COMP412 Computer Security

Lec 09 Message Authentication Codes

Dr. Xiaochen Yuan
2021/2022

Contents

Message Integrity

The cryptography systems that we have studied so far provide secrecy, or confidentiality,
but not **integrity**.

However, there are occasions
where we may not even need secrecy
but instead must have integrity.

Document and Fingerprint

One way to preserve the integrity of a document is
through the use of a fingerprint.

If Alice needs to be sure that
the contents of her document will not be changed,
she can put her fingerprint
at the bottom of the document.

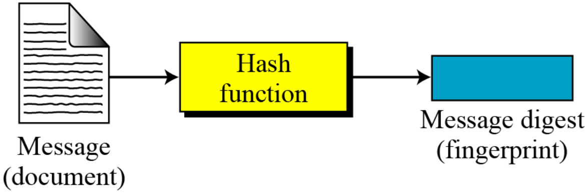
Eve cannot modify the contents of this document
or create a false document
because she cannot forge Alice's fingerprint.
Fingerprint on the document can be compared
to Alice's fingerprint on file.

Message and Message Digest

The electronic equivalent of
the *document* and *fingerprint* pair
is the *message* and *digest* pair.

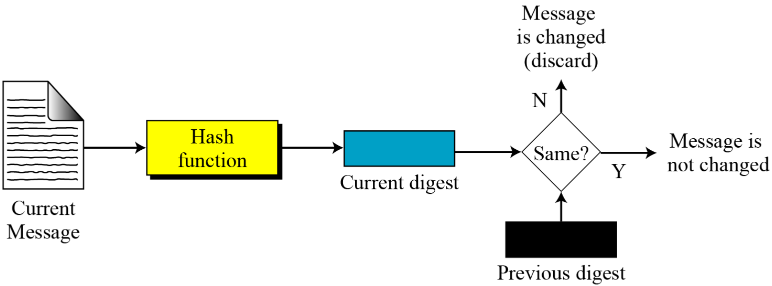
Note that the message digest needs to be safe from change.

To preserve the integrity of a message,
the message is passed through an algorithm
called a *cryptographic hash function*
which creates a compressed image of the message.



Credit: Figure 11.1 in *Cryptography and Network Security* ***Message and digest***

Checking Integrity

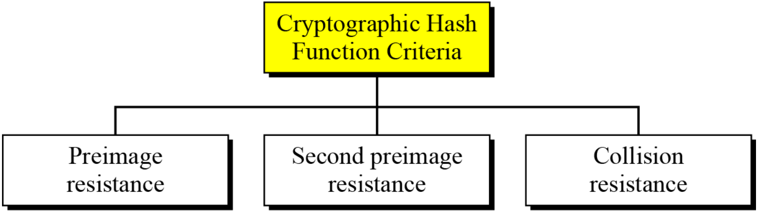


Credit: Figure 11.2 in *Cryptography and Network Security* ***Checking integrity***

Cryptographic Hash Function Criteria

A cryptographic hash function must satisfy three criteria:

preimage resistance,
second preimage resistance,
collision resistance.



Credit: Figure 11.3 in *Cryptography and Network Security*

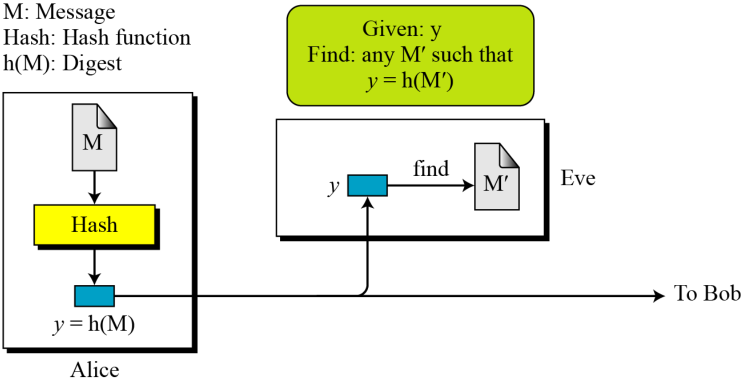
Criteria of a cryptographic hash function

Cryptographic Hash Function Criteria

Preimage Resistance

Preimage Attack

Given: $y = h(M)$ **Find:** M' such that $y = h(M')$



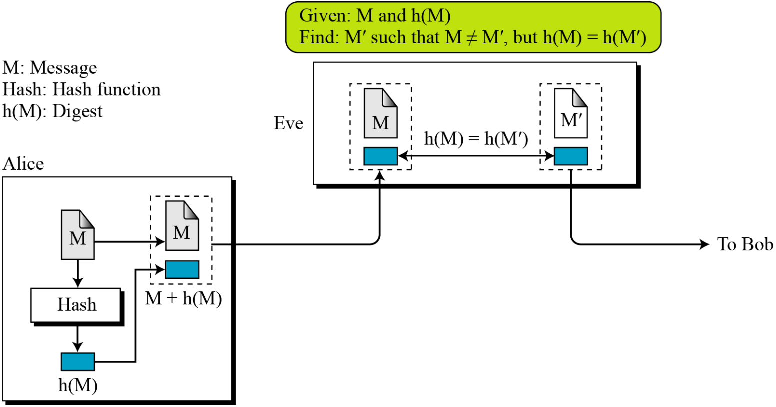
Credit: Figure 11.4 in *Cryptography and Network Security* ***Preimage***

Cryptographic Hash Function Criteria

Second Preimage Resistance

Second Preimage Attack

Given: M and h(M) **Find: M′ ≠ M such that h(M) = h(M′)**



Credit: Figure 11.5 in *Cryptography and Network Security* ***Second preimage***

Cryptographic Hash Function Criteria

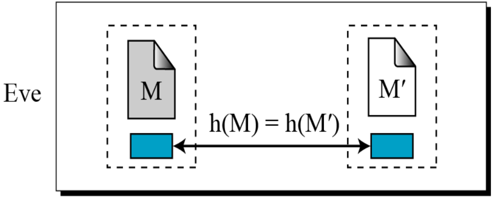
Collision Resistance

Collision Attack

Given: none **Find: $M' \neq M$ such that $h(M) = h(M')$**

M: Message
Hash: Hash function
h(M): Digest

Find: M and M′ such that $M \neq M'$, but $h(M) = h(M')$



Credit: Figure 11.6 in *Cryptography and Network Security* ***Collision***

Contents

Attacks on Communications across Network

1. Disclosure: encryption
2. Traffic analysis: encryption

3. Masquerade: message authentication
4. Content modification: message authentication
5. Sequence modification: message authentication
6. Timing modification: message authentication

7. Source repudiation: digital signatures
8. Destination repudiation: digital signatures

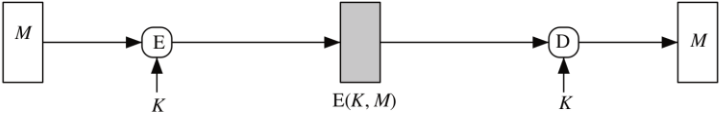
Authentication

- Receiver wants to verify:
  1. Contents of the message have not been modified (***data authentication***)
  2. Source of message is who they claim to be (***source authentication***)

- Different approaches available:
  - Symmetric Key Encryption
  - Message Authentication Codes (MACs)
  - Hash Functions
  - Public Key Encryption (i.e. Digital Signatures)

Contents

Symmetric Encryption for Authentication



- **Confidentiality**: only B (and A) can recover plaintext
- **Source Authentication**: A is only other user with key; must have come from A
- **Data Authentication**: successfully decrypted; data has not been modified
- Assumption: decryptor can *recognise correct plaintext*

Credit: Figure 12.1(a) in Stallings, *Cryptography and Network Security*, 6th Ed.

Recognising Correct Plaintext

Example 1

B receives ciphertext (supposedly from A, using shared secret key K):

DPNFCTEJLYONCJAEZRCLASJTDQFY

B decrypts with key K to obtain plaintext:

SECURITYANDCRYPTOGRAPHYISFUN

- Was the plaintext encrypted with key K (and hence sent by A)?
- Is the ciphertext received the same as the ciphertext sent by A?

Recognising Correct Plaintext

Example 2

B receives ciphertext (supposedly from A, using shared secret key K):

QEFPFPQEBTOLKDJBPPXDBPLOOVX

B decrypts with key K to obtain plaintext:

FTUEUEFTQIDAZSYQEEMSQEADDKM

- Was the plaintext encrypted with key K (and hence sent by A)?
- Is the ciphertext received the same as the ciphertext sent by A?

Recognising Correct Plaintext

Example 3

B receives ciphertext (supposedly from A, using shared secret key K):

01101001101011010101101111000010

B decrypts with key K to obtain plaintext:

01011101000011010010101001011110

- Was the plaintext encrypted with key K (and hence sent by A)?
- Is the ciphertext received the same as the ciphertext sent by A?

Recognising Correct Plaintext

Example 1

- Assume the message is English
- Plaintext had expected structure; assume the plaintext is correct
  - Sent by A and has not been modified

Example 2

- Assume the message is English
- Plaintext had no structure in expected language; assume plaintext is incorrect
  - Either not sent by A or modified

Example 3

- Binary data, e.g. image, compressed file
- Cannot know whether correct or incorrect

Recognising Correct Plaintext

- Valid plaintexts should be small subset of all possible messages
  - E.g. 26^n possible messages of length n; only small subset are valid English phrases
- Plaintext messages have structure
- BUT automatically detecting structure can be difficult
- Add structure to make it easier, e.g.
  - Error detecting code or Frame Check Sequence
  - Packet header

Contents

MESSAGE AUTHENTICATION

A **message digest** does not authenticate the sender of the message.

To provide message authentication,

Alice needs to provide proof that

it is Alice sending the message and not an impostor.

The digest created by a cryptographic hash function

is normally called

a modification detection code (MDC).

What we need for message authentication is

a message authentication code (MAC).

Modification Detection Code (MDC)

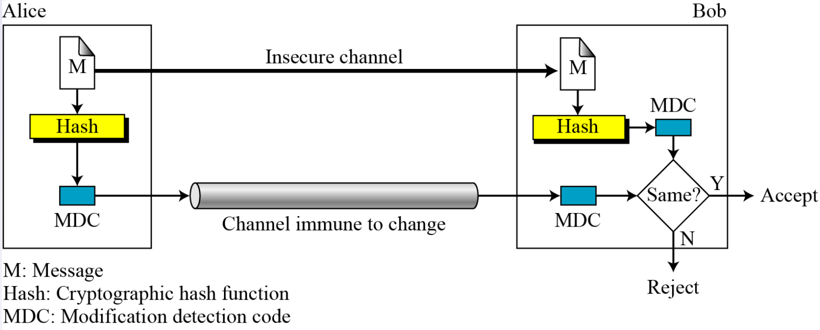
A modification detection code (MDC) is a message digest that can prove the **integrity** of the message: that message has not been changed.

If Alice needs to send a message to Bob and be sure that the message will not change during transmission, Alice can create a message digest, MDC, and send both the message and the MDC to Bob.

Bob can create a new MDC from the message and compare the received MDC and the new MDC.

If they are the same, the message has not been changed.

Modification Detection Code (MDC)

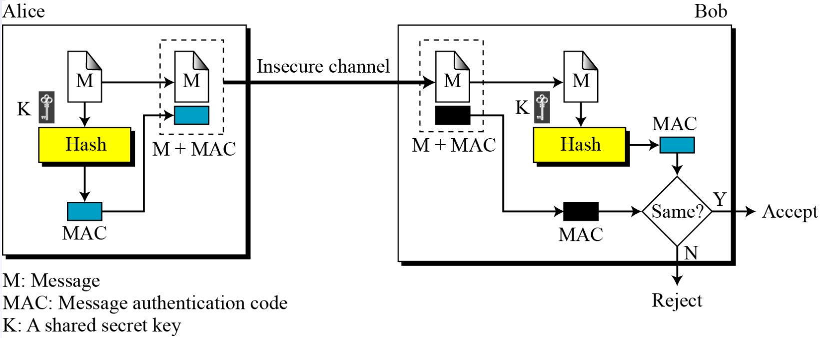


Credit: Figure 11.9 in *Cryptography and Network Security* ***Modification detection code (MDC)***

Message Authentication Code (MAC)

Need to ensure the integrity of the message and the data origin authentication.

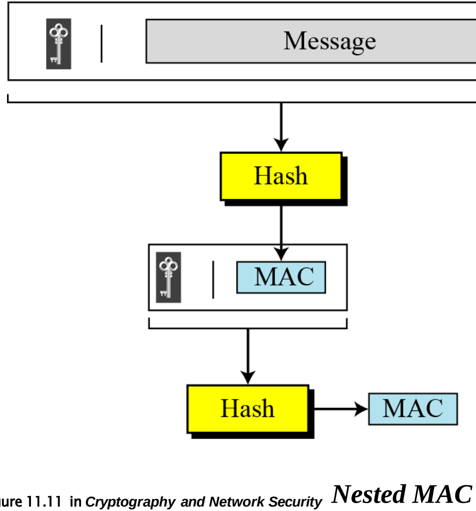
The difference between a MDC and a MAC is that MAC includes a secret between Alice and Bob.



Credit: Figure 11.10 in *Cryptography and Network Security* ***Message authentication code***

Nested MAC

To improve the security of a MAC.



Credit: Figure 11.11 in *Cryptography and Network Security*

Message Authentication Code (MAC)

Prefix MAC
(the key appended to the beginning of the message)

Postfix MAC
(the key appended to the end of the message)

We can combine the prefix and postfix MAC,
with the same key or two different keys.

Authentication with Message Authentication Codes

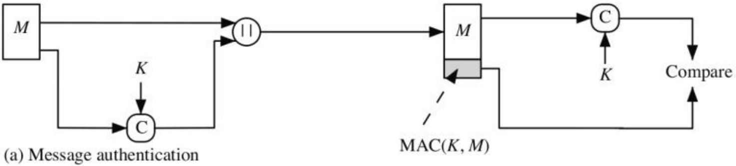
- Append **small, fixed-size** block of data to message: cryptographic checksum or MAC

$$T = MAC(K, M)$$

M = input message
MAC = MAC function
K = shared secret key of k bits
T = message authentication code (or tag) of n bits

- MAC function also called ***keyed hash function***
- MAC function similar to encryption, but does not need to be reversible
  - Easier to design stronger MAC functions than encryption functions

Example Uses of MAC

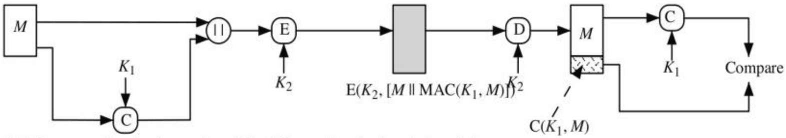


(a) Message authentication

- ***MAC of different messages with same key produces different tags.***
- ***MAC of same message with different keys produces different tags.***

Credit: Figure 12.4 in Stallings, *Cryptography and Network Security*, 6th Ed.

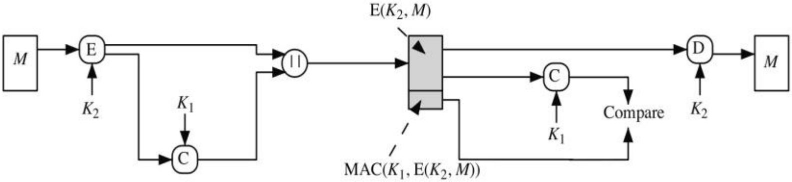
Example Uses of MAC



Message authentication & Confidentiality;
Authentication tied to **plaintext**

Credit: Figure 12.4 in Stallings, *Cryptography and Network Security*, 6th Ed.

Example Uses of MAC



Message authentication & Confidentiality;
Authentication tied to **ciphertext**

Credit: Figure 12.4 in Stallings, *Cryptography and Network Security*, 6th Ed.

Contents

Security of a MAC

Three possible cases of Eve forging a message:

1. Exhaustive search of key.
2. The preimage attack: Eve can find the key from the intercepted MAC = h(K | M).

(* Thus Eve can successfully replace the message with a forged one. *)

3. Given some pairs of messages and their MACs, Eve can manipulate them to come up with a new message and its MAC.

Note that the security of a MAC depends on the security of the underlying hash algorithm.

Requirement of MACs

Objective of Attacker

- Assume MAC function is known, key K is not
- For valid MAC code for given message x

Requirement of MAC Function

Computation Resistance: given one or more text-MAC pairs $[x_i, MAC(K, x_i)]$, computationally infeasible to compute any text-MAC pair $[x, MAC(K, x)]$ for new input x

Security of MACs

Brute Force Attack on **Key**

- Attacker knows $[x_1, T_1]$ where $T_1 = MAC(K, x_1)$
- Key size of k bits: brute force on key, 2^k
- But . . .many tags match T_1

MAC function is a **many-to-one** function!

- For keys that produce tag T_1, try again with $[x_2, T_2]$
- Effort to find K is approximately 2^k

Brute Force Attack on **MAC value**

- For x_m, find T_m without knowing K
- Similar effort required as one-way/weak collision resistant property for hash functions
- For n bit MAC value length, effort is 2^n

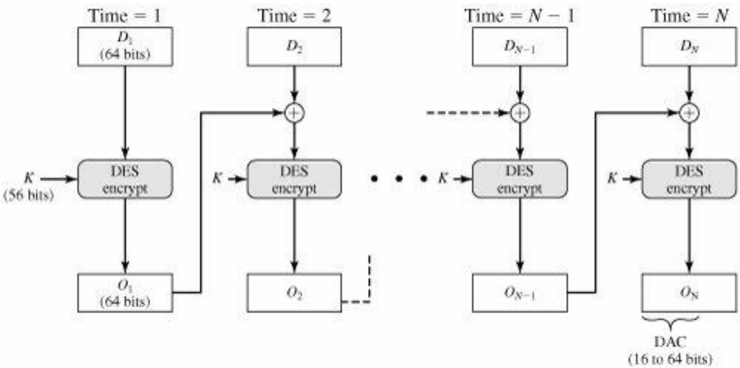
Effort to break MAC: $\min(2^k, 2^n)$

Contents

MACs Based on Block Ciphers

- **Data Authentication Algorithm (DAA)**: based on DES; considered insecure
- **Cipher-Based Message Authentication Code (CMAC)**: mode of operation used with Triple-DES and AES
- OMAC, PMAC, UMAC, VMAC, . . .

Data Authentication Algorithm (DAA)



$O_1 = E(K, D_1)$

$O_2 = E(K, [D_2 \oplus O_1])$

$O_3 = E(K, [D_3 \oplus O_2])$

.

.

$O_N = E(K, [D_N \oplus O_{N-1}])$

Data Authentication Code

Credit: Figure 12.7 in Stallings, *Cryptography and Network Security*, 6th Ed.

Cipher-Based Message Authentication Code (CMAC)

NIST has defined a standard (FIPS113) called
Data Authentication Algorithm, or CMAC, or CBCMAC.

The method is similar to CBC mode.

However, the idea is to create one block of MAC from N blocks of plaintext,

not to create N blocks of ciphertext from N blocks of plaintext.

The message is divided into N blocks, each m bits long.

If the last block is not m bits, it is padded with a 1-bit followed by enough 0-bits to make it m bits.

Cipher-Based Message Authentication Code (CMAC)

The size of CMAC is n bits.

The n leftmost bit from the last block is the CMAC.

In addition to the symmetric key, K,

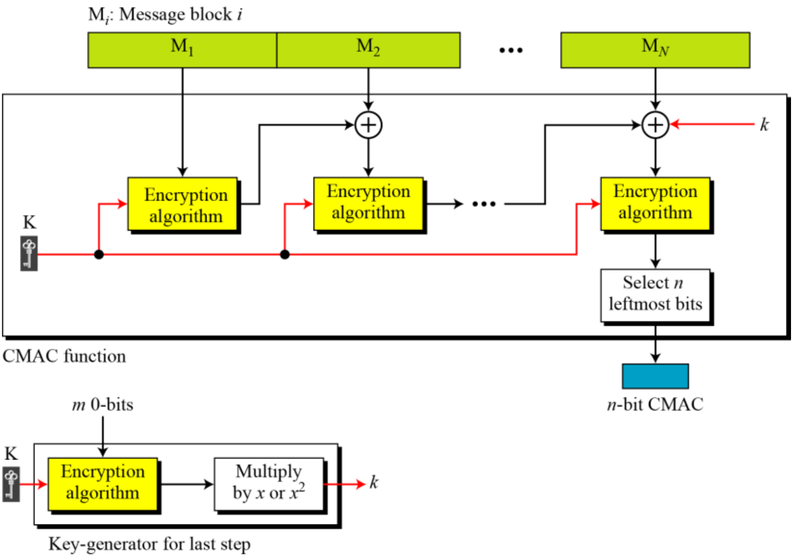
CMAC also uses another key, k, which applied only at the last step.

The result from the Encryption algorithm is multiplied by x if no padding is applied;

and is multiplied by x^2 if padding is applied.

The multiplications is in GF(2^m) with irreducible polynomial of degree m selected by the particular protocol used.

CMAC



Credit: Figure 11.13 in *Cryptography and Network Security* ***CMAC***

MACs Based on Hash Functions: HMAC

- NIST has issued a standard (FIPS198) for a nested MAC that is referred to as HMAC (hashed MAC).
- MAC function derived from cryptographic hash functions
  - MD5/SHA are fast in software (compared to block ciphers)
  - Libraries for hash functions widely available

  $$\mathrm{HMAC}(K, M) = H((K \oplus opad)||H((K \oplus ipad)||M))$$

  where ipad= 00110110 repeated, opad= 01011100 repeated
- Security of HMAC depends on security of hash function used

HMAC

1. The message is divided into *N* blocks, each of *b* bits.
2. The secret key is left-padded with 0's to create a *b*-bit key. Note that it is recommended that secret key (before padding) be longer than *n* bits, where *n* is the size of the HMAC.
3. The result of step 2 is XORed with a constant called ipad (input pad) to create a *b*-bit block. The value of ipad is the *b*/8 repetition of the sequence 00110110 $((36)_{16})$.
4. The resulting block is prepended to *N*-block message.
5. The result of step 4 is hashed to create an *n*-bit digest. We call the digest the *intermediate* HMAC.

HMAC

6. The intermediate n-bit HMAC is left padded with 0s to make a b-bit block.

7. Step 2 and 3 are repeated by a different constant opad (output pad). The value of opad is the $b/8$ repetition of the sequence 01011100 ($(5C)_{16}$).

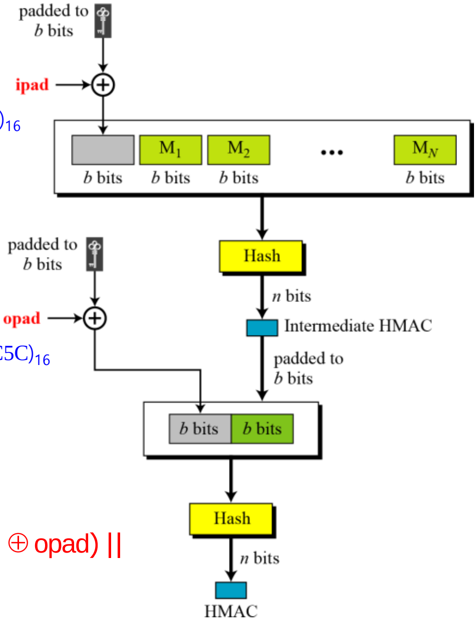
8. The result of step 7 is prepended to the block of step 6.

9. The result of step 8 is hashed with the same hashing algorithm to create the final n-bit HMAC.

HMAC

$(36363636363636363636)_{16}$

$(5C5C5C5C5C5C5C5C5C5C)_{16}$



$$HMAC(K, M) = H[(K \oplus opad) \,||\, H[(K \oplus ipad) \,||\, M]]$$

Credit: Figure 11.12 in *Cryptography and Network Security* ***HMAC***