Cryptography Engineering

- Lecture 3 (Nov 5, 2026)

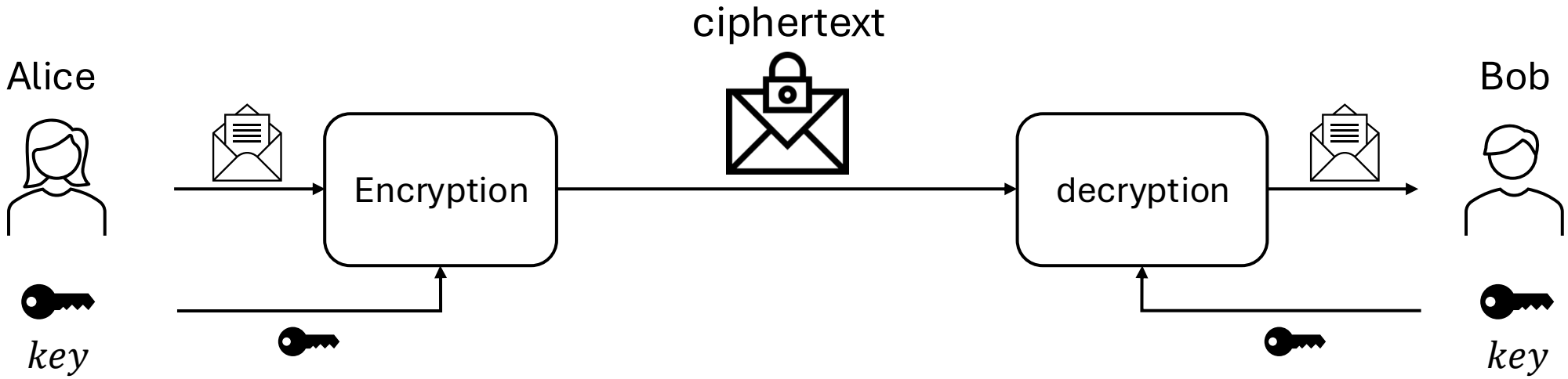
- Today:
  - SKE, AE, AEAD
  - Padding
  - Key strength
  - Summary of symmetric-key primitives
  - Introduction to public-key primitives

Recap

- Hash functions
- HMAC, HKDF

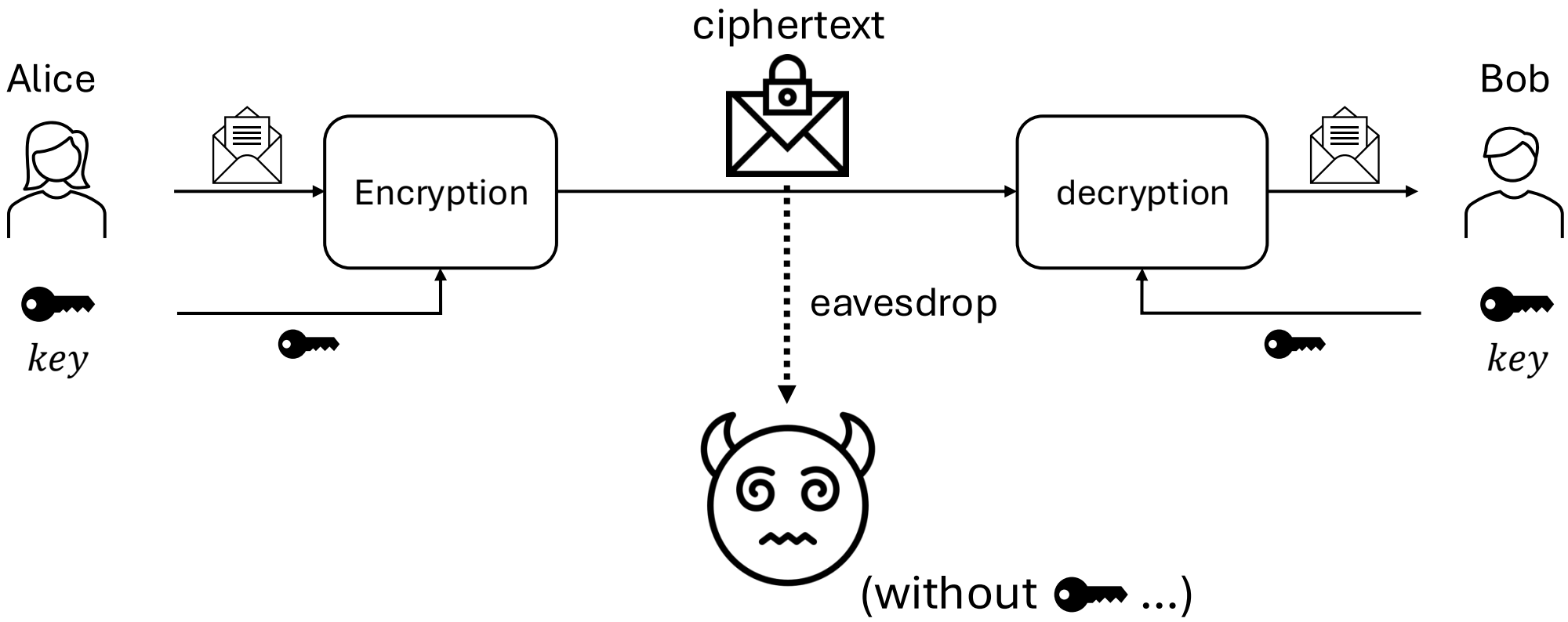
Symmetric Primitives - SKE

- Symmetric-key Encryption



Symmetric Primitives - SKE

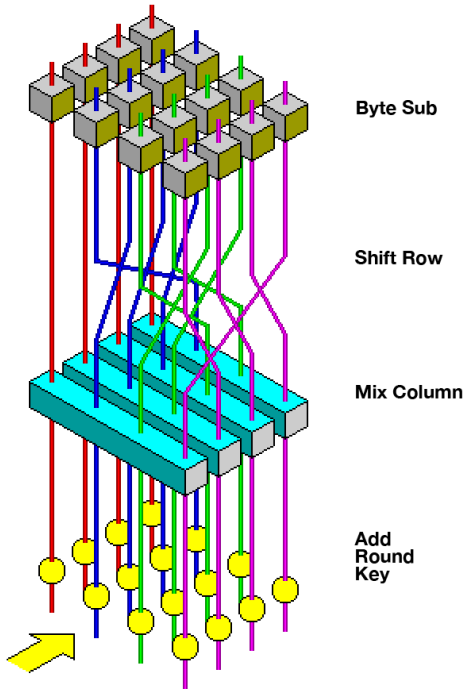
- Symmetric-key Encryption (**Confidentiality**)



Symmetric Primitives - SKE

- Symmetric-key Encryption
  - AES (Advanced Encryption Standard)
  - Fixed-length encryption (block cipher)

- Extend to arbitrary-length encryption **via Mode of Operation**
  - CBC, CTR, …



(Image from Wikipedia)

AES Block Cipher

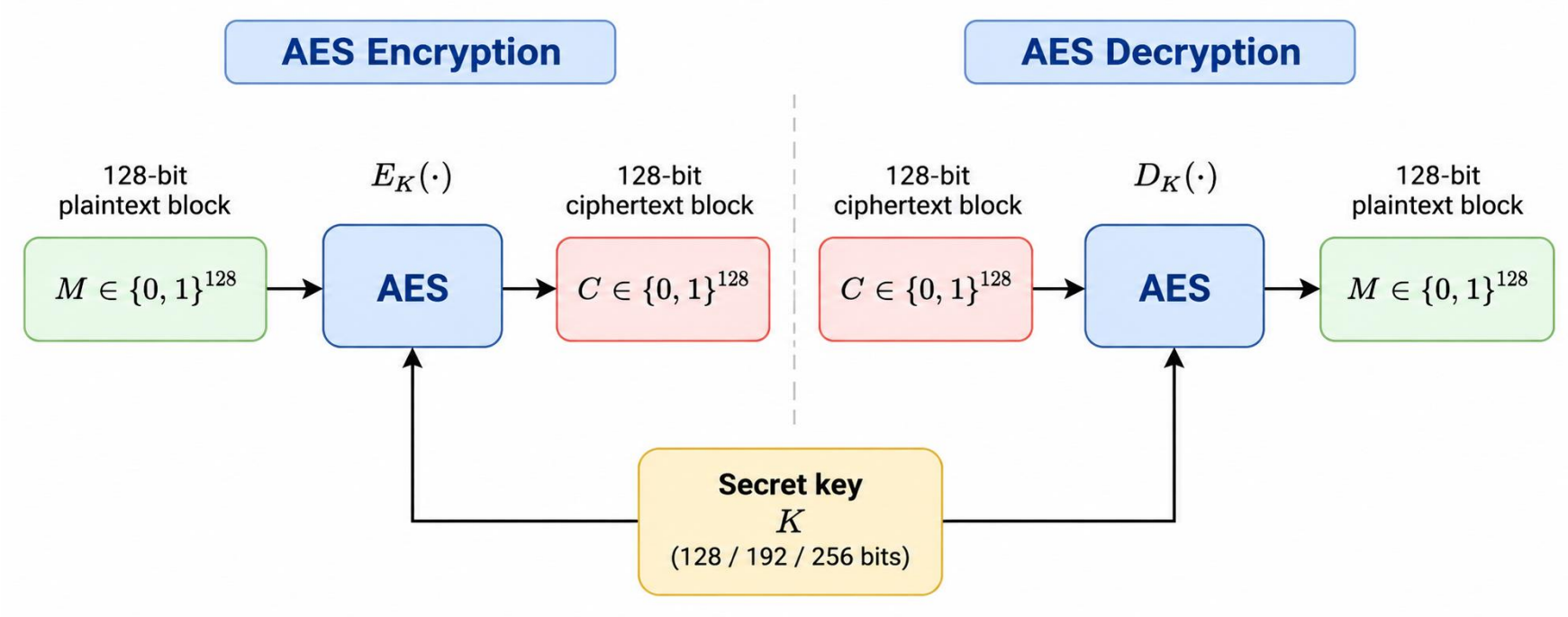


Image from
ChatGPT-6 Sol

Modes of Operations

- We use **Mode of Operation** to handle arbitrary-length messages

Case Study - CBC

- We use **Mode of Operation** to handle arbitrary-length messages
  - Example: AES-CBC (Cipher Block Chaining Mode)

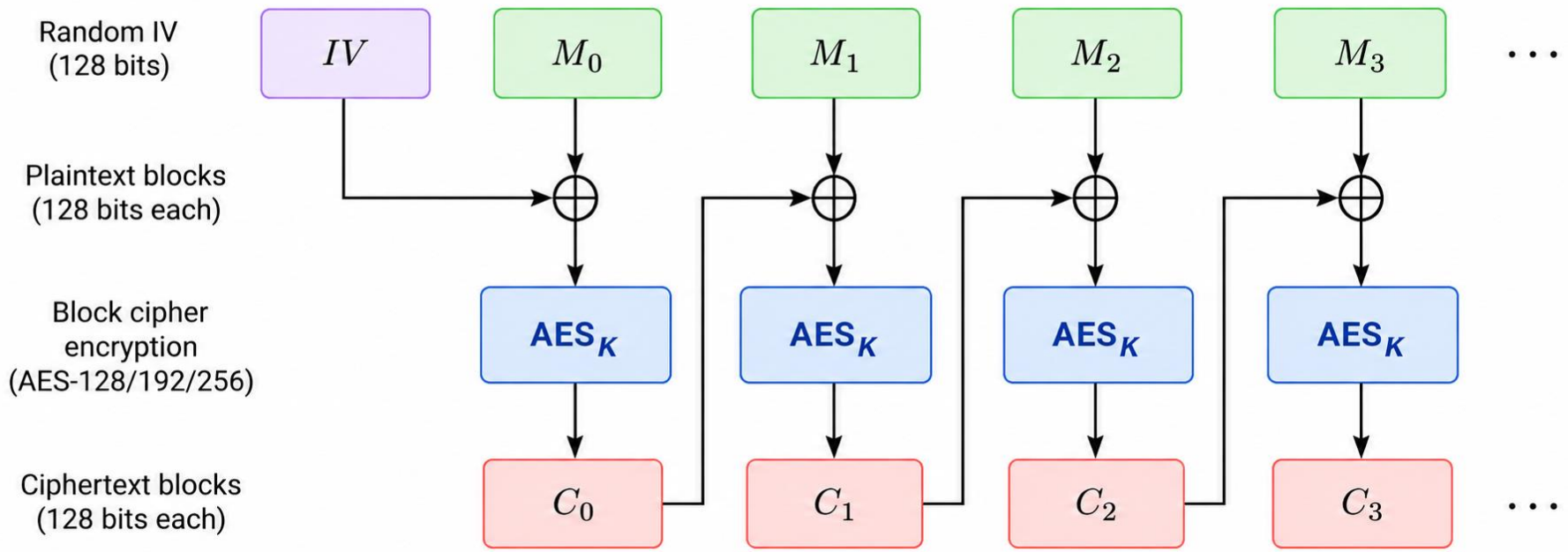


Image from ChatGPT-6 Sol

Case Study - CBC

- We use **Mode of Operation** to handle arbitrary-length messages
  - Example: AES-CBC (Cipher Block Chaining Mode)

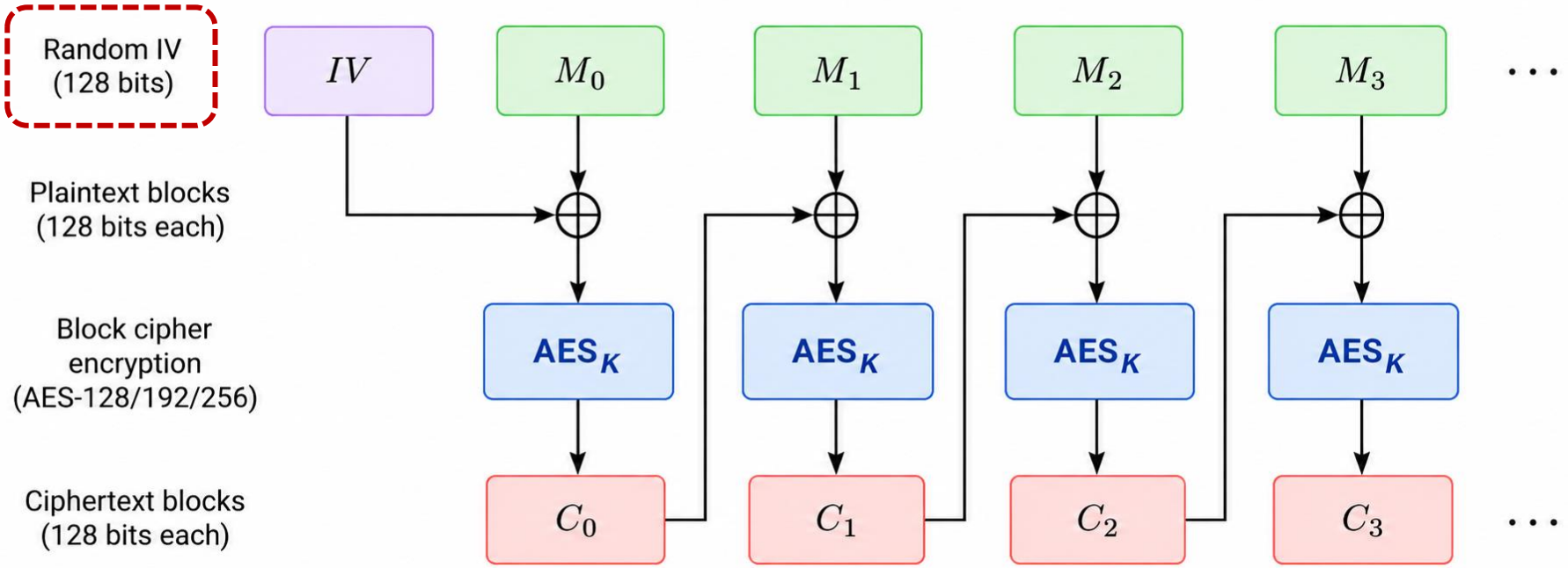


Image from ChatGPT-6 Sol

Case Study - CBC

- AES-CBC (Cipher Block Chaining Mode)
  - Let K be the AES key
  - Let $M = M_0 \parallel M_1 \parallel \cdots \parallel M_{n-1}$ (each plaintext block is 128-bit)

Case Study - CBC

- AES-CBC (Cipher Block Chaining Mode)
  - Let K be the AES key
  - Let $M = M_0 \parallel M_1 \parallel \cdots \parallel M_{n-1}$ (each plaintext block is 128-bit)
  - Choose a **random 128-bit initial vector** IV

Case Study - CBC

- AES-CBC (Cipher Block Chaining Mode)
  - Let K be the AES key
  - Let $M = M_0 \parallel M_1 \parallel \cdots \parallel M_{n-1}$ (each plaintext block is 128-bit)
  - Choose a random 128-bit initial vector IV
  - $C_0 = AES_K(M_0 \oplus IV)$
  - $C_i = AES_K(M_i \oplus C_{i-1}), i = 1, \ldots, n-1$

Case Study - CBC

- AES-CBC (Cipher Block Chaining Mode)
  - Let K be the AES key
  - Let $M = M_0 \parallel M_1 \parallel \cdots \parallel M_{n-1}$ (each plaintext block is 128-bit)
  - Choose a random 128-bit initial vector IV
  - $C_0 = AES_K(M_0 \oplus IV)$
  - $C_i = AES_K(M_i \oplus C_{i-1}), i = 1, \ldots, n-1$

  - **Final ciphertext:**

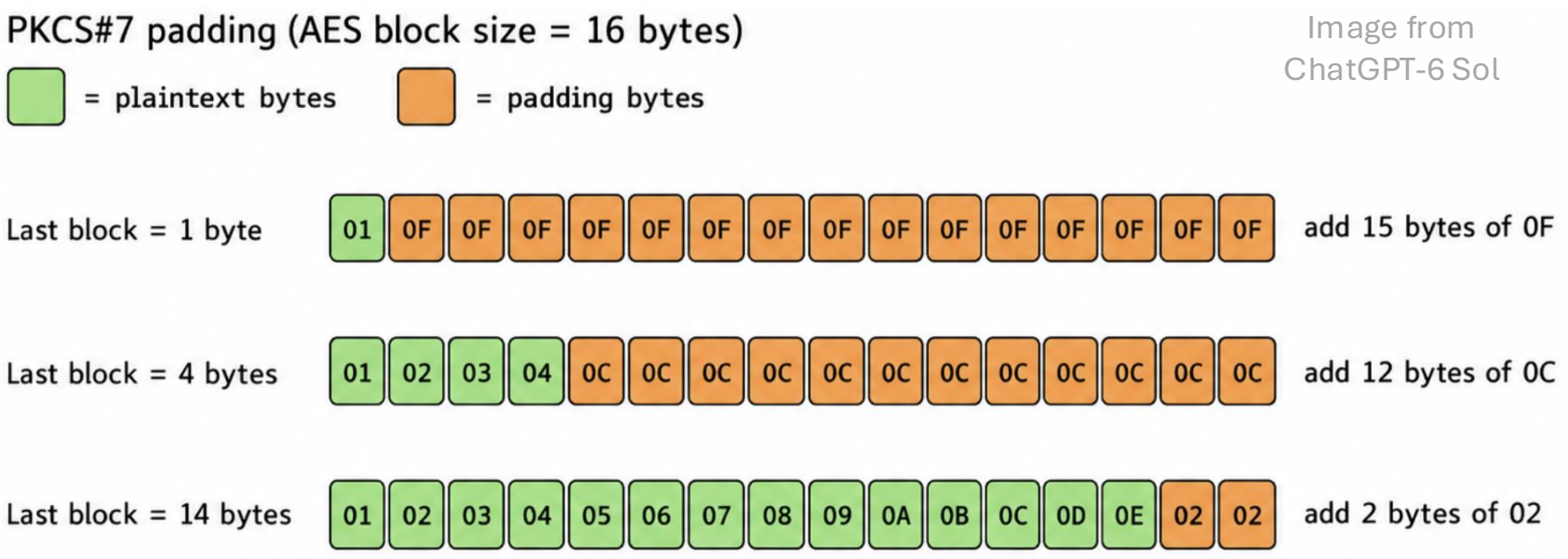
$$IV \parallel C_0 \parallel C_1 \parallel \cdots \parallel C_{n-1}$$

Case Study - CBC

- In CBC mode, what if $|M|$ (the bit-length of M) is not a multiple of 128 bits (16 bytes)?

Case Study - CBC

- In CBC mode, what if $|M|$ is not a multiple of 128 bits (16 bytes)?
- Then we need **Padding**



Case Study - CBC

- In CBC mode, what if $|M|$ is not a multiple of 128 bits (16 bytes)?

- AES-CBC padding via PKCS#7:
  - Let L be the **byte-length** of the last block of M
  - If $L < 16$ bytes, then append $16 - L$ bytes, each with the value $16 - L$

Case Study - CBC

- In CBC mode, what if $|M|$ is not a multiple of 128 bits (16 bytes)?

- AES-CBC padding via PKCS#7:
  - Let L be the byte-length of the last block of M
  - If $L < 16$ bytes, then append $16 - L$ bytes, each with the value $16 - L$
  - If $L = 16$ bytes, then append an **additional** 16-byte block, with every byte set to 0x10
    - This is to **avoid ambiguity**

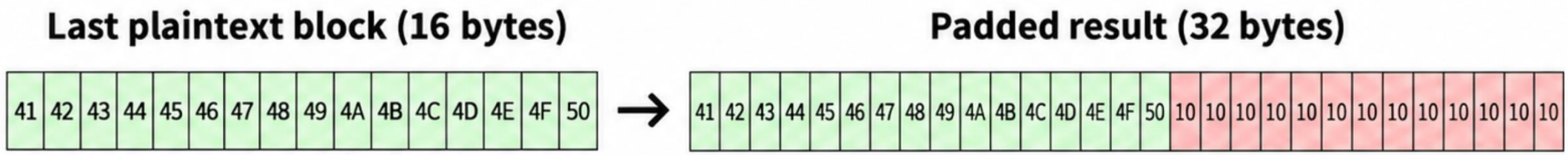


Image from ChatGPT-6 Sol

Case Study - CBC

- In CBC mode, the IV value **must be random and unpredictable**
  - A fresh, random IV should be generated for each encryption.

- Reusing or predicting IVs can enable chosen-plaintext attacks (CPA).
  - Reusing the same IV makes CBC encryption deterministic:

$$AES_K(M \oplus IV) = AES_K(M' \oplus IV)$$
$$\Updownarrow$$
$$M = M'$$

Case Study - CTR

- AES-CTR (counter mode)

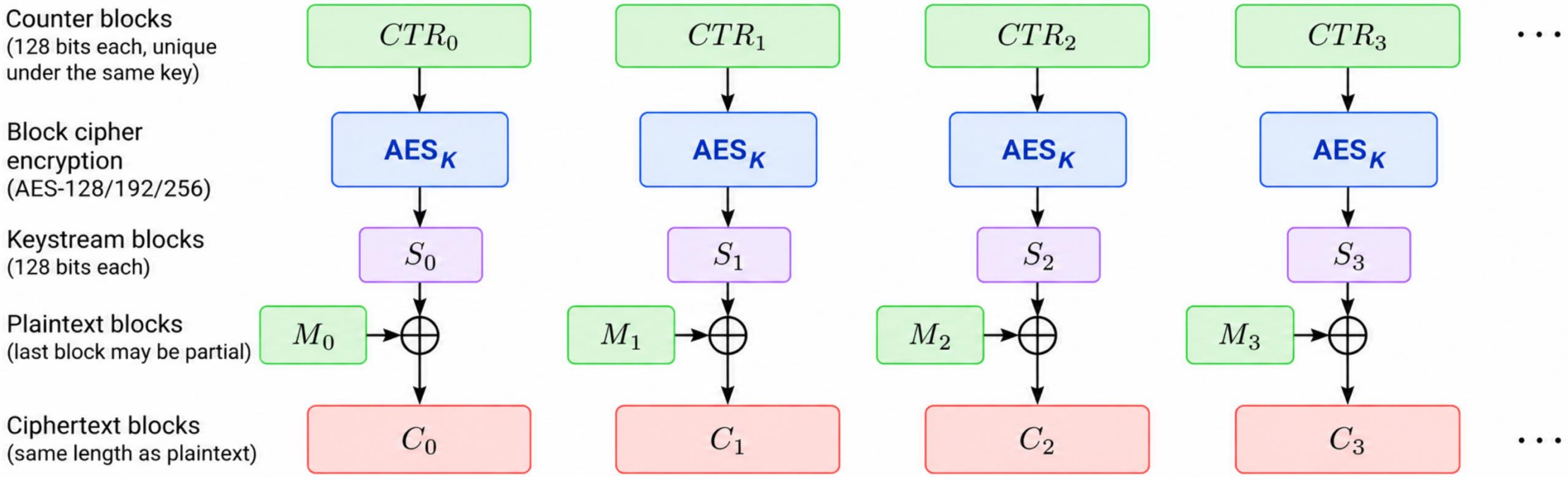


Image from
ChatGPT-6 Sol

Case Study - CTR

- AES-CTR (counter mode)
  - Let K be the AES key
  - Let $M = M_0 \parallel M_1 \parallel \cdots \parallel M_{n-1}$ (each plaintext block is 128-bit)
  - Let $\mathrm{CTR}_0, \mathrm{CTR}_1, \mathrm{CTR}_2, \ldots, \mathrm{CTR}_{n-1}$ be 128-bit counter blocks
  - $C_i = M_i \oplus AES_K(\mathrm{CTR_i}), i = 1, \ldots, n-1$

Image from ChatGPT-6 Sol

Case Study - CTR

- AES-CTR (counter mode)
  - Let K be the AES key
  - Let $M = M_0 \parallel M_1 \parallel \cdots \parallel M_{n-1}$ (each plaintext block is 128-bit)
  - Let $\mathrm{CTR}_0, \mathrm{CTR}_1, \mathrm{CTR}_2, \ldots, \mathrm{CTR}_{n-1}$ be 128-bit counter blocks
  - $C_i = M_i \oplus AES_K(\mathrm{CTR_i}), i = 1, \ldots, n-1$
  - **In practice, the counters are**

$$\mathrm{CTR}_i = \mathrm{CTR}_{i-1} + 1 \ (mod\ 2^{128}),$$

  - **so the ciphertext is**

$$C = (\mathrm{CTR}_0, C_0, C_1, \ldots, C_{n-1})$$

Image from ChatGPT-6 Sol

Case Study - CTR

- AES-CTR (counter mode)
  - Let K be the AES key
  - Let $M = M_0 \parallel M_1 \parallel \cdots \parallel M_{n-1}$ (each plaintext block is 128-bit)
  - Let $\mathrm{CTR}_0, \mathrm{CTR}_1, \mathrm{CTR}_2, \ldots, \mathrm{CTR}_{n-1}$ be 128-bit counter blocks
  - $C_i = M_i \oplus AES_K(\mathrm{CTR_i}), i = 1, \ldots, n-1$
  - **In practice, the counters are**

$$\mathrm{CTR}_i = \mathrm{CTR}_{i-1} + 1 \ (mod\ 2^{128}),$$

  - **so the ciphertext is**

$$C = (\mathrm{CTR}_0, C_0, C_1, \ldots, C_{n-1})$$

How does AES-CTR decryption work?

Image from ChatGPT-6 Sol

Case Study - CTR

- AES-CTR (counter mode)

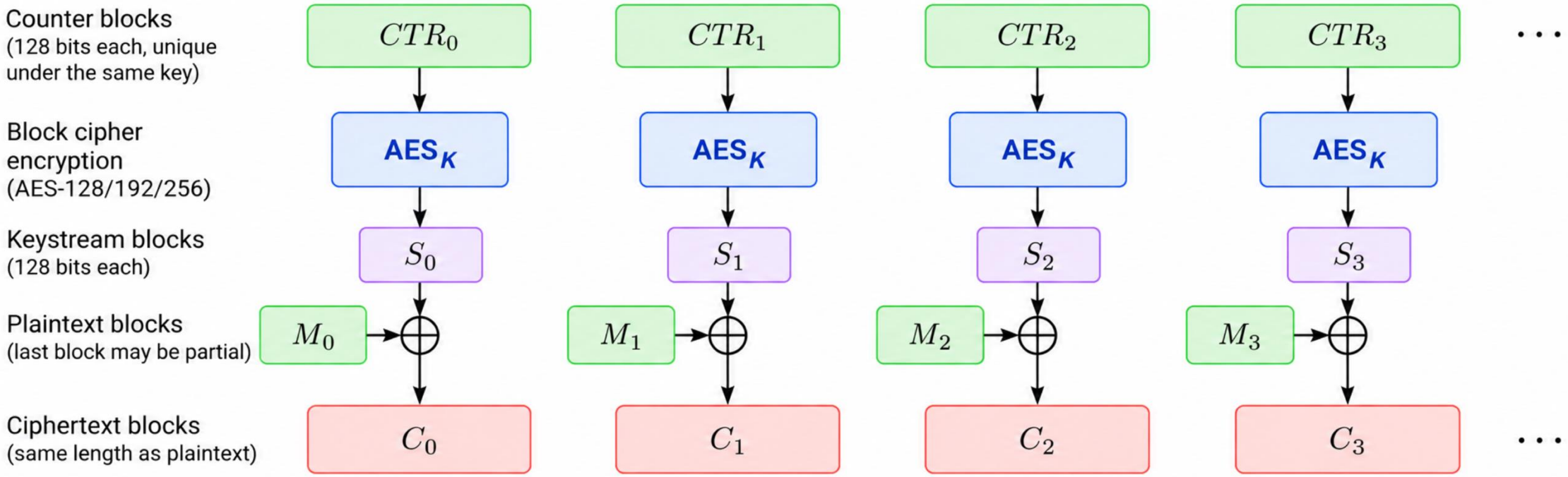


Image from
ChatGPT-6 Sol

Case Study - CTR

- In CTR mode, **a counter block must never be reused under the same key**
  - Repeated counter blocks can cause CPA attacks.

$$\left.\begin{aligned} C_1 &= M_1 \oplus AES_K(\mathrm{CTR}) \\ C_2 &= M_2 \oplus AES_K(\mathrm{CTR}) \end{aligned}\right\} \Rightarrow C_1 \oplus C_2 = M_1 \oplus M_2$$

  - Reusing the same counter block under different keys is fine.

Case Study - CTR

- Real-world implementation of CTR mode **usually uses IV (or nonces)** to avoid repeated blocks

Case Study - CTR

- Real-world implementation of CTR mode **usually uses IV (or nonces)** to avoid repeated blocks
- Example: AES-128-CTR with 32-bit counter (Please refer to the example code in Github repo…)

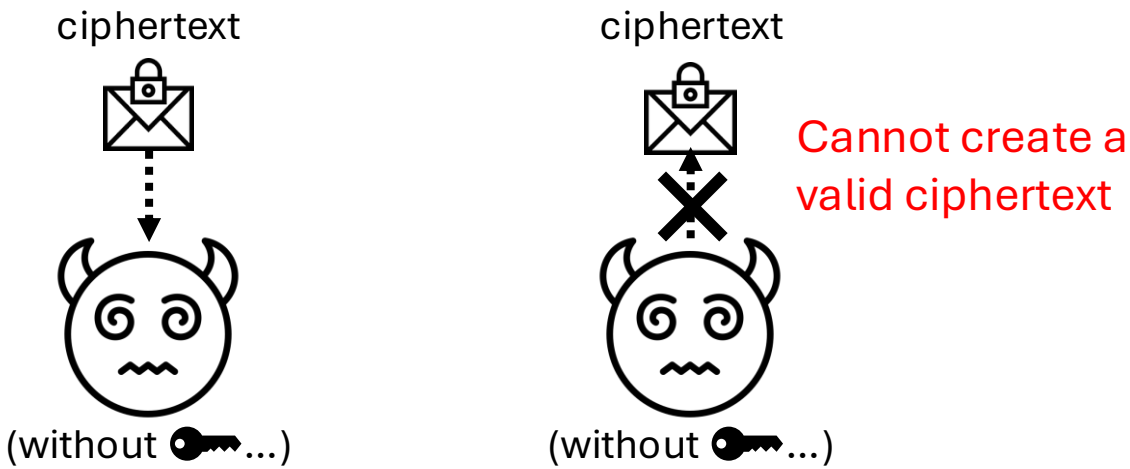
- type **AesCtr** = ctr::Ctr32BE<aes::Aes128>; // AES-128 with a 32-bit big endian counter

- let key = [0u8; 16]; // 128-bit key (not safe to use!)
- let initial_counter_block = [0u8; 16]; // 128-bit initial counter block: 96-bit nonce || 32-bit counter
- let mut message = b"Hello, world!".to_vec(); // message to encrypt
- let mut ciphertext = b"Hello, world!".to_vec(); // store the ciphertext
- let mut cipher = **AesCtr**::new(&key.into(), &initial_counter_block.into());

- cipher.**apply_keystream**(&mut ciphertext); // Encrypted in place.

Authenticated Encryption

- Authenticated Encryption
  - Symmetric Encryption
  - Not only **Confidentiality**, but also **Integrity**



Authenticated Encryption

- How to provide confidentiality and integrity at the same time…

Authenticated Encryption

- How to provide confidentiality and integrity at the same time…

- Natural answers: (suppose we have K_e for encryption and K_m for MAC)
  - Encryption-**and**-MAC: $c = Enc(K_e, M), t = Mac(K_m, M), ct = (c, t)$
  - MAC-**then**-encryption: $t = Mac(K_m, M), ct = Enc(K_e, M \parallel t)$
  - Encryption-**then**-MAC: $c = Enc(K_e, M), t = Mac(K_m, c), ct = (c, t)$

Authenticated Encryption

- How to provide confidentiality and integrity at the same time…
- Natural answers: (suppose we have K_e for encryption and K_m for MAC)
  - ~~Encryption-**and**-MAC: $c = Enc(K_e, M), t = Mac(K_m, M), ct = (c, t)$~~
  - ~~MAC-**then**-encryption: $t = Mac(K_m, M), ct = Enc(K_e, M \| t)$~~
  - **Encryption-then-MAC: $\boldsymbol{c = Enc(K_e, M), t = Mac(K_m, c), ct = (c, t)}$**
- The first two constructions are **insecure** under some instantiations (e.g., HMAC)

AEAD

- Similar to AE: Provide confidentiality and integrity simultaneously

AEAD

- Similar to AE: Provide confidentiality and integrity simultaneously
- But adding AD:
  - Encrypted messages in real-world applications may contain information that **does not need to be confidential** but **must be protected against tampering.**

AEAD

- Similar to AE: Provide confidentiality and integrity simultaneously
- But adding AD:
  - Encrypted messages in real-world applications may contain information that **does not need to be confidential** but **must be protected against tampering.**



AEAD

- Similar to AE: Provide confidentiality and integrity simultaneously
- But adding AD:
  - Encrypted messages in real-world applications may contain information that **does not need to be confidential** but **must be protected against tampering.**



  - Tampering metadata can cause unexpected behavior (e.g., wrong protocol, …)

AEAD

- Similar to AE: Provide confidentiality and integrity simultaneously
- But adding AD:
  - Encrypted messages in real-world applications may contain information that **does not need to be confidential** but **must be protected against tampering.**



  - Tampering metadata can cause unexpected behavior (e.g., wrong protocol, …)

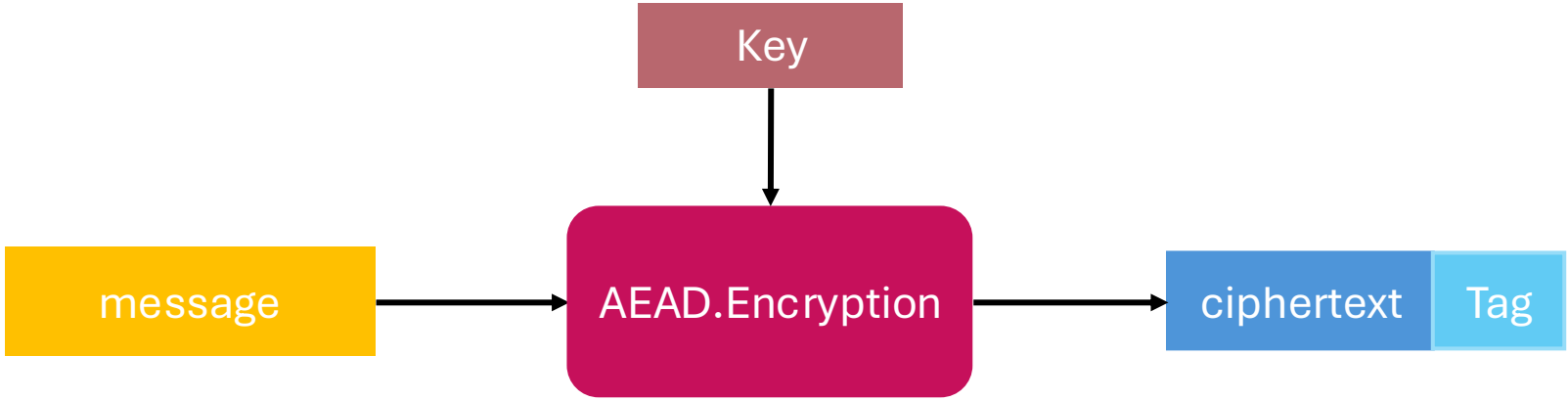
AEAD

- Similar to AE: Provide confidentiality and integrity simultaneously
- But adding AD:
  - Encrypted messages in real-world applications may contain information that **does not need to be confidential** but **must be protected against tampering.**

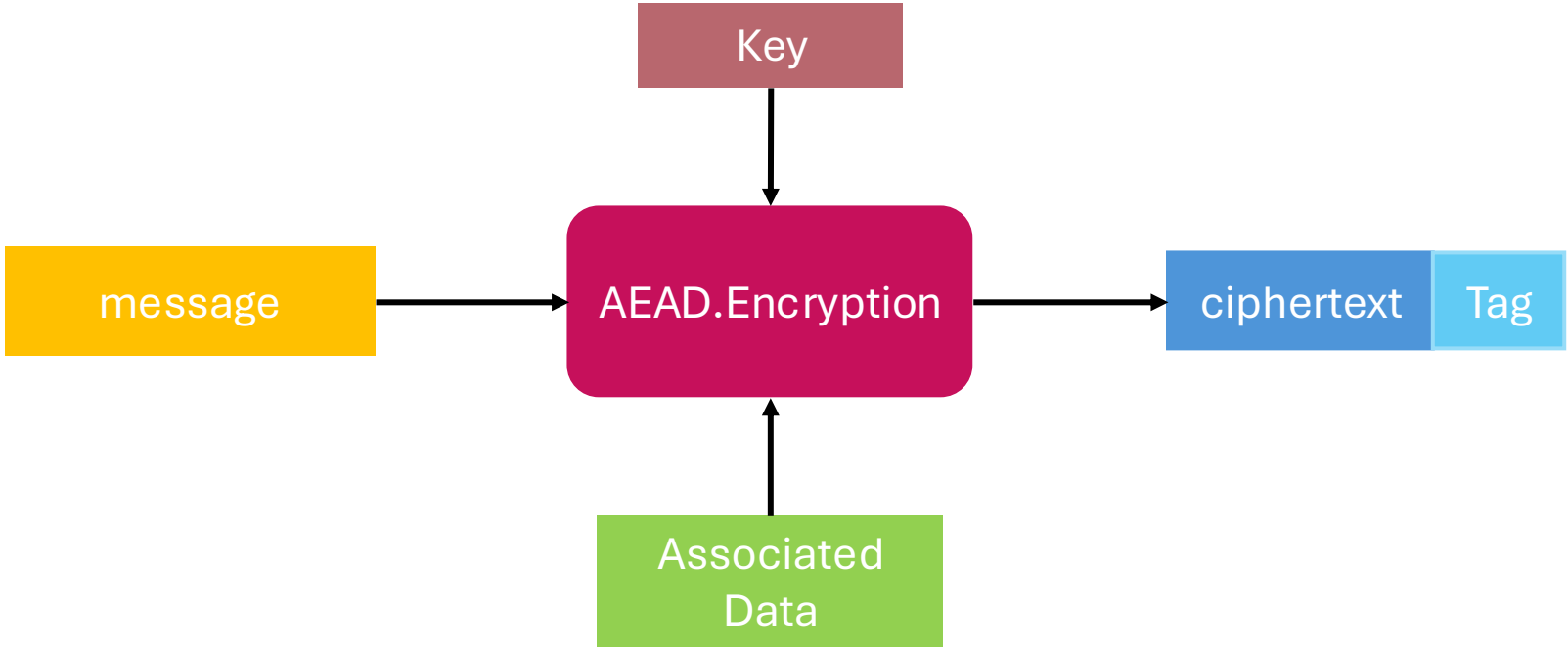


  - Tampering metadata can cause unexpected behavior (e.g., wrong protocol, …)

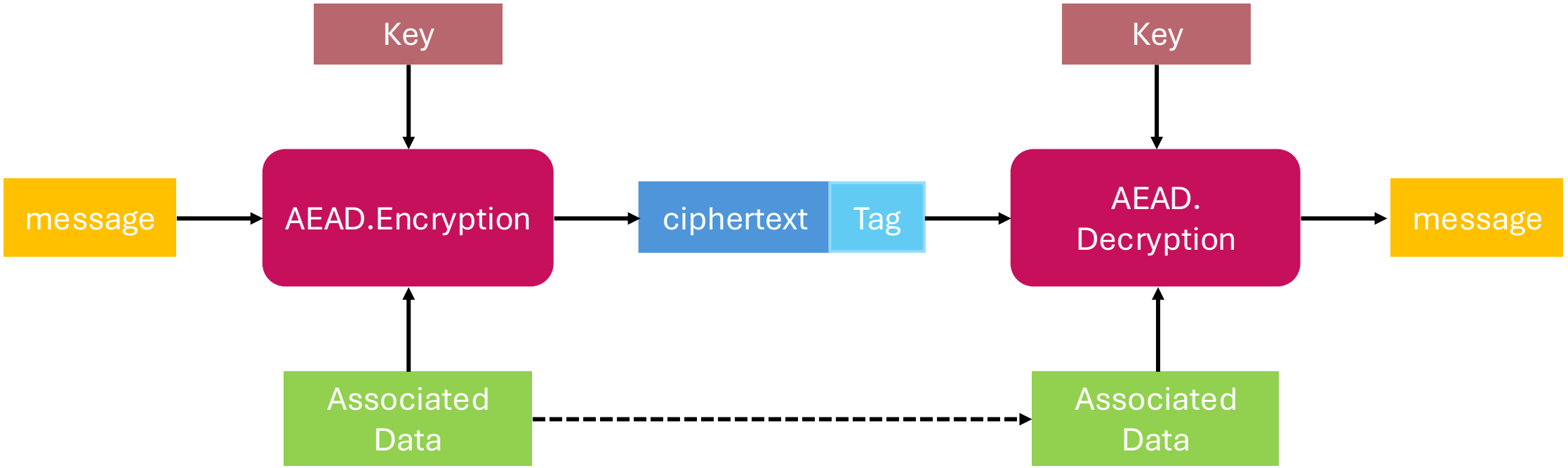
AEAD



AEAD


Key
message
AEAD.Encryption
ciphertext
Tag
Associated Data

AEAD


Key
Key
message
AEAD.Encryption
ciphertext
Tag
AEAD.
Decryption
message
Associated
Data
Associated
Data

AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + Galois-Hash-based authentication

AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication



AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication

Key K
(e,g, 256-bit)

Message M

Associated Data

AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication

Key K
(e,g, 256-bit)

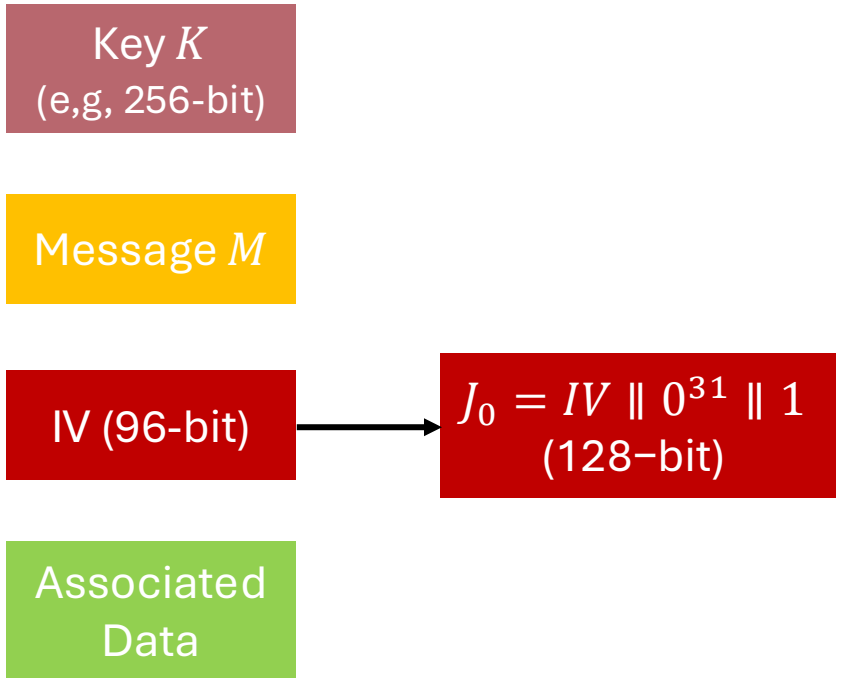
Message M

IV (96-bit)

Associated Data

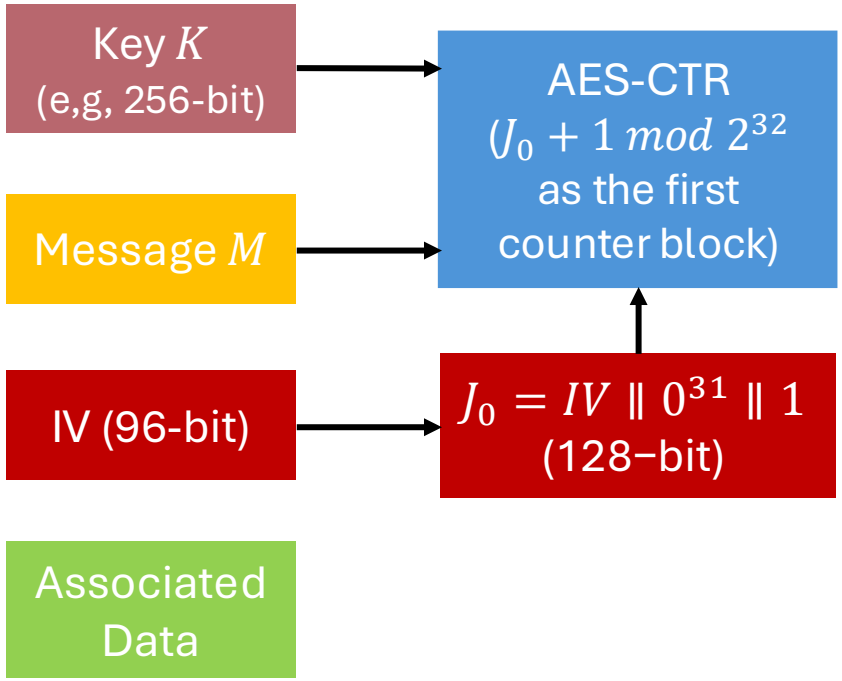
AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication



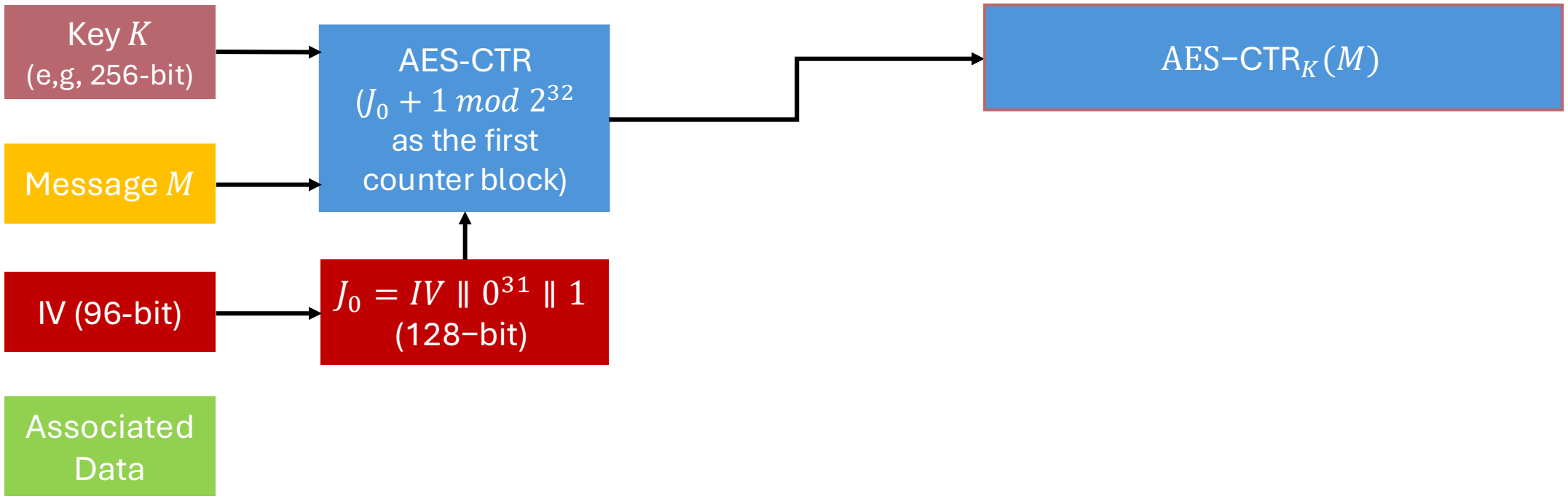
AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication



AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication



AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication

Key K
(e,g, 256-bit)

Message M

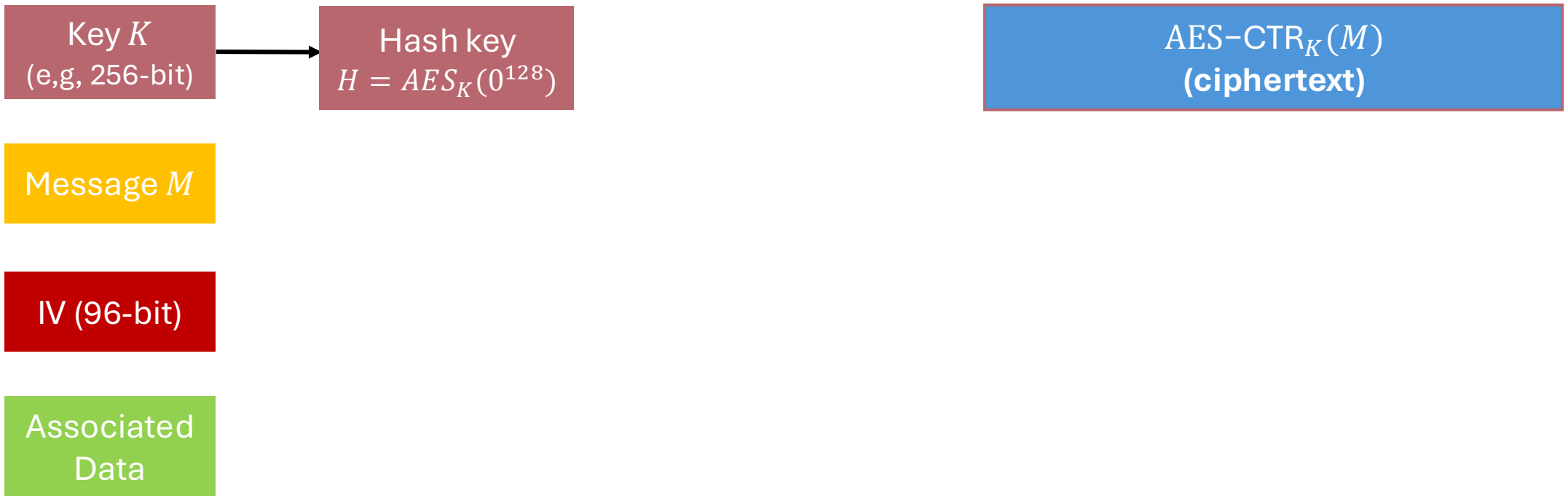
IV (96-bit)

Associated Data

$\text{AES-CTR}_K(M)$
(ciphertext)

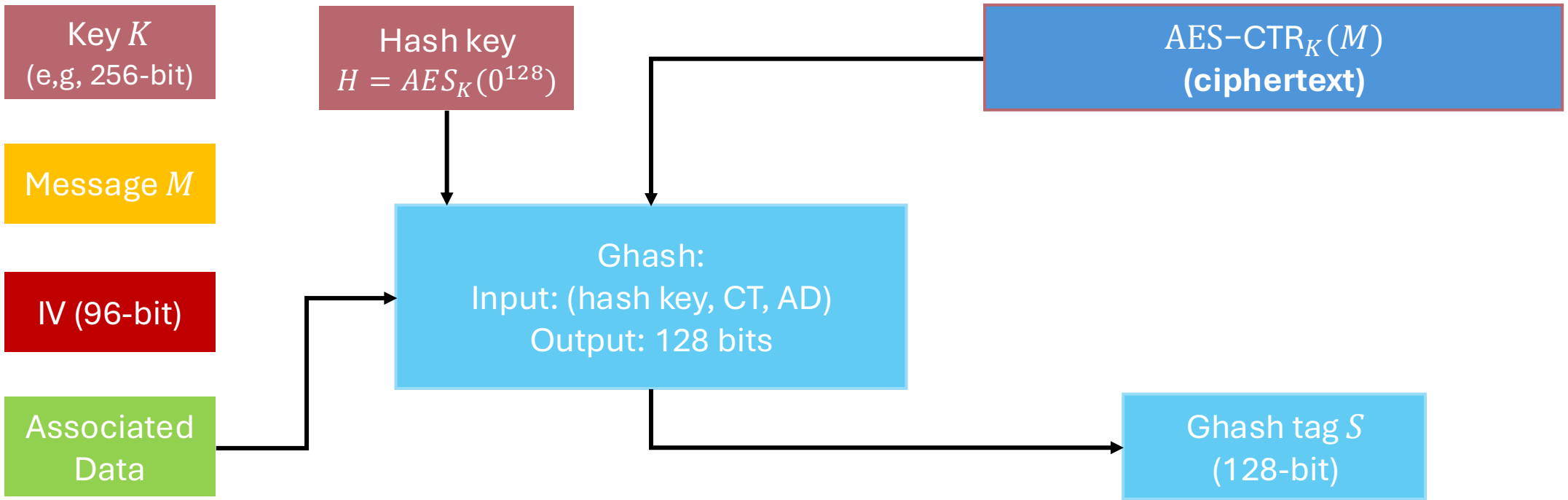
AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication



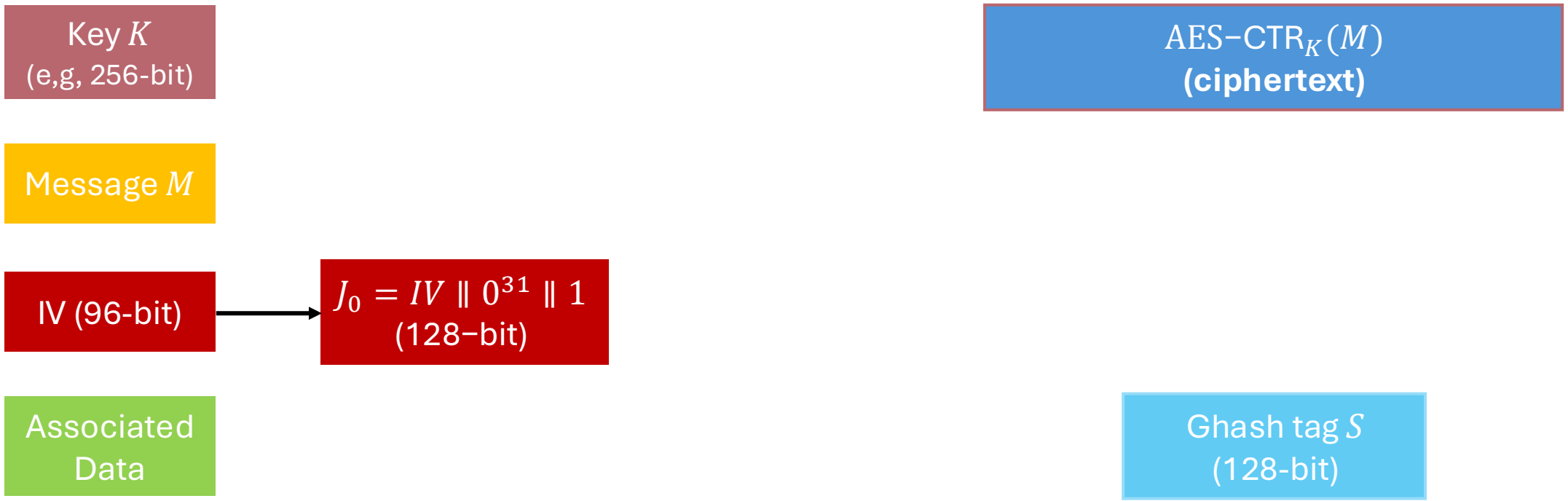
AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication



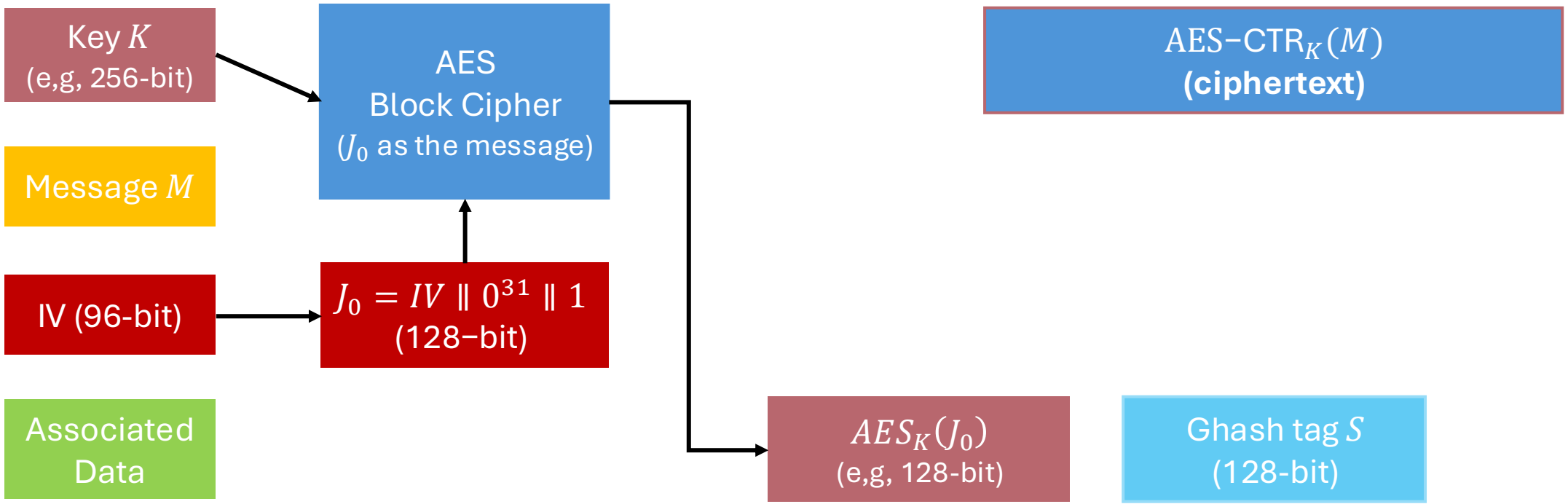
AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication



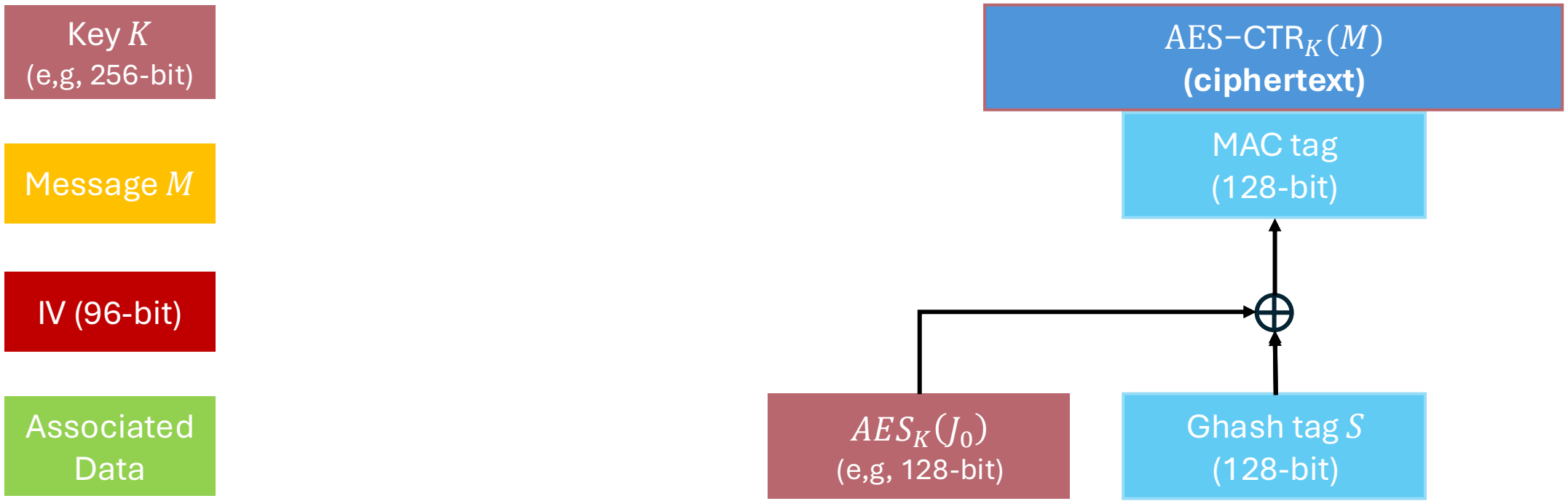
AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication



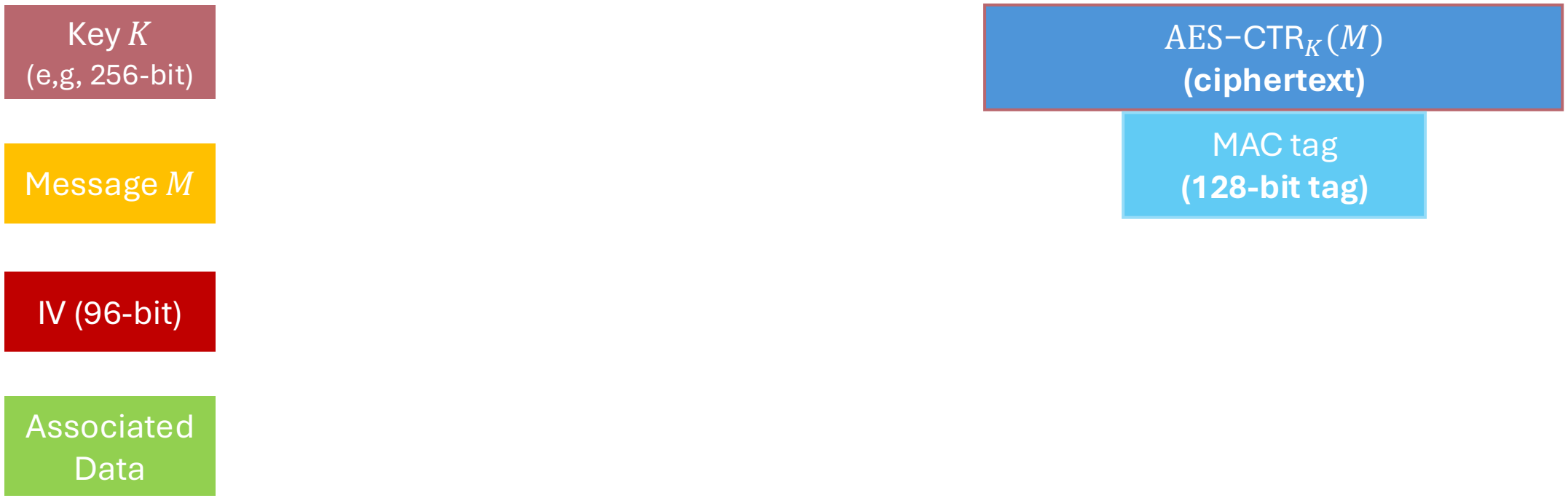
AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication



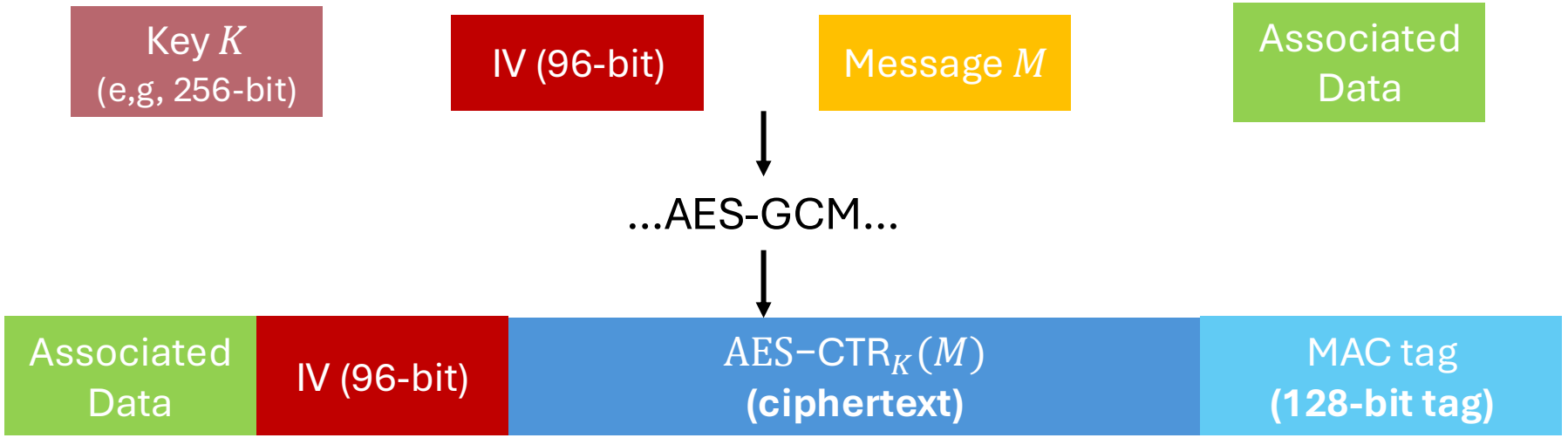
AEAD

- AES-GCM (AES in Galois/Counter Mode): AES-CTR + GHash-based authentication



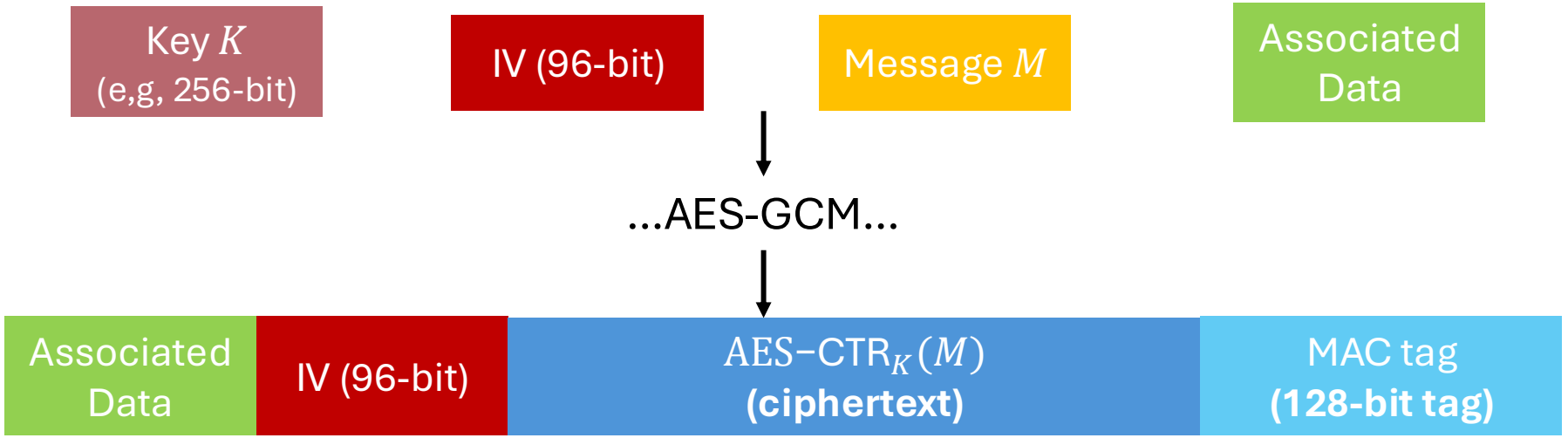
AEAD

- AES-GCM


Key K
(e,g, 256-bit)
IV (96-bit)
Message M
Associated Data
...AES-GCM...
Associated Data
IV (96-bit)
$AES\text{–}CTR_K(M)$
(ciphertext)
MAC tag
(128-bit tag)

AEAD

- AES-GCM



- AD and IV can be public. **But IV should be unique (per key)!**

Padding

- Many cryptographic algorithms have fixed-length inputs or process data in fixed-size blocks
  - E.g., AES block cipher
- However, real-world messages can have arbitrary lengths.

- We need **Padding** to add extra data to a message to meet the input requirement

Padding

- Many cryptographic algorithms have fixed-length inputs or process data in fixed-size blocks
  - E.g., AES block cipher
- However, real-world messages can have arbitrary lengths.

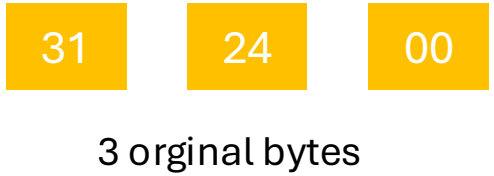
- We need **Padding** to add extra data to a message to meet the input requirement
  - SHA-256: Uses padding to divide messages (and their lengths) into complete 64-byte blocks
  - AES-CBC: Padding messages to be multiple of 16 bytes
  - AES-CTR: Not required
  - (more in the future…)

Padding

- Padding must be **unambiguous**

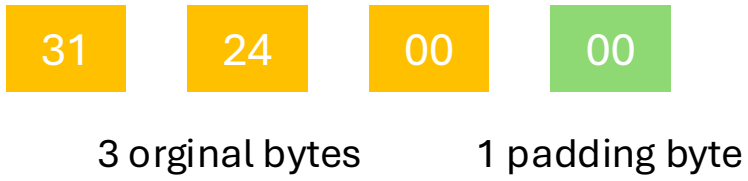
Padding

- Padding must be **unambiguous**
- Wrong example: Zero padding for 4-byte block size



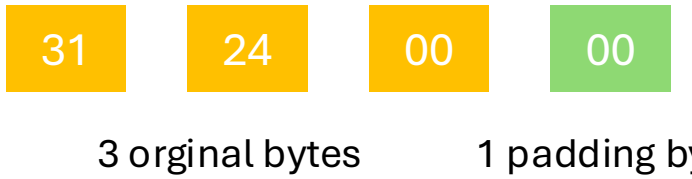
Padding

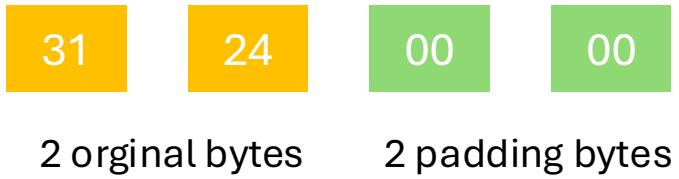
- Padding must be **unambiguous**
- Wrong example: Zero padding for 4-byte block size



Padding

- Padding must be **unambiguous**
- Wrong example: Zero padding for 4-byte block size





Padding

- Padding must be **unambiguous**
- Wrong example: Zero padding for 4-byte block size



- PKCS#7: Append $p = B - (|M| \ mod \ B)$ bytes, each with value p (when $p \neq 0$)

Padding

- Padding must be **unambiguous**

- Wrong example: Zero padding for 4-byte block size

- PKCS#7: Append $p = B - (|M| \ mod \ B)$ bytes, each with value p (when $p \neq 0$).
- Example for $B = 4$:



Padding

- Padding must be **unambiguous**

- Wrong example: Zero padding for 4-byte block size

- PKCS#7: Append $p = B - (|M| \ mod \ B)$ bytes, each with value p (when $p \neq 0$).
- Example for $B = 4$:



- When p = 0 (padding $B = 4$ bytes with value B)

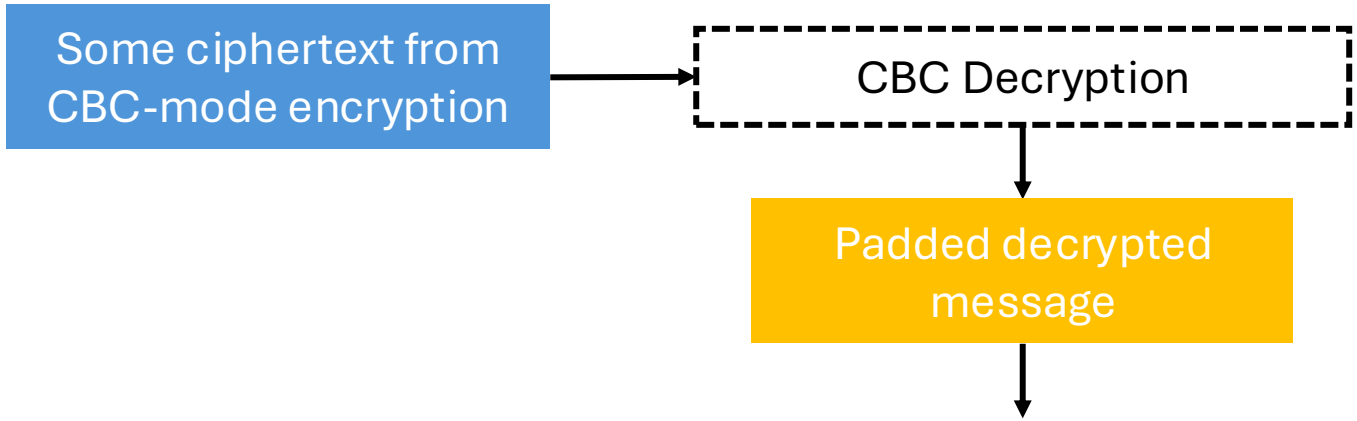


Padding

- **Can padding introduce security vulnerabilities?**

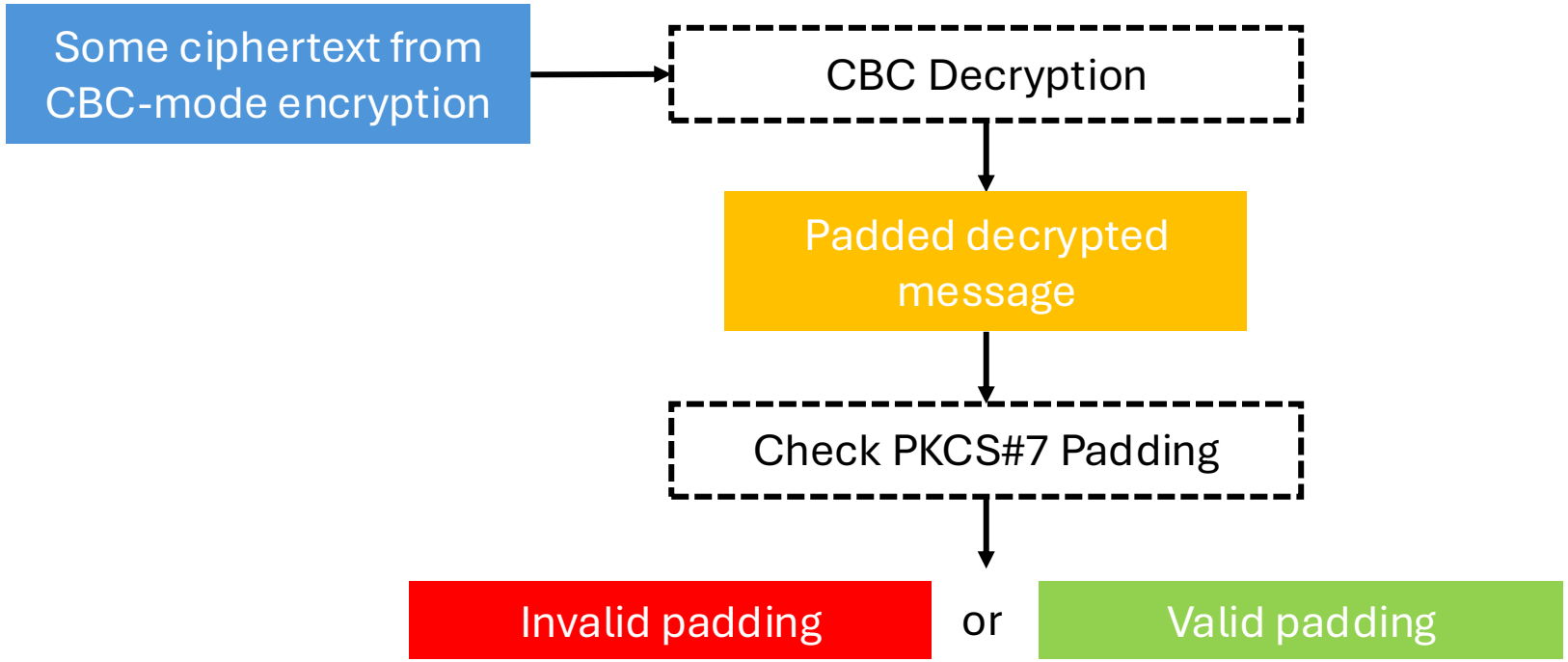
Padding

- Can padding introduce security vulnerabilities?
- **Example: CBC mode**



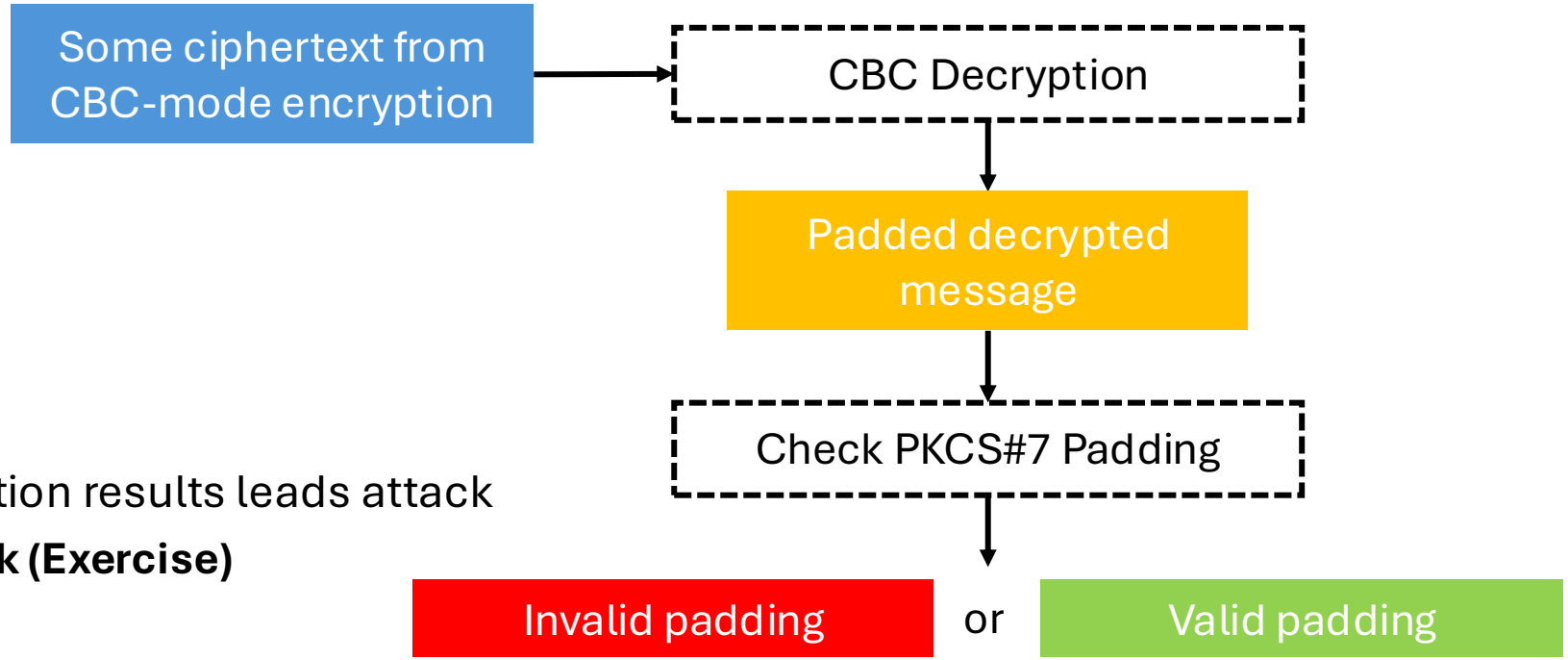
Padding

- Can padding introduce security vulnerabilities?
- **Example: CBC mode**



Padding

- Can padding introduce security vulnerabilities?
- Example: CBC mode



- Revealing padding validation results leads attack
  - **Padding oracle attack (Exercise)**

Padding

- Padding must be **unambiguous.**
  - The original message should always be recoverable.

- Use well-established padding schemes
  - Such as PKCS#7

- **Padding does not provide integrity.**
  - Attackers may modify ciphertexts without knowing the secret key.

- **Do not reveal whether padding is valid.**
  - E.g., through error messages or timing differences.

- Prefer AEAD in real-world applications, such as AES-GCM.

Key Length vs. Security Strength

Longer keys do not necessarily mean stronger security. Cryptographic security depends on the algorithm, its construction, and how the keys are generated and used.

Key Length vs. Security Strength

- **Key length:** The number of bits in a cryptographic key (AES-128/192/256)

- **Security Strength:** Roughly, the logarithm of the computational effort required to break a cryptographic scheme. Sometimes expressed as “bit security”
  - E.g., brute-force an AES-128 key requires roughly 2^{128} operations, which means “128-bit security”.

Key Length vs. Security Strength

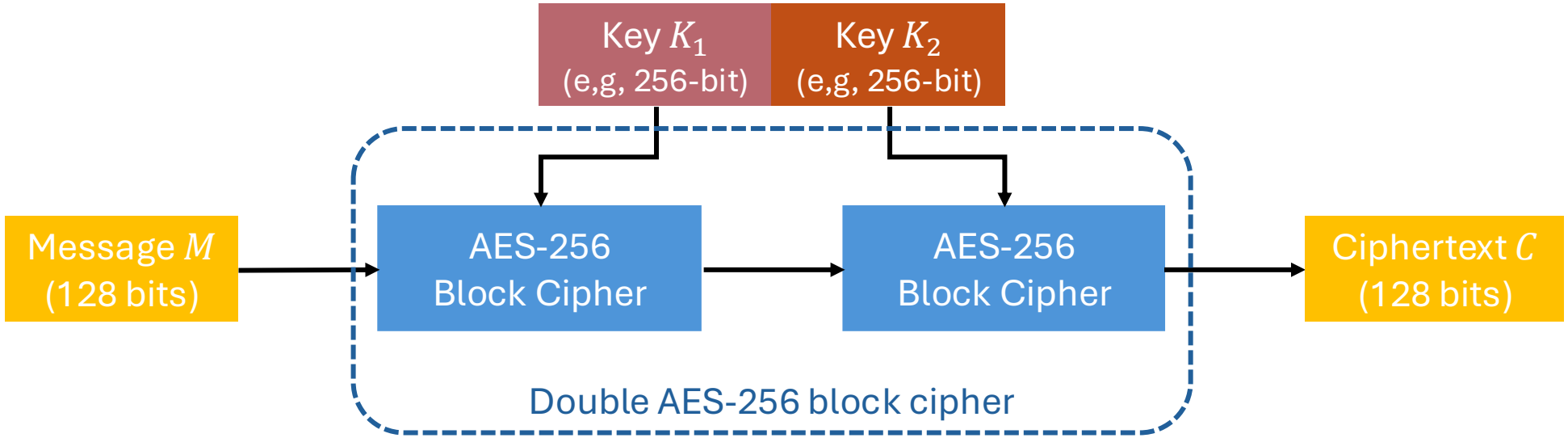
- **Key length:** The number of bits in a cryptographic key (AES-128/192/256)

- **Security Strength:** Roughly, the logarithm of the computational effort required to break a cryptographic scheme. Sometimes expressed as “bit security”
  - E.g., brute-force an AES-128 key requires roughly 2^{128} operations, which means “128-bit security”.

- AES-256 seems to be having the strongest security (i.e., 256-bit security)
- But if we use two independent AES-256 keys, do we obtain 512-bit security?

Key Length vs. Security Strength

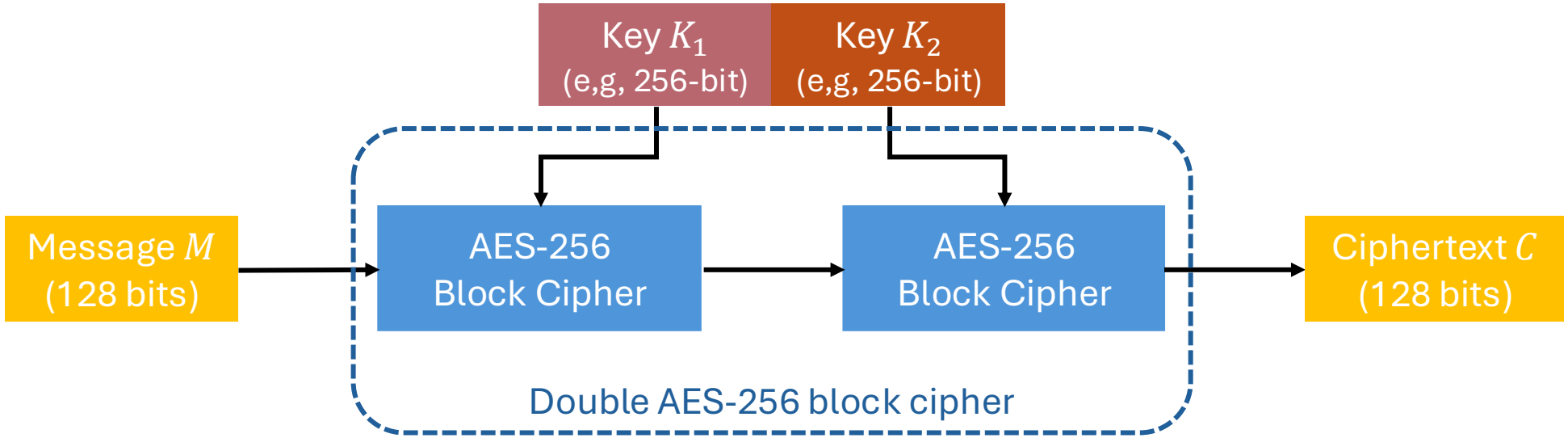
- Consider the following Double AES-256 block cipher:



$$K = K_1 \parallel K_2 \text{ and } C = \text{AES256}_{K_2}\left(\text{AES256}_{K_1}(M)\right)$$

Key Length vs. Security Strength

- Consider the following Double AES-256 block cipher:

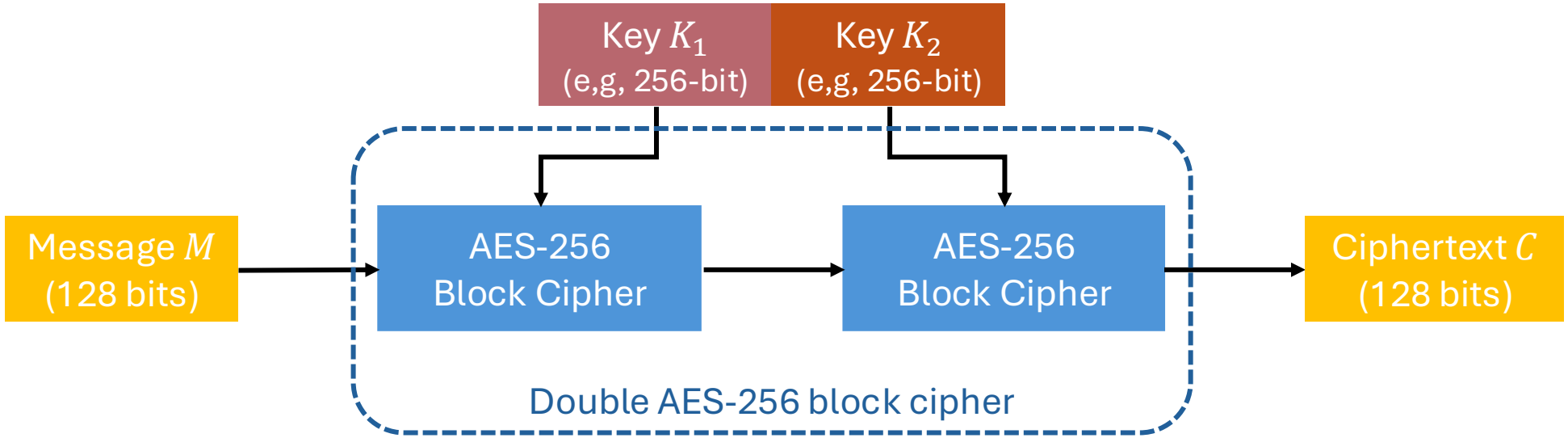


$$K = K_1 \parallel K_2 \text{ and } C = \text{AES256}_{K_2}\left(\text{AES256}_{K_1}(M)\right)$$

- **Does Double-AES-256 has 512-bit security?**

Key Length vs. Security Strength

- Consider the following Double AES-256 block cipher:



$$K = K_1 \parallel K_2 \text{ and } C = \text{AES256}_{K_2}\left(\text{AES256}_{K_1}(M)\right)$$

- **Does Double-AES-256 has 512-bit security?** **Meet-in-the-middle attack => Less than 512 bit security!**

Key Length vs. Security Strength

- Stronger security is not free

	AES-128	AES-256
Key size	128	256
Numof of encryption rounds	10	14
Block size	128	128
Computational cost	Lower	Higher

Key Length vs. Security Strength

- Key length cannot compensate for insufficient entropy
  - **Weak key generation:** K = HKDF(seed, info, 256), but $H_\infty(\text{seed}) = 40$, then brute-force the seed requires about 2^{40} operations, which means K has less than 256 bit security, even it has 256 bits.
  - **Key length ≠ security strength**

Key Length vs. Security Strength

- **Choose an appropriate security level**
  - AES-128 is sufficient for most daily applications
  - AES-256 offers a higher security margin at some computational cost.
- **Use securely generated keys**
  - For example, a 256-bit key derived from a weak password does not provide 256-bit security.
- **Follow secure cryptographic practices**
  - Even strong keys cannot prevent attacks caused by incorrect usage, such as nonce reuse in AES-GCM or AES-CBC.
  - A scheme could have 128-bit security, but implementation flaws do not…

Summary of Symmetric Primitives

- Hash functions: SHA2, SHA3, SHAKE
- HMAC and HKDF.(Extract, Expand)
- AES, AES-CBC, AES-CTR
- AE: Encrypt-then-MAC
- AEAD: AES-GCM
- Padding
- Key length and security strength

Summary of Symmetric Primitives

- Hash functions: SHA2, SHA3, SHAKE
- HMAC and HKDF.(Extract, Expand)
- AES, AES-CBC, AES-CTR
- AE: Encrypt-then-MAC
- AEAD: AES-GCM
- Padding
- Key length and security strength

- Next Topic: **Public-key Cryptography**

Exercise

- Practice your Rust skill

1. **AES-CTR:** Read the AES-CTR example code
2. **Encrypt-then-MAC**: Implement Encrypt-then-MAC construction using your AES-CTR code and HMAC
3. **AEAD:** Read the AEAD example code in L1 again

Exercise

- Deepen your understanding of cryptography

1. **AES-CBC with a reused IV:** Consider AES-CBC encryption with a fixed IV that is reused for every encryption. For simplicity, assume all messages consist of exactly one 128-bit block:

$$\text{AES-CBC}(K, M): \; C = AES_K(M \oplus IV)$$

where IV is some fixed 128-bit string.

- Suppose that now an adversary knows two possible plaintexts $M_0 \neq M_1$, and observes a ciphertext C^* encrypting **one of them.** How can the adversary determine which plaintext was encrypted if it is provided some interface to $AES_K(\cdot)$?

Exercise

- Deepen your understanding of cryptography

2. On the insecurity of Encrypt-and-MAC: Consider the following encrypt-and-mac construction: Suppose that Alice has K_e as the encryption key and K_s as the MAC key. She encrypt her message M in the following way:

$$c = Enc(K_e, M), \qquad t = \mathrm{HMAC}(K_m, M)$$

and the ciphertext is $ct = (c, t)$. Assume the encryption is AES-CTR with random initial block.

1. Alice encrypts the same message M twice. Will the two ciphertexts be identical? If not, which parts of the ciphertexts are identical?
2. An adversary observes $ct_1 = (c_1, t_1)$ and $ct_2 = (c_2, t_2)$, where $t_1 = t_2$, then what information can the adversary infer about their messages?

Exercise

- Deepen your understanding of cryptography

3. For the PKC#7 padding, what if we pad zeros instead of padding? Is it unambiguous? If so, try to prove it (or give the reasons).

4. Exploring Other Padding Schemes: Besides PKCS#7, there are other padding schemes, such as ANSI X9.23 and ISO/IEC 7816-4 padding. Investigate these two padding schemes and answer the following questions:

(1) How does each scheme pad a message to a multiple of the block size?

(2) Why is each scheme unambiguous (i.e., why can the original message always be recovered)?

(3) What happens if the original message length is already a multiple of the block size?

Hint: Use a block size of 4 bytes and illustrate your answers with simple examples.

Exercise

- Deepen your understanding of cryptography

5. CBC Padding Oracle Attack. Consider AES-CBC with PKCS#7 padding. For simplicity, assume the ciphertext contains **only one encrypted block.**

- $(IV, C) = \text{AES-CBC.Enc}(K, M) \coloneqq \left(IV, AES.E_K(IV \oplus M')\right)$, where IV is random and M' is the PKC#7 padded message of M
- $M' = \text{AES-CBC.Dec}(K, C) \coloneqq AES.D_K(C) \oplus IV$

Consider the following game: A server has a secret key K and encrypted M, with the ciphertext (IV, C). An adversary may modify the IV part (in whatever way it wants) and submit the modified (IV, C) to the server. The server will return "Valid padding" if the decrypted M' has valid PKCS#7 padding.

(Continued on next page...)

Exercise

- Deepen your understanding of cryptography

5. CBC Padding Oracle Attack (continue…)

Suppose that now:

(1) The last byte of the original IV is 0xA0.

(2) The adversary changes the last byte of IV to 0xA5, keeping all other bytes unchanged. The server responds “Valid padding”.

(3) The adversary then changes the second-to-last byte of the modified IV, keeping the last byte as 0xA5. The server again responds “Valid padding”.

Questions:

(a) What is the padding length after the first modification? Why?

(b) What is the last byte of $AES.D_K(C)$?

(c) What is the last byte of the original padded plaintext M'