IERG 4130 – 2026 Fall

香港中文大學
CUHK

MACs and Authenticated Encryption

Yajin ZHOU

https://yajin.org

Credits: the slides are from CS161 of UC Berkeley,

Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	• One-time pads • Block ciphers with chaining modes (e.g. AES-CBC) • Stream ciphers	• RSA encryption • ElGamal encryption
Integrity, Authentication	• **MACs (e.g. HMAC)**	• Digital signatures (e.g. RSA signatures)

- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)

- Key management (certificates)
- Password management

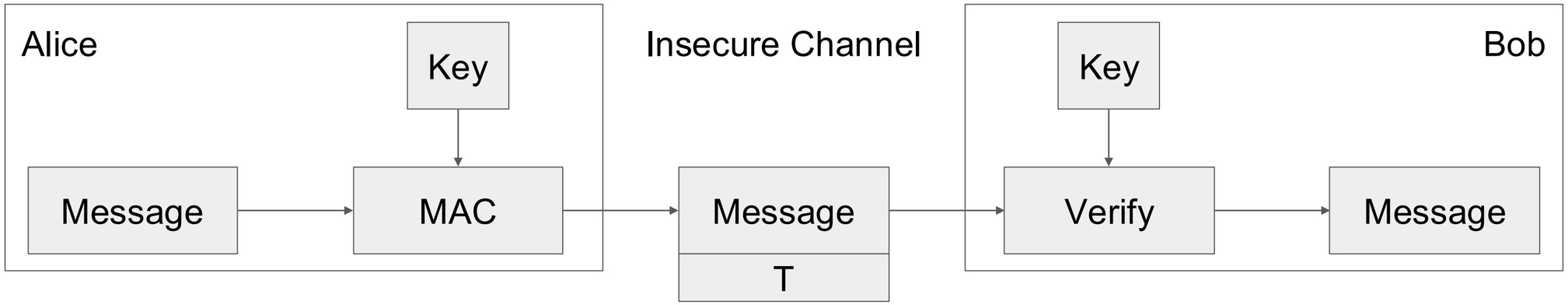
How to Provide Integrity



- **Reminder: We're still in the symmetric-key setting**
  - Assume that Alice and Bob share a secret key, and attackers don't know the key
- We want to attach some piece of information to ***prove* that someone** with the key sent this message
  - This piece of information can only be generated by someone with the key

MACs: Usage

- **Alice wants to send *M* to Bob, but doesn't want Mallory to tamper with it**
- **Alice sends *M* and *T* = MAC(*K*, *M*) to Bob**
- **Bob receives *M* and *T***
- **Bob computes MAC(*K*, *M*) and checks that it matches *T***
- If the MACs match, Bob is confident the message has not been tampered with (integrity)



MACs: Definition

- **Two parts:**
  - KeyGen() → *K*: Generate a key *K*
  - MAC(*K*, *M*) → *T*: Generate a tag *T* for the message *M* using key *K*
    - Inputs: A secret key and an arbitrary-length message
    - Output: A fixed-length **tag** on the message
- **Properties**
  - **Correctness**: Determinism
    - Note: Some more complicated MAC schemes have an additional Verify(*K*, *M*, *T*) function that don't require determinism, but this is out of scope
  - **Efficiency**: Computing a MAC should be efficient
  - **Security**: EU-CPA (existentially unforgeable under chosen plaintext attack)

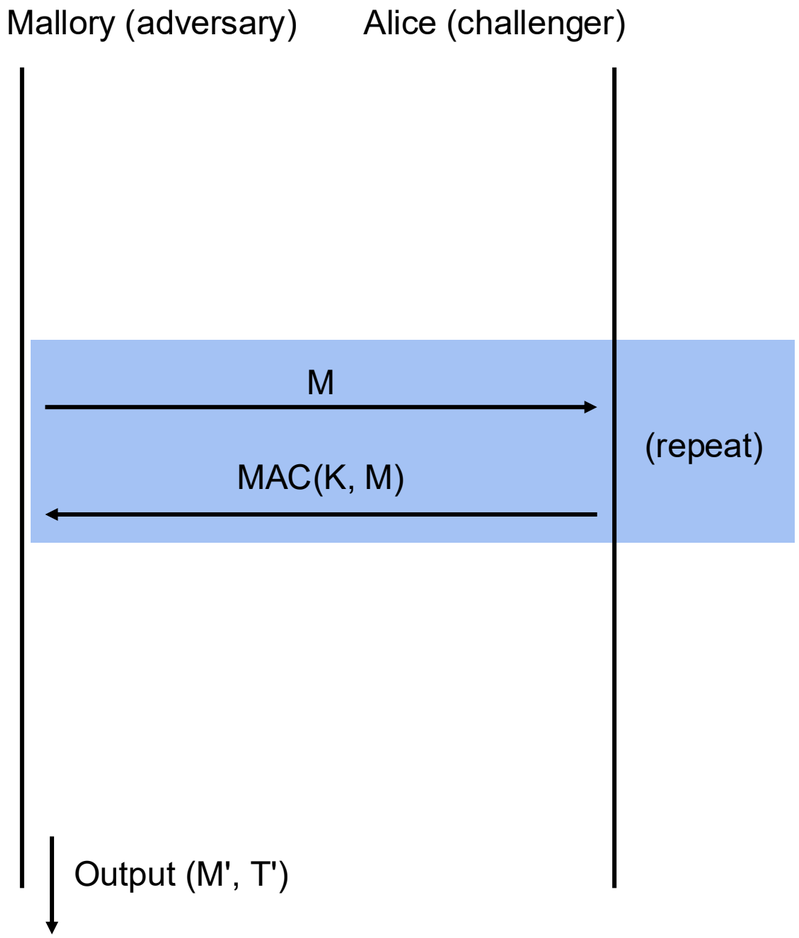
Defining Integrity: EU-CPA



- A secure MAC is **existentially unforgeable**: without the key, an attacker cannot create a valid tag on a message
  - Mallory cannot generate MAC(K, M') without K
  - Mallory cannot find any $M' \neq M$ such that MAC(K, M') = MAC(K, M)
- Formally defined by a security game: existential unforgeability under chosen-plaintext attack, or EU-CPA
- **MACs should be unforgeable under chosen plaintext attack**
  - Intuition: Like IND-CPA, but for integrity and authenticity
  - Even if Mallory can trick Alice into creating MACs for messages that Mallory chooses, Mallory cannot create a valid MAC on a message that Alice hasn't sent before

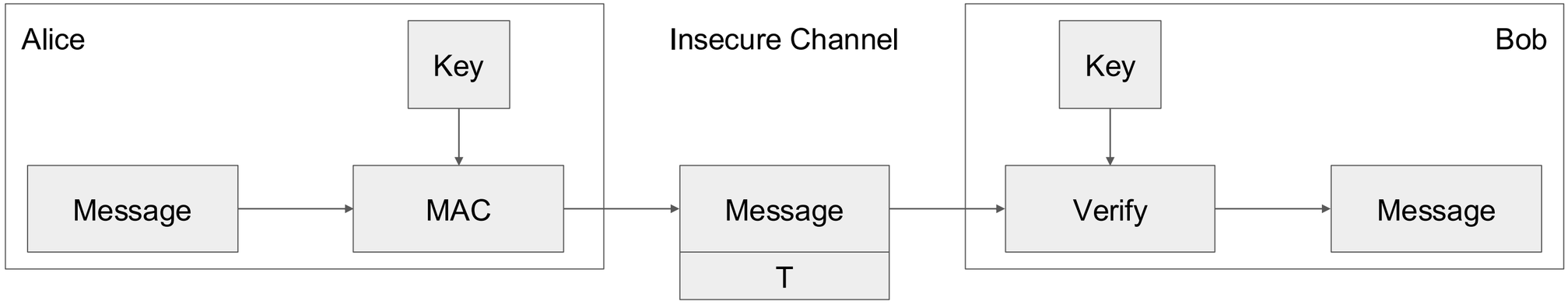
Defining Integrity: EU-CPA

1. **Mallory may send messages to Alice and receive their tags**
2. **Eventually, Mallory creates a message-tag pair (*M'*, *T'*)**
   - *M'* cannot be a message that Mallory requested earlier
   - If *T'* is a valid tag for *M'*, then Mallory wins. Otherwise, she loses.
   - A scheme is EU-CPA secure if for *all* polynomial time adversaries, the probability of winning is 0 or negligible



MACs: Usage

- **Alice wants to send *M* to Bob, but doesn't want Mallory to tamper with it**
- **Alice sends *M* and *T* = MAC(*K*, *M*) to Bob**
- **Bob receives *M* and *T***
- **Bob computes MAC(*K*, *M*) and checks that it matches *T***
- If the MACs match, Bob is confident the message has not been tampered with (integrity)



Example: NMAC

- **Can we use secure cryptographic hashes to build a secure MAC?**
  - Intuition: Hash output is unpredictable and looks random, so let's hash the key and the message together
- **KeyGen():**
  - Output two random, n-bit keys K_1 and K_2, where n is the length of the hash output
- **NMAC(K_1, K_2, M):**
  - Output $H(K_1 \,||\, H(K_2 \,||\, M))$
- **NMAC is EU-CPA secure**
  - Provably secure if the underlying hash function is secure
- **Intuition: Using two hashes prevents a length extension attack**
  - Otherwise, an attacker who sees a tag for M could generate a tag for $M \,||\, M'$

Example: HMAC

- **Issues with NMAC:**
  - Recall: NMAC(K_1, K_2, M) = $H(K_1 \,||\, H\,(K_2 \,||\, M))$
  - We need two keys
  - NMAC requires the keys to be the same length as the hash output (n bits)
  - Can we use NMAC to design a scheme that uses one key?
- **HMAC(K, M):**
  - Compute K' as a version of K that is the length of the hash output
    - If K is too short, pad K with 0's to make it n bits (be careful with keys that are too short and lack randomness)
    - If K is too long, hash it so it's n bits
  - Output $H((K' \oplus opad) \,||\, H((K' \oplus ipad) \,||\, M))$

Example: HMAC



- **HMAC(*K*, *M*):**
  - Compute *K'* as a version of *K* that is the length of the hash output
    - If *K* is too short, pad *K* with 0's to make it *n* bits (be careful with keys that are too short and lack randomness)
    - If *K* is too long, hash it so it's *n* bits
  - Output $H((K' \oplus opad) \| H((K' \oplus ipad) \| M))$
- **Use *K'* to derive two different keys**
  - *opad* (outer pad) is the hard-coded byte **0x5c** repeated until it's the same length as *K'*
  - *ipad* (inner pad) is the hard-coded byte **0x36** repeated until it's the same length as *K'*
  - As long as *opad* and *ipad* are different, you'll get two different keys
  - For paranoia, the designers chose two very different bit patterns, even though they theoretically need only differ in one bit

HMAC Properties



- **HMAC(*K*, *M*) = *H*((*K*' ⊕ *opad*) || H((*K*' ⊕ *ipad*) || *M*))**
- **HMAC is a hash function, so it has the properties of the underlying hash too**
  - It is collision resistant
  - Given HMAC(*K*, *M*) and *K*, an attacker can't learn *M*
  - If the underlying hash is secure, HMAC doesn't reveal *M*, but it is still deterministic
- **You can't verify a tag *T* if you don't have *K***
  - This means that an attacker can't brute-force the message *M* without knowing *K*

Do MACs provide integrity?



- **Do MACs provide integrity?**
  - Yes. An attacker cannot tamper with the message without being detected
- **Do MACs provide authenticity?**
  - If a message has a valid MAC, you can be sure it came from *someone with the secret key*, but you can't narrow it down to one person
  - If only two people have the secret key, MACs provide authenticity: it has a valid MAC, and it's not from me, so it must be from the other person
- **Do MACs provide confidentiality?**
  - MACs are deterministic ⇒ No IND-CPA security
  - MACs in general have no confidentiality guarantees; they can leak information about the message
    - HMAC doesn't leak information about the message, but it's still deterministic, so it's not IND-CPA secure

Authenticated Encryption: Definition



- **Authenticated encryption** (**AE**): A scheme that simultaneously guarantees confidentiality and integrity (and authenticity, depending on how it is used) on a message
- **Two ways of achieving authenticated encryption:**
  - Combine schemes that provide confidentiality with schemes that provide integrity
  - Use a single scheme that is designed to provide both confidentiality and integrity

Combining Schemes: Let's design it together



- **You can use:**
  - An IND-CPA encryption scheme (e.g. AES-CBC): Enc(K, M) and Dec(K, M)
  - An unforgeable MAC scheme (e.g. HMAC): MAC(K, M)
- **First attempt: Alice sends Enc(K_1, M) and MAC(K_2, M)**
  - Integrity? Yes, attacker can't generate a valid MAC on a modified message
  - Confidentiality? No, the MAC is not IND-CPA secure
- Idea 1: Let's compute the MAC on the *ciphertext* instead of the plaintext:
  Enc(K_1, M) and MAC(K_2, Enc(K_1, M))
  - Integrity? Yes, attacker can't generate a valid MAC on a modified message
  - Confidentiality? Yes, the MAC might leak info about the ciphertext, but that's okay
- **Idea 2: Let's encrypt the MAC:** Enc(K_1, M || MAC(K_2, M))
  - Integrity? Yes, attacker can't generate a valid MAC on a modified message
  - Confidentiality? Yes, everything is encrypted

MAC-then-Encrypt or Encrypt-then-MAC?



- **Encrypt-then-MAC**
  - First compute Enc(K_1, M), then MAC the ciphertext: MAC(K_2, Enc(K_1, M))
- **MAC-then-encrypt**
  - First compute MAC(K_2, M), then encrypt the message and the MAC together: Enc(K_1, M || MAC(K_2, M))
- **Which is better?**
  - In theory, both are IND-CPA and EU-CPA secure if applied properly
  - MAC-then-encrypt has a theoretical flaw: not secure against chosen-ciphertext attacks
  - MAC-then-encrypt has an engineering flaw: You don't know if tampering has occurred until after decrypting
    - Attacker can supply arbitrary tampered input, and you always have to decrypt it
    - Passing attacker-chosen input through the decryption function can cause side-channel leaks
- **Always use encrypt-then-MAC because it's more robust to mistakes**

MAC-then-Encrypt or Encrypt-then-MAC?



- MAC-then-encrypt has a theoretical flaw: not secure against chosen-ciphertext attacks
  - Enc(K_1, M || MAC(K_2, M))
  - Block cipher needs the padding and after decryption it will check the padding
    - If padding format is not right -> error padding
    - If padding format is right -> bad MAC
  - The leak of information can give attacker extra information.
    - SSL 3.0 vulnerable to POODLE attack.

Key Reuse

- **Key reuse: Using the same key in two different use cases**
  - Note: Using the same key multiple times for the same use (e.g. computing HMACs on different messages in the same context with the same key) is not key reuse
  - However, if you use the same key for both computing HMAC or AES encryption -> reuse
- Reusing keys can cause the underlying algorithms to interfere with each other and affect security guarantees
  - Example: If you use a block-cipher-based MAC algorithm and a block cipher chaining mode, the underlying block ciphers may no longer be secure

Key Reuse



- **Simplest solution: Do not reuse keys! One key per *use*.**
  - Encrypt a piece of data and MAC a piece of data?
    - Different use (purpose); different key
  - MAC one of Alice's messages to Bob and MAC one of Bob's messages to Alice?
    - Different use (purpose); different key
  - Encrypt one of Alice's files and encrypt another one of Alice's files?
    - Same use (purpose); this is fine
  - Encrypt user metadata, encrypt file metadata, and encrypt file data?
    - You'll have to think about this in Project 2!

Summary



- **Inputs: a secret key and a message**
- **Output: a tag on the message**
- A secure MAC is unforgeable: Even if Mallory can trick Alice into creating MACs for messages that Mallory chooses, Mallory cannot create a valid MAC on a message that she hasn't seen before
  - Example: HMAC(K, M) = $H((K' \oplus opad) \,||\, H((K' \oplus ipad) \,||\, M))$
- **MACs do not provide confidentiality**

Summary

- Authenticated encryption: A scheme that simultaneously guarantees confidentiality and integrity (and authenticity) on a message
- Combine schemes that provide confidentiality with schemes that provide integrity and authenticity
  - MAC-then-encrypt: Enc(K_1, M || MAC(K_2, M))
  - Encrypt-then-MAC: Enc(K_1, M) || MAC(K_2, Enc(K_1, M))
  - Always use Encrypt-then-MAC because it's more robust to mistakes