split $k = (k_0 || k_1)$

$c_0 \leftarrow E(k_0, m)$

$c_1 \leftarrow MAC(k_1, m)$

return $c = (c_0, c_1)$

AE.Decrypt

Split $k = (k_0 || k_1)$

$m \leftarrow D(k_0, c_0)$

$b \leftarrow Ver(k_1, m, c_1)$

if $b = 1$ return m

Else return $\perp$

Encrypt-then-MAC:

AE.Encrypt

Split $k = (k_0 || k_1)$

$c_0 \leftarrow E(k_0, m)$

$c_1 \leftarrow MAC(k_1, c_0)$

return $c = (c_0, c_1)$

AE.Decrypt

Split $k = (k_0 || k_1)$

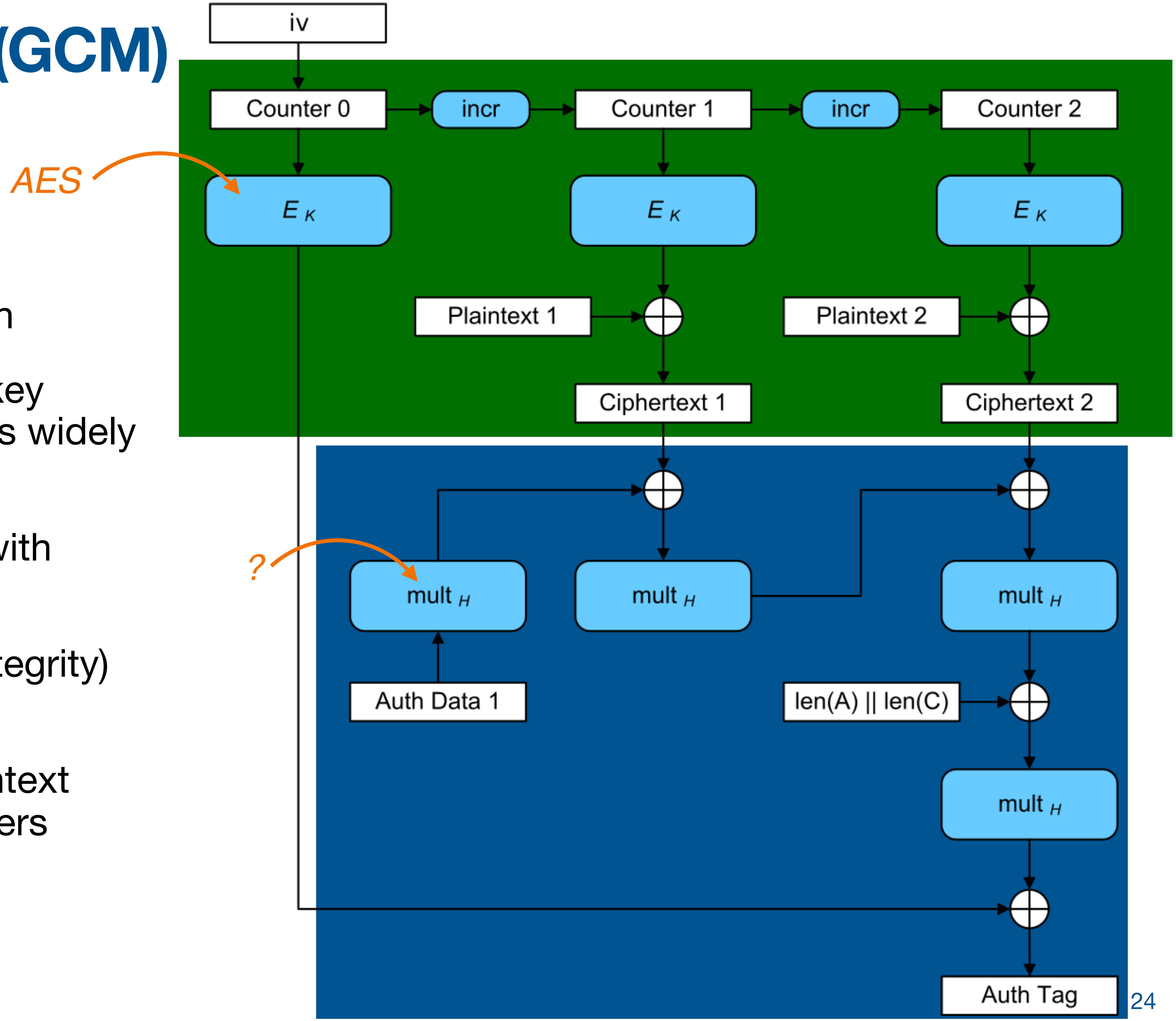
$b \leftarrow Ver(k_1, c_0, c1)$

if $b = 1$ return $D(k_0, c_0)$

Else return $\perp$

There are many ways to combine a cipher and a MAC, not all combinations are secure!

Galois Counter Mode (GCM)



- Encrypt-then-MAC AE construction
- Mode of operation for symmetric-key cryptographic block ciphers which is widely adopted for its performance
- State-of-the-art throughput rates with inexpensive hardware resources
- Provides both data authenticity (integrity) and confidentiality
- Additionally may authenticate plaintext *Associated Data (**AEAD**)*, e.g., headers

mult_H : GHASH Keyed Hash Function Over a Galois Field

$$GHASH : \mathscr{K} \times \mathscr{P} \times \mathscr{X} \rightarrow \mathscr{X}$$

GHASH(H,A,C) = X

$$\underbrace{A_1 \; \ldots \; A_m || 0^{128-v} \; C_1 \; \ldots \; C_n || 0^{128-u} \; L}$$

$$S \in \{0,1\}^{128 \times (m+n+1)}$$

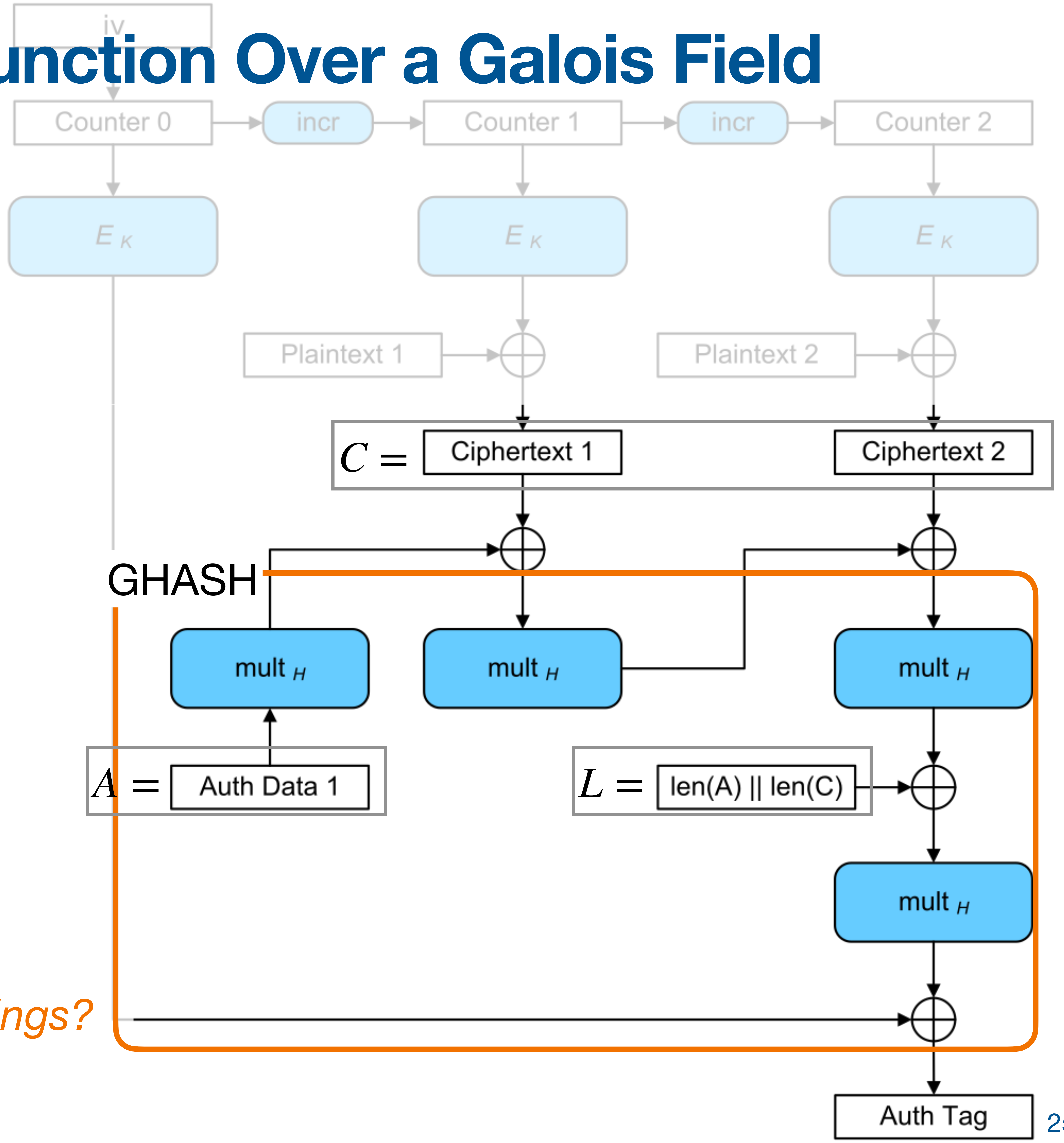
GHASH(H,S_i) = X_i

$$X_i = (X_{i-1} \oplus S_i) \cdot H \quad (for \; i > 0)$$

$$X_0 = 0^{128}$$

$$H = E_k(0^{128})$$

🧐 *How to **multiply** two bit-strings?*



Galois Field Multiplication

Intuition:

1- see bit-strings as vectors with coefficients over $\mathbb{Z}_2$

2- see vectors as polynomials

3- we know how to multiply polynomials

Math caveat

In order to make sure the result of the multiplication is *always* a bit-string of length 128 we need to do operations in a special mathematical object called **Galois Field**

$$GF(2^{128}) := \mathbb{Z}_2[x]/(x^{128} + x^7 + x^2 + x + 1)$$

Overview of Module 1

Hash Functions

OTP

AE

IND-CPA

Perfect-Secrecy

Commitment Schemes

MAC

Block-Cipher

Semantic Security

Now you can understand ~70% of the cryptographic tools used nowadays

---

What's left?

- Public key encryption
- Key exchange protocols
- Digital signatures and Certificates
- Proof Systems (NIZK, SNARK)
- MPC (secure multi party computation)
- Privacy Enhancing Technologies