MH4311 Cryptography

Lecture 12
Message Authentication Code

Wu Hongjun

Lecture Outline

- Classical ciphers
- Symmetric key encryption
- Hash function and Message Authentication Code
  - Birthday attack
  - Hash function
  - **Message Authentication Code**
    - **CBC-MAC, CMAC**
    - **HMAC**
- Public key encryption
- Digital signature
- Key establishment and management
- Introduction to other cryptographic topics

Recommended Reading

- CTP Section 4.4
- HAC Section 9.5
- NIST documents
  - CMAC: http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf
  - HMAC: http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf
- Wikipedia
  - Message Authentication Code
    http://en.wikipedia.org/wiki/Message_authentication_code
  - CBC-MAC
    http://en.wikipedia.org/wiki/CBC-MAC
  - CMAC
    http://en.wikipedia.org/wiki/CMAC
  - HMAC
    http://en.wikipedia.org/wiki/HMAC

Message Authentication

- Message Authentication: check whether the received message was sent from the sender
  - Message authentication implies data integrity
    - Data integrity: whether there is unauthorized modification to the message

Message Authentication

- Two approaches to authenticate message
  - Symmetric key approach:

    Message Authentication Code

    - Compress a message together with a secret key
    - The person who knows the secret key can verify the authenticity of the message
  - Public key approach: **digital signature**

Message Authentication Code

- Message Authentication Code (MAC)
  - MAC is an algorithm that compresses a **secret key** and a message with arbitrary length into a fixed length output (we call this output as authentication tag)
  - Verification: After receiving a message together with the authentication tag, generate a new authentication tag from the message, and compare it with the tag being received

Note: Another frequently used MAC refers to Media Access Control address (MAC address). For example, each network card in a computer has a 48-bit unique identifier (MAC address).

Message Authentication Code

- Two security parameters
  - The size of secret key
    - Should be large enough to resist brute force attack
  - The size of authentication tag
    - For a strong MAC with n-bit authentication tag, the probability of modifying (or forging) a message without being detected is 2^{-n}.
      - The size of authentication tag should not be too small
      - Typical authentication tag sizes: 64 bits, 128 bits, 160 bits

Message Authentication Code

- Repeated trials
  - For a strong MAC with n-bit authentication tag, the probability of modifying (or forging) a message without being detected is 2^{-n}.
  - However, if there is no mechanism that stops repeated trials, the chance to forge a message successfully can become higher.
    - Example: with 1024 trials, the probability that one of the forged messages & their tags can pass the verification with probability about 2^{-n+10}

Message Authentication Code

- Security requirements on MAC algorithm

Requirement 1: The key should remain secret even if it has been used to generate many authentication tags

- Large key size to resist brute force attack
- MAC algorithm should resist other key recovery attacks

Requirement 2: Without knowing the secret key, any forged or modified message can pass verification with negligible probability

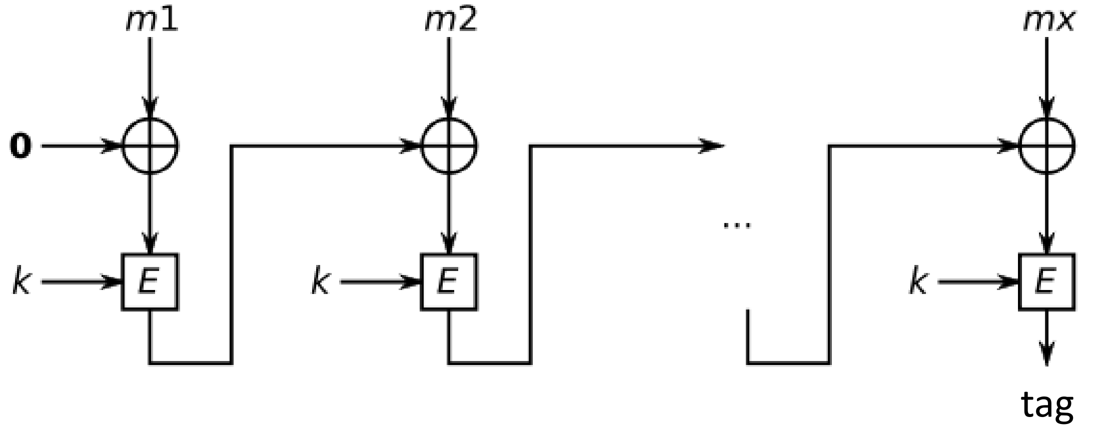
- An attacker may obtain the authentication tags of many messages, then try to forge some message (generate the authentication tag for a forged message without knowing the secret key)

MAC Constructions

- MAC algorithm based on block cipher
  - CBC-MAC
  - CMAC (NIST recommendation)
  - ……...
- MAC algorithm based on hash function
  - HMAC (NIST standard)
- Dedicated MAC algorithm

CBC-MAC

- Construct MAC algorithm from block cipher in CBC mode



- Differences between CBC-MAC and CBC encryption
  - CBC-MAC
    - Authentication, fixed public IV (0), output is the authentication tag
  - CBC encryption:
    - Encryption, random public IV, output is the ciphertext

CBC-MAC

- CBC-MAC is insecure
  - Attack 1. Use the authentication tags of two messages. The authentication tag of a new message can be forged without knowing the secret key
    - Example: (diagram in the next slide)

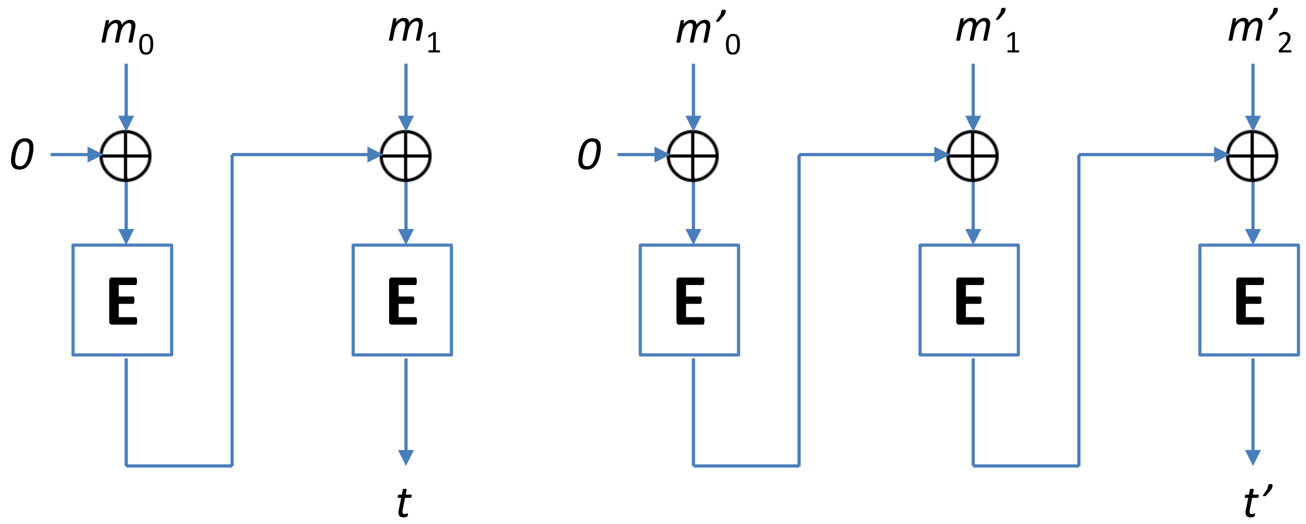
$M = m_0 \parallel m_1$ CBC-MAC(M) = t

$M' = m'_0 \parallel m'_1 \parallel m'_2$ CBC-MAC(M') = t'

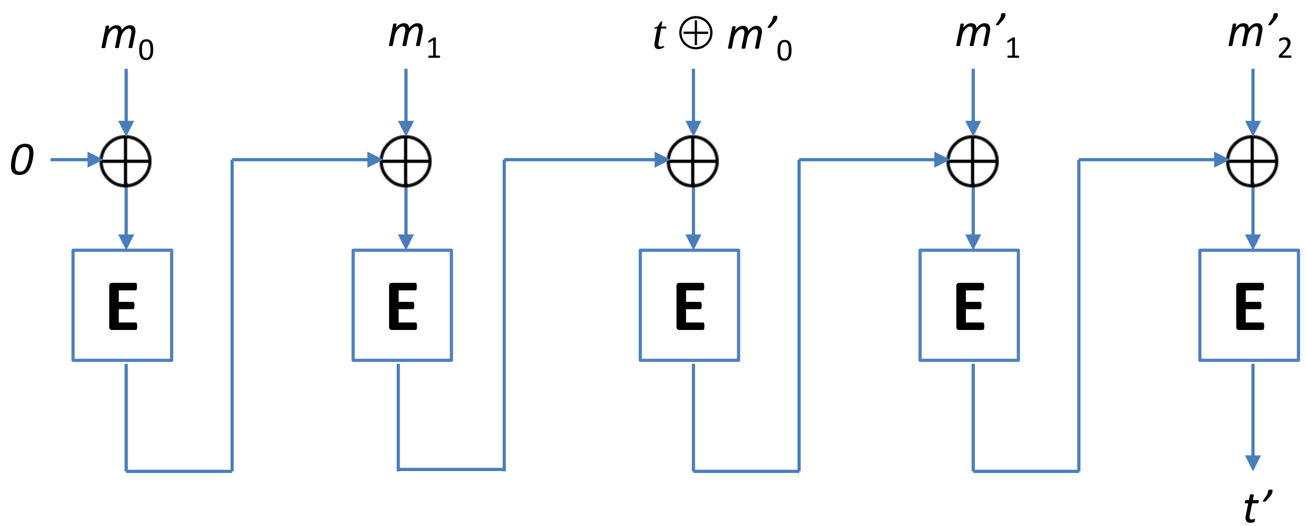
The new message: $m_0 \parallel m_1 \parallel (m'_0 \oplus t) \parallel m'_1 \parallel m'_2$

The forged tag: t'

(diagram in next page)



m0
m1
m'0
m'1
m'2
0
E
t
t'



m0
m1
t ⊕ m'0
m'1
m'2
0
E
t'

CBC-MAC

- CBC-MAC is insecure
  - Attack 2. With message length being appended to message, message can still be forged
    - Example (diagram in the next page):

Let M_1' represents M_1 with length padding

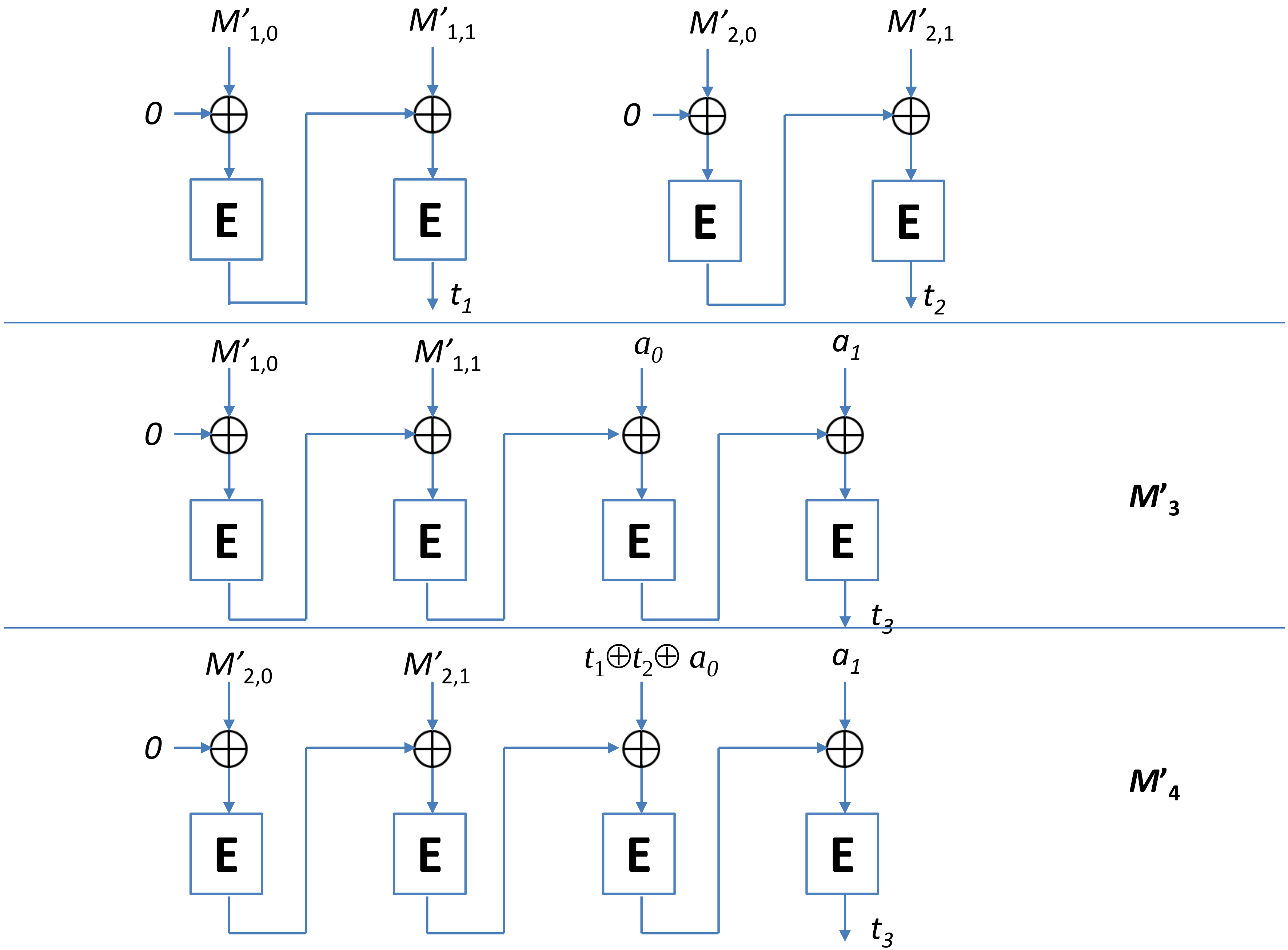
M_2' represents M_2 with length padding (the length of M_1 is the same as that of M_2)

Let M_3' represents M_3 with length padding,

$M'_3 = M_1' || a_0 || a_1$, where each a_i represents one message block.

Suppose now an attacker knows that the authentication tags of M_1, M_2, M_3 are t_1, t_2, t_3, then an attacker can generate the authentication tag of the following message without knowing the secret key:

$M'_4 = M_2' || (a_0 \oplus t_1 \oplus t_2) || a_1$, and the authentication tag is t_3



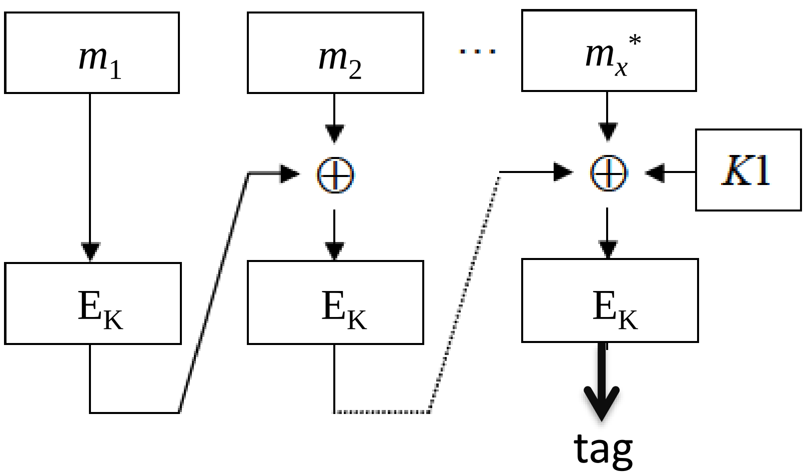
M'1,0
M'1,1
M'2,0
M'2,1
0
0
E
E
E
E
t1
t2
M'1,0
M'1,1
a0
a1
0
E
E
E
E
t3
M'3
M'2,0
M'2,1
t1⊕t2⊕ a0
a1
0
E
E
E
E
t3
M'4

CMAC (Cipher-based Message Authentication Code)

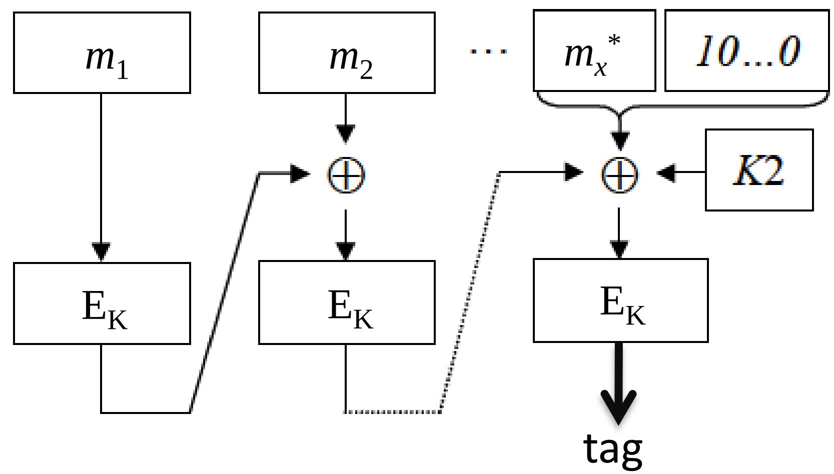
- CMAC is a NIST recommendation
- CMAC strengthens CBC-MAC
  - Use **an additional key (K_1 or K_2) for the last message block** to thwart the attacks on CBC-MAC
  - The last message block (to eliminate the ambiguity in the last block: full or partial, and how "partial")
    - **K_1 is used if it is a full block**
    - **K_2 is used if it is partial block**
      - **Pad the partial block by bit '1' followed by some zero bits**
  - Derive K_1 and K_2 from the secret key K
  - If the MAC size is less than the block size of block cipher, truncate some less significant bits

CMAC

- CMAC without considering truncation



No partial message block



with partial message block

CMAC

- Derive K_1 and K_2 from K
  - Let $W = E_K(0)$
  - Represent W as polynomial over GF(2)
    - If the block size of block cipher is 128 bits:
      - $K_1 = W \cdot x \mod (x^{128} + x^7 + x^2 + x + 1)$
      - $K_2 = K_1 \cdot x \mod (x^{128} + x^7 + x^2 + x + 1)$
    - If the block size of block cipher is 64 bits:
      - $K_1 = W \cdot x \mod (x^{64} + x^4 + x^3 + x + 1)$
      - $K_2 = K_1 \cdot x \mod (x^{64} + x^4 + x^3 + x + 1)$

CMAC

- *Security of CMAC
  - With $2^{n/2}$ authentication tags being generated from the same key, an attacker may generate the tag of a new message without knowing the secret key
    - Example:

Find two messages $M = m_0 \| m_1 \| m_2 \| m_3 \| m_4$

$M' = m'_0 \| m'_1 \| m_2 \| m_3 \| m_4$

with the same tag t (birthday attack). Then

$$E_K(m_0) \oplus m_1 = E_K(m'_0) \oplus m'_1$$

Now an attacker requests for the MAC of

$M'' = m_0 \| (m_1 \oplus x) \| y$ (for any x and y)

the attacker can forge the new message

$M''' = m'_0 \| (m'_1 \oplus x) \| y$

(the tag of M''' is the same as that of M'')

HMAC

- The Keyed-Hash Message Authentication Code (HMAC)
  - NIST standard (2002)
  - Construct a secure MAC from a strong hash function

HMAC

- How to construct a MAC algorithm from a strong hash function?
  - Method 1: key-prefix (insecure)
    - The key is prepended to the message, then hash

$$\mathrm{MAC}_K(M) = \mathrm{Hash}(K \,\|\, M)$$

('||' indicates concatenation)

    - Insecure
      - An attacker can extend the message and generate new tag of an extended message (such as MD5, SHA-1, SHA-256).
      - Details: Suppose that ($K \,\|\, M$) after padding becomes ($K \,\|\, M \,\|\, p$), and the attacker knows the tag value Hash($K \,\|\, M \,\|\, p$). Then an attacker can compute Hash($K \,\|\, M \,\|\, p \,\|\, x$) for an x due to the iterated nature of has function

HMAC

- How to construct a MAC algorithm from a strong hash function?
  - Method 2: Key-suffix (insecure)
    - A secret key is appended to the end of message, then hash

      $$\mathrm{MAC}_K(M) = \mathrm{Hash}(M \,\|\, K)$$

    - Not strong
      - A MAC can be forged with $2^{n/2}$ hash function computations
      - Details:
        - » An attacker can apply birthday attack to find two messages *M* and *M'* (with the same length) satisfying Hash(*M*)= Hash(*M'*)
        - » Then the attacker requests for the tag of *M*
        - » The attacker immediately knows the tag of *M'*

          (The forgery is successful, since the people who know the secret key *K* has never generated the tag of *M'*)

HMAC

- How to construct a MAC algorithm from a strong hash function?
  - Method 3: Key-prefix + key-suffix (envelope) (not strong)
    - A secret key is used as both prefix and suffix, then hash

$$MAC_K(M) = \mathrm{Hash}(K \,\|\, M \,\|\, K)$$

    - *Not strong(suppose *n*-bit key, and hash function with *n*-bit message digest)
      - The attacker requests the authentication tags of more than $2^{n/2}$ messages, then recovers the *n*-bit secret key
        - » Reason: depending on the message length, a secret key may be separated into two parts, and those two parts appear in two compression functions

HMAC

- How to construct a secure MAC algorithm from a strong hash function?
  - Method 3: key-prefix + key-suffix (envelope)
    - *Example: recovering 32 key bits
      - Suppose that the messages lengths are chosen so that after padding, the last block contains only n-32 key bits, i.e., the previous block contains the first 32 bits of the key
      - An attacker requests for the tags of about $2^{n/2+0.5}$ messages, with difference in the last message block (but the last padded block is the same).
        - » The attacker obtains two collisions MAC(x) = MAC(x'):
          - one collision happens before compressing the last padded block

            i.e., hash $(K \| x \| K_{32})$ = hash $(K \| x' \| K_{32})$

HMAC

- How to construct a secure MAC algorithm from a strong hash function?
  - Method 3: key-prefix + key-suffix (envelope)
    - Example: recovering 32 key bits (cont.)
      - Then an attacker requests for tags of MAC($x \| r \| y$) and MAC($x' \| r \| y$), where r is a 32-bit number, and y is a fixed arbitrary message.
        - » For all the 2^{32} values of r, if any particular r satisfies that $MAC_k(x \| r \| y) = MAC_k(x' \| r \| y)$, then we know that this r is equal to the first 32 bits of the key with high chance.
          - Reason: if hash ($K \| x \| K_{32}$) = hash ($K \| x' \| K_{32}$), and if some $r = K_{32}$, then hash ($K \| x \| r$) = hash ($K \| x' \| r$)

            => hash ($K \| x \| r \| y \| K$) = hash ($K \| x' \| r \| y \| K$),

            i.e., $MAC_k(x \| r \| y) = MAC_k(x' \| r \| y)$

HMAC

- How to construct a MAC algorithm from a strong hash function?
  - Method 4: **HMAC** (NIST standard, FIPS 198a)
    - Suppose: key size is less than or equal to message block size
    - Append '0's to the end of key K so that the resulting ***K'*** is with one message block length
      - Example: 128-bit key, and 512-bit message block size, then 384 '0' bits are appended to the key
    - Let opad and ipad be two constants with one block size
      - `opad = 0x5c5c5c5c5c5c` …………
      - `ipad = 0x363636363636` …………
    - Compute the MAC as

$$MAC_K(M) = \underline{Hash(\,(K' \oplus opad) \,||\, \underline{Hash((K' \oplus ipad) \,||\, M)}\,)}$$

HMAC

More specifically, HMAC is computed by applying the hash function twice:

$$\text{temp} = \text{Hash}((K' \oplus \text{ipad}) \,\|\, M);$$

$$\text{MAC}_K(M) = \text{Hash}(\,(K' \oplus \text{opad}) \,\|\, \text{temp});$$

Authentication and Encryption

Authentication and Encryption

- For secure communication,
  - Authentication is always necessary
  - Encryption may be optional
    - Depending on the sensitivity of the content
- In general, both authentication and encryption are needed for secure communication.

Authentication and Encryption

- If both symmetric key encryption and authentication are required
  - It is recommended to encrypt the message first, then authenticate the ciphertext (encrypt-then-authenticate)
    - If we compute the authentication tag from the plaintext, then encrypt the plaintext (authenticate-then-encrypt), the padding information in some cipher modes are not protected by authentication, then the encryption may become insecure. (For example, POODLE attack against the CBC encryption in TLS/SSL. In TLS/SSL, strong HMAC and strong AES-CBC are used, but the early versions are insecure.)

Authentication and Encryption

- In recent years, there is demand to combine authentication and encryption into a single algorithm for better security and efficiency
  - The combined algorithm is called authenticated-encryption algorithm
  - In the coming TLS 1.3, it is required that authenticated-encryption algorithms must be used
    - Currently, there is only one NIST recommendation on authenticated-encryption algorithm: AES-GCM (NIST SP 800-38D)
    - Research are needed on authentication-encryption algorithm

Authentication and Encryption

- Current research on authenticated-encryption algorithm
  - CAESAR competition (2014 to 2018)

    (Competition on Authenticated Encryption:

    Security, Applicability, and Robustness)
    - 56 submissions received in March 2014
    - 7 finalists (selected in March 2018)
    - Winner(s) will be announced in 2018

Authentication and Encryption

- Coming research on authenticated-encryption algorithm
  - NIST lightweight crypto competition (2019 -- ?)
    - To design authenticated-encryption algorithms for resource constrained applications (low cost, low power, low energy, …)

Summary

- Message Authentication Code
  - Compresses a secret key and a message into an authentication tag with fixed length
  - MAC based on block cipher
    - CBC-MAC
    - CMAC (NIST recommendation)
  - MAC based on hash function
    - HMAC (NIST standard)
- Authentication and encryption
  - Use encryption-then-authentication approach
  - Use the authenticated-encryption algorithms