Message Authentication (消息认证)

Sheng Zhong Yuan Zhang

Computer Science and Technology Department
Nanjing University

Outline

1. The Need for Message Integrity
2. Message Authentication Code
   - Definition of MAC
   - Correctness and security requirements of MAC
   - How to use MAC
3. Constructing Secure MACs
   - Constructing a fixed-length MAC
   - CBC-MAC for handling long messages
4. Authenticated Encryption
   - Definition of AE
   - CCA security
   - Unforageable encryption
   - Constructing AE

Questions about message integrity

To enforce secure communication is more than protecting message secrecy, we may also encounter the following message integrity/authenticity questions:

- How can you be sure the message (e.g. emails, SMS, etc.) is from the sender it claims to be from?
- How can you be sure the message's content is intact?

图 1: "Boss" sends you a message[1]

[1] Pic from Internet

Encryption cannot guarantee integrity

例 (An "encrypt-to-authenticate" scheme)

Consider the setting where Alice and Bob share a secret key/pad k.

1. To authenticate her message, Alice encrypts its message m by $c := m \oplus k$ and sends it to Bob.
2. To verify the message's authenticity, Bob decrypts to get a message m' and checks whether it "looks good" (e.g. whether it's properly formatted and meaningful).

- The adversary Eve can flip any bits of the message. If Eve knows the message format, she can modify the message without causing any suspicions of Bob.
- For message integrity authentication, encryption DOES NOT directly solve our problem.

Attach a tag to authenticate

In our daily lives, tags are often used to authenticate a variety of objects:

- Brand tags for merchandises (e.g. clothes, shoes, bags, foods, etc.)
- Name tags for animals (e.g. cats, dogs, etc.)
- Stamps for documents (e.g. letters, certificates, etc.)
- ……

图 2: The authentication tag of Yangcheng Lake Crab

Use a "tag" to authenticate messages

Q: Can we apply the "attach-a-tag" idea to solve our problem?
A: Yes, we can use a special tag called Message Authentication Code (**消息认证码**) or MAC.

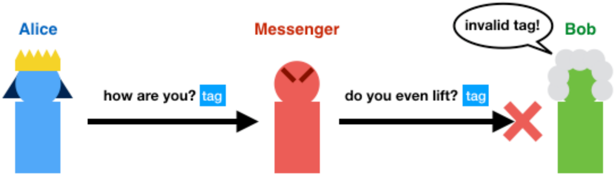

图 3: Use a MAC tag to authenticate messages[2]

[2]Pic from `https://livebook.manning.com/book/real-world-cryptography/`

Formal definition of MAC

DEFINITION 4.1

A **message authentication code** (or **MAC**) Π consists of three probabilistic polynomial-time algorithms (Gen, Mac, $Vrfy$):

1. The **key-generation algorithm** Gen takes as input the security parameter 1^n and outputs a key k with $|k| \geq n$.
2. The **tag-generation algorithm** Mac takes as input a key k and a message $m \in \{0,1\}^*$, and outputs a tag t:
$$t \leftarrow Mac_k(m) \text{ or } t := Mac_k(m).$$
3. The deterministic **verification algorithm** $Vrfy$ takes as input a key k and a message m and a tag t, it outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid:
$$b := Vrfy_k(m, t).$$

Correctness of MAC

- **Correctness**: it is required that for every n, every key k output by $Gen(1^n)$, and every $m \in \{0,1\}^*$, it holds that

$$Vrfy_k(m, Mac_k(m)) = 1.$$

- When Mac is deterministic, a **canonical way** to perform verification, which is actually often used, is to simply recompute the tag and check for equality.

The message authentication experiment

The **security** is defined via the following experiment for a message authentication scheme $\Pi = (Gen, Mac, Vrfy)$:

The message authentication experiment $\textit{Mac-forge}_{\mathcal{A},\Pi}(n)$:

1. A key k is generated by running $Gen(1^n)$ and kept secret from the adversary.
2. The adversary $\mathcal{A}$ is given input 1^n and oracle access to $Mac_k(\cdot)$. The adversary eventually outputs (m, t). Let $\mathcal{Q}$ denote the set of queries that $\mathcal{A}$ asked its oracle.
3. $\mathcal{A}$ succeeds iff (1)$Vrfy_k(m, t) = 1$ and (2)$m \notin \mathcal{Q}$. In that case, the output of the experiment is defined to be 1 and written as:

$$\textit{Mac-forge}_{\mathcal{A},\Pi}(n) = 1.$$

Definition of a secure MAC

DEFINITION 4.2

A message authentication code $\Pi = (Gen, Mac, Vrfy)$ is **existentially unforgeable under an adaptive chosen-message attack** or just **secure**, if for all PPT adversaries $\mathcal{A}$, there is a negligible function *negl* such that:

$$Pr[\textit{Mac-forge}_{\mathcal{A},\Pi}(n) = 1] \leq negl(n).$$

- A secure MAC ensures that an adversary cannot generate a valid tag on any new message that was never previously authenticated.

Strongly secure MAC

- Recall a secure MAC ensures that an adversary cannot generate a valid tag on a new message that was never previously authenticated.
- Nevertheless, a secure MAC does not ensure that the adversary cannot generate a new tag on a previously authenticated message.
- A strongly secure MAC can rule out the above "weakness"[3].

[3] In general, this type of "weakness" is not a concern. Nevertheless, in some settings it is useful to consider a stronger security.

Definition of strong MACs

We define a similar experiment called *Mac-sforge*:

The message authentication experiment $\textit{Mac-sforge}_{\mathcal{A},\Pi}(n)$:

1. A key k is generated by running $Gen(1^n)$.
2. The adversary $\mathcal{A}$ is given input 1^n and oracle access to $Mac_k(\cdot)$. The adversary eventually outputs (m, t). Let $\mathcal{Q}$ denote the set of pairs of queries that $\mathcal{A}$ asked its oracle and their corresponding responses.
3. $\mathcal{A}$ succeeds iff (1)$Vrfy_k(m, t) = 1$ and (2)$(m, t) \notin \mathcal{Q}$. In that case, the output of the experiment is defined to be 1 and written as:

$$\textit{Mac-sforge}_{\mathcal{A},\Pi}(n) = 1.$$

Definition of strong MACs

DEFINITION 4.3

A message authentication code $\Pi = (\mathit{Gen}, \mathit{Mac}, \mathit{Vrfy})$ is **strongly secure**, or **a strong MAC**, if for all PPT adversaries $\mathcal{A}$, there is a negligible function *negl* such that:

$$\mathit{Pr}[\mathit{Mac\text{-}sforge}_{\mathcal{A},\Pi}(n) = 1] \leq \mathit{negl}(n).$$

PROPOSITION 4.4

Let $\Pi = (\mathit{Gen}, \mathit{Mac}, \mathit{Vrfy})$ is a secure MAC that uses canonical verification. Then Π is a strong MAC.

- In fact, all real-world MACs use canonical verification.

How to use a secure MAC?

- For each message m, Sender sends $<m, t = MAC_k(m)>$ to Receiver.
- Receiver runs $Vrfy(m, t)$. If verification passes, accepts m and adds m to $\mathcal{Q}$. Otherwise, reject m.

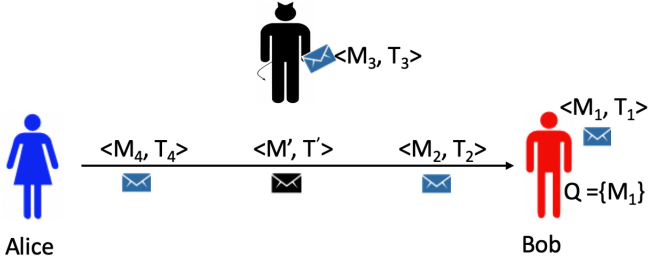

图 4: Alice sends M1, M2, M3, M4 one by one to Bob. The attacker intercepts M_2 and replace it with M'.

How to handle repeated messages with a secure MAC?

What if Alice needs to send a message in $\mathcal{Q}$, (i.e. was sent previously)?

- Directly applying the MAC would cause repeated messages to be rejected.
- One can append a unique **timestamp** or **nonce** to each message.

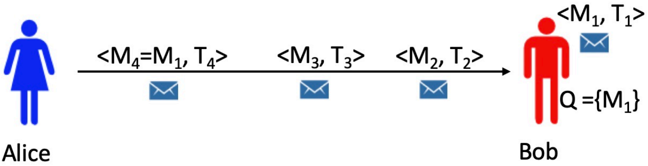

图 5: M_4 would be rejected by Bob when directly applying MAC

Constructing a fixed-length MAC with PRFs

- A fixed-length MAC: If there is a function l such that for every k output by $Gen(1^n)$, Mac_k is ONLY defined for message $m \in \{0,1\}^{l(n)}$, we call the scheme a *fixed-length MAC for messages of length* $l(n)$.
- Basic idea: If the MAC tag t is obtained by applying a PRF to a message m, then forging a tag on a previously unauthenticated message requires the adversary to correctly guess the output of the PRF at a "new" input. However, this is infeasible for any PPT adversaries.

Construction details

CONSTRUCTION 4.5

Let F be a PRF. Define a fixed-length MAC for message of length n as follows:

- *Mac*: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, output the tag: $t := F_k(m)$.
- *Vrfy*: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^n$, and a tag $t \in \{0,1\}^n$, output 1 iff $t = F_k(m)$.

Security analysis of Construction 4.5

THEOREM 4.6

If F is a PRF, then Construction 4.5 is a secure fixed-length MAC for message of length n.

Proof Sketch:

First, we construct a new ideal scheme $\tilde{\Pi}$ by replacing the PRF in Construction 4.5 with a real random function. We know

$$Pr[\textit{Mac-forge}_{\mathcal{A},\tilde{\Pi}}(n) = 1] \leq 2^{-n}. \quad (1)$$

Q: Why?

Proof Sketch (Contd.):
Next, due to the definition of PRF, we know the *Mac-forge* experiment cannot differentiate Π (which is built with a PRF) and $\tilde{\Pi}$ (which is built with a real random function), i.e.

$$|Pr[\textit{Mac-forge}_{\mathcal{A},\Pi}(n) = 1] - Pr[\textit{Mac-forge}_{\mathcal{A},\tilde{\Pi}}(n) = 1]| \leq negl(n). \quad (2)$$

Q: Why?

Summarizing (1) and (2), we know

$$Pr[\textit{Mac-forge}_{\mathcal{A},\Pi}(n) = 1] \leq 2^{-n} + negl(n). \quad (3)$$

□

The domain extension problem

- Message length needs to be equal to the underlying PRF's input length in Construction 4.5.
- Existing practical PRFs (i.e. block ciphers) take short, fixed-length inputs. (e.g. AES takes 128-bit input only.)

Q: Given a fixed-length MAC_k, how to extend it to handle longer messages?

Trial 1 is NOT secure

Given a MAC_k of a fixed-length n,

- Break the long message M into multiple short blocks $m_1, \ldots, m_d$ of fixed length n.
- **Authenticate each block separately**, i.e. compute $t_i = MAC_k(m_i)$ for all i, and output $< t_1, \ldots, t_d >$ as the tag.

Q: Is this secure? Why?
A: No, an attacker can launch a "block reordering attack" e.g. generate a new message-tag pair $< m_2||m_1, t_2||t_1 >$ after seeing a correct pair $< m_1||m_2, t_1||t_2 >$.

Trial 2 is NOT secure

Given a MAC_k of a fixed-length n,

- Break the long message M into multiple short blocks $m_1, \ldots, m_d$ of fixed length.
- **Authenticate each block along with its index separately**, i.e. compute $t_i = MAC_k(i||m_i)$ for all i, and output $< t_1, \ldots, t_d >$ as the tag.

Q: Is this secure? Why?
A: No, an attacker can launch a "truncation attack", e.g. generate a new message-tag pair $< m_1, t_1 >$ after seeing a correct pair $< m_1||m_2, t_1||t_2 >$.

Trial 3: Adding message length

Given a MAC_k of a fixed-length n,

- Break the long message M into multiple short blocks $m_1, \dots, m_d$ of fixed length n.
- **Authenticate each block along with its index and the message length separately**, i.e. compute $t_i = MAC_k(d||i||m_i)$ for all i, and output $< t_1, \dots, t_d >$ as the tag.

Q: Is this secure? Why?

A: No, an attacker can launch a "mix-and-match attack", e.g. generate a new message-tag pair $< m_1||m_2', t_1||t_2' >$ after seeing two correct pairs $< m_1||m_2, t_1||t_2 >$ and $< m_1'||m_2', t_1'||t_2' >$.

CBC-MAC for domain extension

- Trial 3 can be fixed by further including a unique "message identifier", however, it still has a big drawback of "long tag", i.e. the tag is as long as the message.
- In practice, a standardized MAC that is used widely is **CBC-MAC**.

The basic CBC-MAC

When authenticating messages of any fixed length, one can use:

Construction 4.11: Basic CBC-MAC for fixed-length messages

Let F be a pseudorandom function, and fix a length function $l > 0$:

- *Mac*: on input a key $k \in \{0,1\}^n$ and a message of length $l(n) \cdot n$,
  1. Parse $m = m_1, ..., m_l$ where each m_i is of length n.
  2. Set $t_0 := 0^n$. Then, for $i = 1$ to l:
     Set $t_i := F_k(t_{i-1} \oplus m_i)$.

  Output t_l as the tag.
- *Vrfy*: on input a key $k \in \{0,1\}^n$, a message m, and a tag t, do: If m is not of length $l(n) \cdot n$ then output 0. Otherwise, output 1 if.f. $t = Mac_k(m)$.

The security of basic CBC-MAC

Regarding the security, we have

THEOREM 4.12

Let l be a polynomial. If F is a pseudorandom function, then Construction 4.11 is a secure MAC for messages of length $l(n) \cdot n$.

- We caution that the basic CBC-MAC is NOT secure in the general case when messages of different lengths may be authenticated. In such case, a variant of CBC-MAC can be used.
- Read the proof and also the variant in textbook if you are interested.

Despite the similarities, they have important differences:

- CBC-mode encryption uses a random IV (**crucial for security**); CBC-MAC uses NO IV or a fixed zero-IV (also **crucial for security**).
- CBC-mode encryption output all blocks; CBC-MAC only output the final block (**crucial for security**)

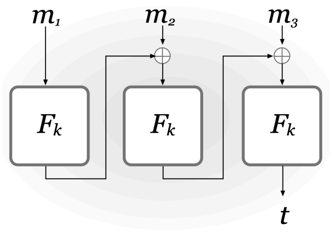

图 6: The basic CBC-MAC

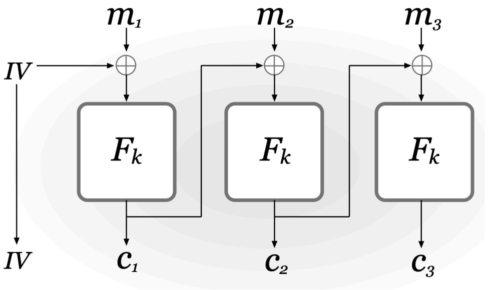

图 7: CBC mode of operation

Achieving secrecy and integrity simultaneously

- We have shown how to obtain secrecy in private-key setting using encryption.
- We have just shown how to obtain integrity using MAC.

Q: How can we achieve both goals simultaneously?

A: We resort to Authenticated Encryption (AE).

Definition of AE

To achieve an “ideally secure” communication, we require AE can simultaneously protect:

- data secrecy: Specifically, we require AE to be CCA-secure.
- data integrity: Specifically, we require AE to be unforgeable.

DEFINITION 4.17

A private-key encryption is an **authenticated encryption** scheme if it is CCA-secure and unforgeable.

The CCA indistinguishability experiment

To formalize our secrecy goal, we first define the following experiment:

The CCA indistinguishability experiment $PrivK^{cca}_{\mathcal{A},\Pi}(n)$:

1. A key k is generated by running $Gen(1^n)$.
2. Adversary $\mathcal{A}$ is given input 1^n and oracle access to $Enc_k(\cdot)$ and $Dec_k(\cdot)$. It outputs a pair of messages m_0, m_1 of the same length.
3. A uniform bit $b \in \{0, 1\}$ is chosen, and then a ciphertext $c \leftarrow Enc_k(m_b)$ is computed and given to $\mathcal{A}$. We call c the challenge ciphertext.
4. The adversary $\mathcal{A}$ continues to have oracle access to $Enc_k(\cdot)$ and $Dec_k(\cdot)$, but is not allowed to query the latter on the challenge ciphertext itself. Eventually, $\mathcal{A}$ outputs a bit b'.
5. The output of the experiment is defined to 1 if $b' = b$, and 0 otherwise. If the output of the experiment is 1, we say that $\mathcal{A}$ **succeeds**.

Definition of CCA-secure encryption

DEFINITION 3.33

A private-key encryption scheme Π has **indistinguishable encryption under a chosen-ciphertext attack**, or is **CCA-secure**, if for all PPT adversaries $\mathcal{A}$ there is a negligible function *negl* such that:

$$Pr[PrivK^{cca}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n), \tag{4}$$

where the probability is taken over all randomness used in the experiment.

An example of non-CCA-secure encryption

- Is the following construction CCA-secure? No!

Construction 3.30: A CPA-secure scheme from any pseudo-random function

Let F be a pseudorandom function. Define a private-key encryption scheme for messages of length n as follows:

- *Gen*: on input 1^n, choose uniform $k \in \{0,1\}^n$ and output it.
- *Enc*: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, choose uniform $r \in \{0,1\}^n$ and outputs the ciphertext

$$c := < r, F_k(r) \oplus m > .$$

- *Dec*: on input a key $k \in \{0,1\}^n$ and a ciphertext $c = < r, s >$, output the plaintext message

$$m := F_k(r) \oplus s$$

CCA-secureness implies “non-malleablility”

- An encryption algorithm is **malleable** means one can generate a new ciphertext based on a ciphertext of m without knowing m, and the new ciphertext' corresponding plaintext is $f(m)$ for a known function f.
- Example: Given $c =< c_1 = r, c_2 = F_k(r) \oplus m >$, adversary can construct
$$c' =< c_1, c_2 \oplus a >$$
which decrypts to $m \oplus a$.
- A CCA-adversary can query c''s decryption, gets $m \oplus a$ and m.
- CCA-secureness implies “non-malleablility”(“不可延展性”或“不可锻造性”).

The unforgeable encryption experiment

To formalize our integrity goal, we define the following experiment:

The unforgeable encryption experiment $\mathit{Enc\text{-}Forge}_{\mathcal{A},\Pi}(n)$:

1. Run $Gen(1^n)$ to obtain a key k.
2. The adversary $\mathcal{A}$ is given input 1^n and access to an encryption oracle $Enc_k(\cdot)$. The adversary outputs a ciphertext c.
3. Let $m := Dec_k(c)$, and let $\mathcal{Q}$ denote the set of all queries that $\mathcal{A}$ asked its encryption oracle. The output of the experiment is 1 iff (1) $m \neq \perp$ (i.e. decryption fails.) and (2) $m \notin \mathcal{Q}$ (i.e. m has not been queried.).

Definition of unforgeable encryption

DEFINITION 4.16

A private-key encryption scheme Π is unforgeable if for all PPT adversaries $\mathcal{A}$, there is a negligible function *negl* such that:

$$Pr[\textit{Enc-Forge}_{\mathcal{A},\Pi}(n) = 1] \leq \textit{negl}(n).$$

Constructing AE via combining secure encryption scheme and AE

Our basic idea is to combine a CPA-secure encryption scheme and a MAC scheme to construct an AE. We have three different approaches to perform the combining:

- Encrypt-and-authenticate.
- Authenticate-then-encrypt
- Encrypt-then-authenticate

Encrypt and authenticate

Let $\Pi_E = (Enc, Dec)$ be an arbitrary CPA-secure encryption scheme and let $\Pi_M = (Mac, Vrfy)$ denote an arbitrary (strongly) secure MAC scheme, where key generation in both schemes simply involves choosing a uniform n-bit key.

Encrypt-and-authenticate:

1. The sender and the receiver agree on independent secret keys: $k_E \xleftarrow{\$} \{0,1\}^n$, $k_M \xleftarrow{\$} \{0,1\}^n$.
2. For message m, sender computes

$$c \leftarrow Enc_{k_E}(m) \text{ and } t \leftarrow Mac_{k_M}(m),$$

   and sends (c, t) to receiver.
3. Receiver decrypts c to recover m; assuming no error occurred, it then verifies the tag t. If $Vrfy(m, t) = 1$, receiver outputs m; otherwise, it outputs an error.

Possible weakness of Encrypt-and-authenticate

Weakness of AEs constructed using this approach?

- Message secrecy is NOT necessarily protected (as a secure MAC does NOT guarantee any secrecy.).
- Specifically, the encrypt-and-authenticate scheme is likely to be insecure against chosen-plaintext attack (since most MAC used in practice are deterministic.).

Note: the above analysis assumes a very general setting (where an arbitrary CPA-secure encryption and an arbitrary secure MAC are adopted). It does not mean the encrypt-and-authenticate cannot protect message secrecy under all circumstances.

Authenticate then encrypt

Let $\Pi_E = (Enc, Dec)$ be an arbitrary CPA-secure encryption scheme and let $\Pi_M = (Mac, Vrfy)$ denote an arbitrary (strongly) secure MAC scheme, where key generation in both schemes simply involves choosing a uniform n-bit key.

Authenticate-then-encrypt:

1. The sender and the receiver agree on independent secret keys: $k_E \xleftarrow{\$} \{0,1\}^n$, $k_M \xleftarrow{\$} \{0,1\}^n$.
2. For message m, sender computes
$$t \leftarrow Mac_{k_M}(m) \text{ and } c \leftarrow Enc_{k_E}(m||t),$$
and sends c to receiver.
3. Receiver decrypts c to recover $m||t$; assuming no error occurred, it then verifies the tag t. If $Vrfy(m,t) = 1$, receiver outputs m; otherwise, it outputs an error $\perp$.

Possible weakness of Authenticate-then-encrypt

- Ciphertext integrity is NOT protected.
- Thus, the adversary may be able to manipulate the ciphertext. In addition, the receiver has to first decrypt and then know whether the message is authentic or not.
- The above facts have be utilized to launch attacks, e.g. Padding oracle attacks, POODLE attacks and Lucky 13 for historical SSL and TLS protocol.
- Authenticate-then-encrypt does not provide AE in general, and should NOT be used.

Encrypt then authenticate

Let $\Pi_E = (Enc, Dec)$ be an arbitrary CPA-secure encryption scheme and let $\Pi_M = (Mac, Vrfy)$ denote an arbitrary (strongly) secure MAC scheme. Define a private-key encryption scheme (Gen', Enc', Dec') as follows.

CONSTRUCTION 4.18 (Encrypt-then-authenticate):

1. Gen': on input 1^n, $k_E \xleftarrow{\$} \{0,1\}^n$, $k_M \xleftarrow{\$} \{0,1\}^n$.
2. Enc': on input a key (k_E, k_M) and plaintext m, computes

   $$c \leftarrow Enc_{k_E}(m) \text{ and } t \leftarrow Mac_{k_M}(c).$$

   Output the ciphertext $< c, t >$.
3. Dec': on input a key (k_E, k_M) and ciphetext $< c, t >$, outputs $\perp$ if $Vrfy(c, t) \neq 1$, and otherwise, outputs $Dec_{k_E}(c)$.

Construction 4.18 is an authenticated encryption scheme

THEOREM 4.19

Let Π_E be a CPA-secure encryption scheme and let Π_M be a strongly secure MAC scheme. Then Construction 4.18 is an authenticated encryption scheme.

Why is it true?

- Strong security of Π_M directly implies Construction 4.18 is unforgeable.
- CCA-security of Construction 4.18 reduces to the CPA-security of Π_E.

"Among the three, Encrypt-then-Authenticate is the most ideal approach to construct an AE."