

Message Authentication Codes

Chanathip Namprempre

Computer Science
Reed College

Agenda: Message Authentication Codes

1. Motivation: why integrity protection
2. Encryption doesn't provide authenticity.
3. MAC schemes: syntax and security definitions
4. Block cipher based MAC schemes: π_1 , π_2 , CBC MAC, XCBC
5. Example attacks and secure MAC schemes
6. Hash-based MAC scheme

Motivation

Message Authentication

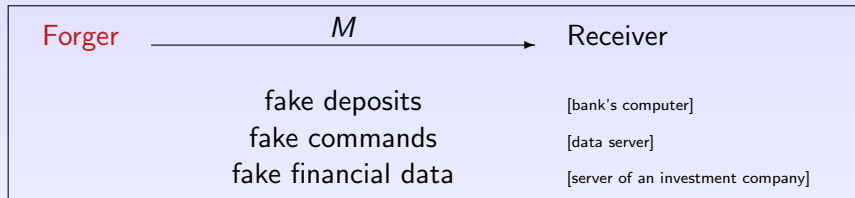
We often want to protect **integrity** of messages.

Private-key setting : MACs

Public-key setting : Digital Signatures

Motivation

Integrity is important in many applications.



Receiver thinks M comes from some legitimate sender, but in fact M comes from **Forger**.

Points to remember

- ▶ We do **not** assume that M is to be kept secret.
- ▶ F (forger) controls the channel, i.e. F can drop, inject, modify, repeat packets.

Encryption does NOT provide authenticity.

F wants to change deposit from 100 to 900.

Some may think “it is hard for F to change ciphertext so that it decrypts to 900 without knowing the key K .”

WRONG! This is easy!

Example : suppose encryption is OTP.

Example :One-time Pad

- ▶ Suppose $M = 0001$ [i.e. 1 in decimal].
- ▶ F wants to change M to $M' = 1001$ [i.e. 9 in decimal].
- ▶ Suppose $K = 1010$.
- ▶ So $C = M \oplus K = 1011$.

F : take $C = 1011$
let $\Delta = M \oplus M' = 0001 \oplus 1001 = 1000$
compute $C' = C \oplus \Delta = 1011 \oplus 1000 = 0011$

- ▶ When receiver decrypts, they get $0011 \oplus 1010 = 1001 = M'$!
- ▶ F did not need to know K .

False conclusions:

- ▶ “Don’t use OTP.”
OTP is for secrecy! It does its job, not something else. (Don’t use a car if you want to fly.)
- ▶ “Example is contrived.”
OTP is for real. CTR is pretty much OTP. CBC doesn’t work much better either.
- ▶ “Should add redundancy”
Adding pads won’t help here.

Correct conclusions

Encryption gives you privacy, not authenticity.

[In fact, with most encryption schemes, any ciphertext will decrypt to something.]

Bottom line:

Good cryptographic design is **goal-oriented**.

Must understand goal before designing schemes.

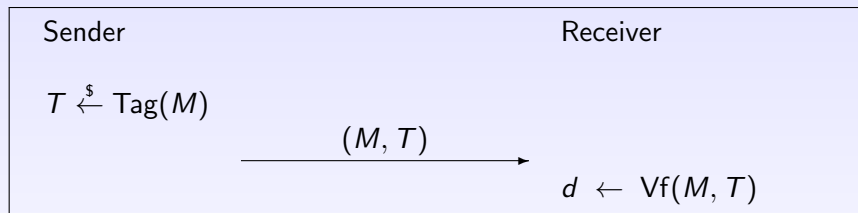
A good designer uses the right tool for the desired goal.

Message Authentication Codes

Usage

Syntax and security definitions

Usage



Syntax (MAC : scheme & tag)

Definition

A MAC scheme consists of 3 algorithms : $\pi = (\mathcal{K}, \text{Tag}, \text{Vf})$

\mathcal{K} : randomized key generation algorithm

$$K \xleftarrow{\$} \mathcal{K}$$

$[\text{Keys}(\pi) = \{K \mid K \text{ has non-zero probability of being output by } \mathcal{K}\}]$

Tag : MAC-generation algorithm
randomized or stateful

$$T \xleftarrow{\$} \text{Tag}_K(M)$$

$[M \in \{0, 1\}^*, K \in \text{Keys}(\pi), T \in \{0, 1\}^* \cup \{\perp\}]$

Syntax

Vf : MAC-verification algorithm
deterministic

$$d \leftarrow Vf_K(M, T)$$

$$[M \in \{0, 1\}^*, K \in Keys(\pi), T \in \{0, 1\}^*, d \in \{0, 1\}]$$

Correctness

$$\forall K \in Keys(\pi), M \in \{0, 1\}^*,$$

$$\Pr \left[T = \perp \text{ OR } Vf_K(M, T) = 1 : T \stackrel{\$}{\leftarrow} Tag_K(M) \right] = 1 .$$

Points to remember about syntax

- ▶ Just **syntax**. No security yet.
- ▶ Vf is **deterministic**. Hard to require stateful receiver with **consistent** states.
- ▶ MAC-generation could be **deterministic & stateless**. When this happens, $\text{Vf}_K(\cdot, \cdot)$ just recomputes Tag and compare, i.e.

$\text{Vf}_K(M, T)$

$T' \leftarrow \text{Tag}_K(M)$

If $(T = T' \text{ and } T' \neq \perp)$ then 1 else 0

So for deterministic MAC, we can say $\pi = (\mathcal{K}, \text{Tag})$.

Security definition

Issues

Want: hard for adversary F to forge valid tags.

- ▶ Want this for **any** messages, not just “meaningful” ones.
(Want security guarantee for all applications)
- ▶ Want **more than** hardness of **key recovery**.
(Maybe can forge without knowing key. That'd be bad.)
- ▶ Let F **see valid message-tag pairs** *before* forging?
(No-message attacks? Chosen-message attacks?)
- ▶ **Replay?** (F sees a (M, Tag) pair and repeats it.)
(Don't allow for now.)
- ▶ What if F can **forge many pairs** of valid (M, Tag) 's and is content if any of the pair is valid?
(Let F submits many verification queries & wins if at least one is valid)

Security definition (cont.)

- ▶ “signing” query
(# of queries = q_s ; # of bits = μ_s)
- ▶ verification query
(# of queries = q_v ; # of bits of M 's = μ_v)
- ▶ **F** win if $\text{Vf}_K(\cdot)$ ever returns 1 on a pair (M, Tag) not previously returned by $\text{Tag}_K(\cdot)$.

WUF-CMA

Subroutine *Initialize*

$K \xleftarrow{\$} \text{KG}; S \leftarrow \emptyset; \text{win} \leftarrow \text{false}$

Subroutine *Tag*(M)

$S \leftarrow S \cup \{M\}; \text{Return Tag}(K, M)$

Subroutine *Vf*(M, T)

$v \leftarrow \text{Vf}(K, M, T)$

If $v = 1$ and $M \notin S$ then $\text{win} \leftarrow \text{true}$

Return v

Subroutine *Finalize*

Return win

Experiment $\mathbf{Exp}_{\text{MA}}^{\text{wuf-cma}}(A)$

Initialize

$A_{\text{Tag}, \text{Vf}}$

Return *Finalize*

wuf-cma advantage

The **wuf-cma advantage** of an adversary A mounting a chosen-message attack against MA is

$$\mathbf{Adv}_{\text{MA}}^{\text{wuf-cma}}(A) = \Pr \left[\mathbf{Exp}_{\text{MA}}^{\text{wuf-cma}}(A) \Rightarrow \text{true} \right].$$

SUF-CMA

Subroutine *Initialize*

$K \xleftarrow{\$} \text{KG}; S \leftarrow \emptyset; \text{win} \leftarrow \text{false}$

Subroutine *Tag*(M)

$T \xleftarrow{\$} \text{Tag}(K, M); S \leftarrow S \cup \{(M, T)\}$

Return T

Subroutine *Vf*(M, T)

$v \leftarrow \text{Vf}(K, M, T)$

If $v = 1$ and $(M, T) \notin S$ then $\text{win} \leftarrow \text{true}$

Return v

Subroutine *Finalize*

Return win

Experiment $\text{Exp}_{\text{MA}}^{\text{suf-cma}}$

Initialize

$A_{\text{Tag}, \text{Vf}}$

Return *Finalize*

suf-cma advantage

The **suf-cma advantage** of an adversary A mounting a chosen-message attack against MA is

$$\text{Adv}_{\text{MA}}^{\text{suf-cma}}(A) = \Pr \left[\text{Exp}_{\text{MA}}^{\text{suf-cma}}(A) \Rightarrow \text{true} \right] .$$

Block cipher based MAC schemes

Example: MAC scheme π_0

MAC scheme π_0

Let $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^L$ be a family of functions.

$\text{Tag}_K(M)$

if $(|M| \neq n)$ then return \perp

Return $F_K(M)$

Theorem: PRF \implies MAC

If $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^L$ is a secure PRF and 2^L is large, then π_0 is a secure MAC.

For every efficient adversary A against π_0 , there exists an efficient adversary B against F such that

$$\mathbf{Adv}_{\pi_0}^{\text{uf-cma}}(A) \leq \mathbf{Adv}_F^{\text{prf}}(B) + \frac{1}{2^L}.$$

Example: MAC scheme π'_0

MAC scheme π'_0

Let $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^L$ and $F' : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^t$ be families of functions. Let F' be the same as F except the output is truncated to t bits.

$\text{Tag}_K(M)$
if $(|M| \neq n)$ then return \perp
Return $F'_K(M)$

Fact: Let t and L be integers and let $t < L$. Then, if $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^L$ is a secure PRF, then $F' : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^t$ is also a secure PRF.

So, if F is a secure PRF, then π'_0 is also a secure MAC.

MAC for large inputs?

What if the input messages are longer than one block?

Example: MAC scheme π_1

π_1

Let $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^L$ be a family of functions.

$\text{Tag}_K(M)$

if $(|M| \bmod n \neq 0 \text{ or } |M| = 0)$ then return \perp

Break M into n -bit blocks $M = M[1] \dots M[s]$

for $i = 1$ to s do $y_i \leftarrow F_K(M[i])$

$\text{Tag} \leftarrow y_1 \oplus \dots \oplus y_s$

Return Tag

Example : MAC scheme π_2

What if we modify π_1 to prevent the previous attack by enciphering the block number along with the data block?

π_2

Let m be an integer such that $1 \leq m \leq n - 1$.

Let $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^L$ be a family of functions.

$\text{Tag}_K(M)$

$t \leftarrow n - m$

if $(|M| \bmod t \neq 0 \text{ or } |M| = 0 \text{ or } |M|/t \geq 2^m)$ then return \perp

Break M into t -bit blocks $M = M[1] \dots M[s]$

for $i = 1$ to s do $y_i \leftarrow F_K([i]_m \| M[i])$

$\text{Tag} \leftarrow y_1 \oplus \dots \oplus y_s$

Return Tag

Secure?

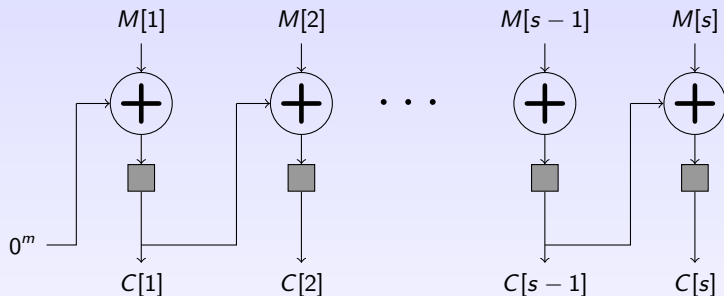
CBC MAC

CBC MAC is like CBC encryption but with $IV = 0^n$ and the tag is the last ciphertext block.

If fixed-length input, then **SECURE**.

Otherwise, **INSECURE!**

CBC MAC: pictorially



- ▶ The tagging algorithm for CBC MAC. The gray boxes denote the permutation E_K where E is the underlying block cipher and K is the shared secret key.
- ▶ It is assumed here that the message length is a *fixed* multiple of the block size.

ECBC MAC

- ▶ CBC MAC is vulnerable to a **length-extension attack**.
- ▶ How to fix CBC MAC for variable-length messages?
 - ▶ Encipher the last block of output with another key.
 - ▶ The additional encipherment prevents the length-extension attack.
 - ▶ This solution is called **ECBC MAC**.

Let $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^L$ be a block cipher.

Theorem: ECBC MAC security

For every efficient q -query PRF adversary A attacking ECBC, there exists an efficient adversary B attacking F such that

$$\mathbf{Adv}_{ECBC}^{\text{uf-cma}}(A) \leq \mathbf{Adv}_F^{\text{prp-cpa}}(B) + \frac{2q^2}{2^n}.$$

ECBC MAC security: interpretation

$$\mathbf{Adv}_{ECBC}^{\text{uf-cma}}(A) \leq \mathbf{Adv}_F^{\text{prp-cpa}}(B) + \frac{2q^2}{2^n}.$$

Suppose q is the number of message MACed with a secret key.

- ▶ ECBC MAC is secure as long as $q \ll 2^{n/2}$
- ▶ Suppose we want $\mathbf{Adv}_{ECBC}^{\text{uf-cma}}(A) \leq 1/2^{32}$
 - ▶ Then, $q^2/2^n < 1/2^{32}$
 - ▶ With AES, $n = 128$. So, $q \leq 2^{48}$. Acceptable.
 - ▶ With 3DES, $n = 64$. So, $q \leq 2^{16}$. Unacceptable.
- ▶ Once $q = 2^{n/2}$, we can attack ECBC using birthday attack + length-extension.
 - ▶ Find $M_1 \neq M_2$ that get MACed to the same tag t , then query for tag of $M_1 \| w$, and forge with $(M_2 \| w, t)$.

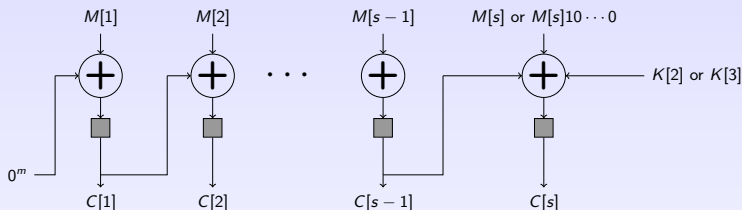
Padding input messages

What if input messages aren't of length multiple of n ?

- ▶ pad with 0's?
- ▶ pad with 100...0?
- ▶ pad with 100...0 with dummy block?
- ▶ CMAC (like CBC MAC) but
 - ▶ pad with 100...0 if necessary and don't pad if unnecessary
 - ▶ then xor the last block with another key K_1 if there's a pad or with K_2 if there's no pad.
 - ▶ No dummy block.
 - ▶ No additional encipherment like ECBC MAC.

XCBC MAC

XCBC MAC is like CBC MAC but uses two extra keys so that it can deal with variable-length input messages.



- ▶ The tagging algorithm for CBC MAC. The gray boxes denote the permutation E_K where E is the underlying block cipher and K is the shared secret key.
- ▶ If the last block $M[s]$ is too short, it is padded by $10\cdots0$ until its length equals the block size.
- ▶ If the last block has not been padded, then $K[2]$ is additionally exclusive-ored with the message. Otherwise, $K[3]$ is used instead.

CMAC (aka OMAC for One-key MAC)

- ▶ CMAC is XCBC MAC but with $K[2]$ and $K[3]$ derived from $E_{K[1]}(0^n)$.
- ▶ CMAC is recommended by NIST.

Example attacks against MACs

Example: Attack against π_1

Adverary $A_1^{\text{Tag}_K(\cdot), \text{Vf}_K(\cdot, \cdot)}$

$$M \leftarrow 0^n || 0^n$$

$$\text{Tag} \leftarrow 0^L$$

$$d \leftarrow \text{Vf}_K(M, \text{Tag})$$

$$\mathbf{Adv}_{\pi_1}^{\text{uf-cma}}(A_1) = 1$$

Resources: $t = O(n + L)$, $q_s = 0$, $q_v = 1$, $\mu_s = 0$, $\mu_v = 2n$

Example: Attack against π_2

idea:

To forge on $M = b_1b_2$, we want to know what

$$F_K([1]_m b_1) \oplus F_K([2]_m b_2)$$

looks like. So we ask a query b_1a_2 & a_1b_2 then XOR out the extras, i.e. $F_K([1]_m a_1)$ & $F_K([2]_m a_2)$.

We ask for Tag_1 of $M_1 = a_1a_2$

Tag_2 of $M_2 = a_1b_2$

Tag_3 of $M_3 = b_1a_2$

Forge on $M = b_1b_2$ with tag value $tag_1 \oplus tag_2 \oplus tag_3$.

$$M_1 : tag_1 = F_K([1]_m a_1) \oplus F_K([2]_m a_2)$$

$$M_2 : tag_2 = F_K([1]_m a_1) \oplus F_K([2]_m b_2)$$

$$M_3 : tag_3 = F_K([1]_m b_1) \oplus F_K([2]_m a_2)$$

$$M : tag = F_K([1]_m b_1) \oplus F_K([2]_m b_2)$$

Example: Attack against π_2

Adversary $A_2^{\text{Tag}_K(\cdot), \text{Vf}_K(\cdot, \cdot)}$

Let a_1, b_1 be distinct $l - m$ -bit strings

Let a_2, b_2 be distinct $l - m$ -bit strings

$\text{Tag}_1 \leftarrow \text{Tag}_K(a_1 a_2)$

$\text{Tag}_2 \leftarrow \text{Tag}_K(a_1 b_2)$

$\text{Tag}_3 \leftarrow \text{Tag}_K(b_1 a_2)$

$\text{Tag} \leftarrow \text{Tag}_1 \oplus \text{Tag}_2 \oplus \text{Tag}_3$

$d \leftarrow \text{Vf}_K(b_1 b_2, \text{Tag})$

$$\text{Adv}_{\pi_1}^{\text{uf-cma}}(A_1) = 1$$

Resources: ??

Attack against variable-length input version of CBC MAC

$$\mathbf{Adv}_{\pi}^{\text{uf-cma}}(A) = 1$$

Resources: $q_v = 1, q_s = 1, \mu_v = 2n, \mu_s = n, t = O(n)$

This attack doesn't work if input length is fixed.

In fact, CBC MAC with fixed-length inputs is secure.

Making π_2 secure

π_2 with i being a counter is **secure**.

MAC algorithm is **stateful**: ctr starts at zero and gets incremented across messages.

stateful π_2

Let $F : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^L$ be a family of functions.

$\text{Tag}_K(M) :$

static $ctr \leftarrow 0$

$t \leftarrow n - m$

If $(|M| \bmod t \neq 0 \text{ or } |M| = 0 \text{ or } ctr + \frac{|M|}{t} \geq 2^m)$ then return \perp

Break M into t -bits blocks $M = M[1] || \dots || M[s]$

For $i = 1, \dots, s$ do $y_i \leftarrow F_K([ctr + i]_m || M[i])$

$Tag \leftarrow y_1 \oplus \dots \oplus y_s$

$ctr \leftarrow ctr + s$

Return Tag

Stateful version of π_2 is a secure MAC.

Result (informal)

π is a secure *MAC* assuming that F is a PRF.

To prove this, we need to show that, $\forall A$ attacking π , we can construct B attacking F .

Idea

- ▶ B runs A using g in place of F_K to compute Tag
- ▶ If A ever forges successfully, B outputs 1. Otherwise, it outputs 0.

Common paradigm for building block-cipher-based MAC

$$\text{Prefix-free VIL PRF} \xRightarrow{*} \text{VIL PRF} \Rightarrow \text{MAC}$$

► Prefix-free VIL PRF:

- Similar to regular PRF but the adversary is only allowed to submit prefix-free queries
- Raw CBC is one example

► The conversion $\xRightarrow{*}$:

1. **ECBC**: encipher the output of the prefix-free PRF with another key before outputting the resulting tag
2. **Prefix-free encoding**: e.g., prepend the message with its length
3. **Randomized prefix-free encoding**: CMAC (Without knowledge of the xored key ($K[2], K[3]$), it is hard to find two messages for which one is a prefix of the other.)

Here, VIL = Variable Input Length. For simplicity, assume that the message length is a multiple of block length of the underlying block cipher.

MAC based on hash function

HMAC

$$\text{Tag}_K(M) = H(K \oplus \text{opad} \| H(K \oplus \text{ipad} \| M))$$