



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Introduction to Cryptography and Security

Message Integrity

Slides are taken from Dan Boneh's Course

Outline

- 1 Message Authentication Codes
- 2 MACs based on Pseudo Random Functions
- 3 MAC padding

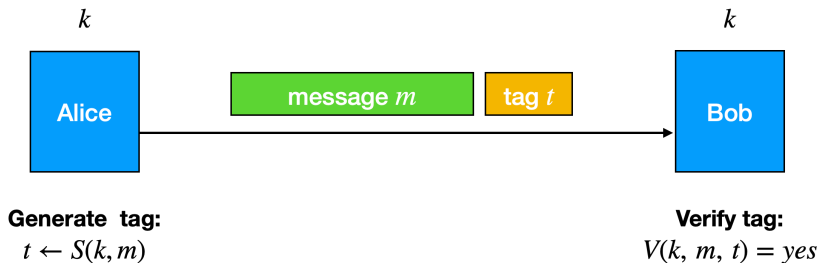
Message Integrity

Goal: **integrity**, no confidentiality.

Examples:

- Protecting public binaries on disk.
- Protecting banner ads on web pages.

Message integrity: MACs

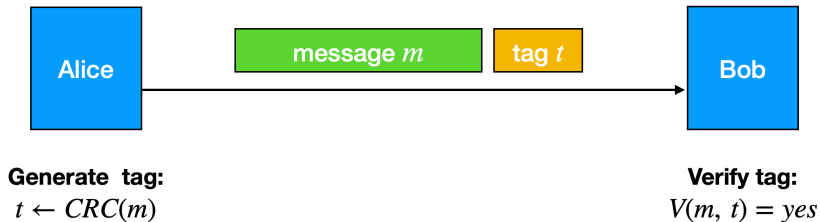


Definition

A **MAC** system $\mathcal{I} = (S, V)$ is a pair of efficient algorithms, S and V ,

- where S is called a **signing algorithm** and V is called a **verification algorithm**.
- Algorithm S is used to generate tags and algorithm V is used to verify tags.

Integrity requires a secret key



- Attacker can easily modify message m and re-compute CRC.
- CRC designed to detect **random**, not malicious errors.

Secure MACs

- Attacker's power: **chosen message attack**
for m_1, m_2, \dots, m_q attacker is given $t_i \leftarrow S(k, m_i)$
- Attacker's goal: **existential forgery**
produce some **new** valid message/tag pair (m, t)

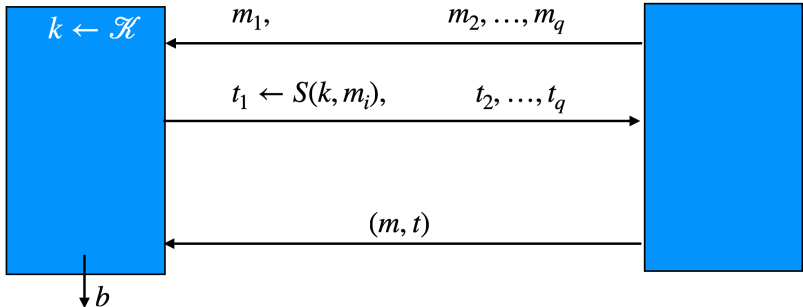
$$(m, t) \notin \{(m_1, t_1), \dots, (m_q, t_q)\}$$

- Thus, attacker cannot produce some new valid message/tag pair (m, t) .
- Given (m, t) attacker cannot even produce (m, t') for $t' \neq t$

MAC-game

MAC Challenger

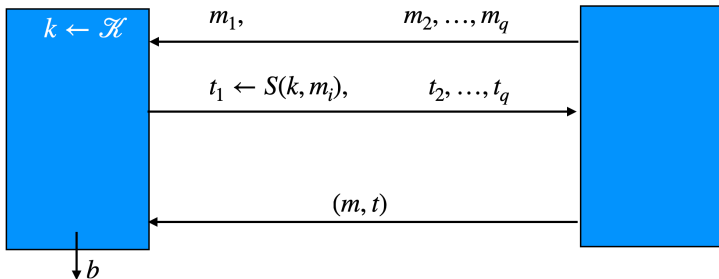
Adversary \mathcal{A}



$$\begin{cases} b = 1 & \text{if } V(k, m, t) = \text{yes} \quad \text{and} \quad (m, t) \notin \{(m_1, t_1), \dots, (m_q, t_q)\} \\ b = 0 & \text{otherwise} \end{cases}$$

MAC Challenger

Adversary \mathcal{A}



$$\begin{cases} b = 1 & \text{if } V(k, m, t) = \text{yes} \quad \text{and} \quad (m, t) \notin \{(m_1, t_1), \dots, (m_q, t_q)\} \\ b = 0 & \text{otherwise} \end{cases}$$

Definition

$I = (S, V)$ is a **secure** MAC if for all “efficient” \mathcal{A} :

$$\text{Adv}(I, \mathcal{A}) = \Pr[\text{Challenger outputs } 1] \text{ is “negligible.”}$$

Question

Let $I = (S, V)$ be a MAC. Suppose an attacker is able to find $m_0 \neq m_1$ such that

$$S(k, m_0) = S(k, m_1) \quad \text{for } 1/2 \text{ of the keys } k \in \mathcal{K}.$$

Can this MAC be secure?

- 1 Yes, the attacker cannot generate a valid tag for m_0 or m_1
- 2 No, this MAC can be broken using a chosen msg attack
- 3 It depends on the details of the MAC

Question

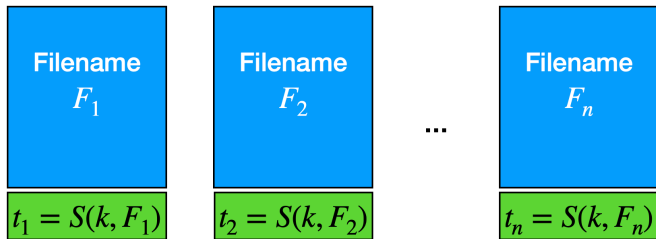
Let $I = (S, V)$ be a MAC. Suppose $S(k, m)$ is always 5 bits long.

Can this MAC be secure?

- 1 No, an attacker can simply guess the tag for messages
- 2 It depends on the details of the MAC
- 3 Yes, the attacker cannot generate a valid tag for any message

Example: protecting system files

- Suppose at install time the system computes:



- Later a virus infects system and modifies system files
- User reboots into clean OS and supplies his password
- Then: **secure MAC \Rightarrow all modified files will be detected**

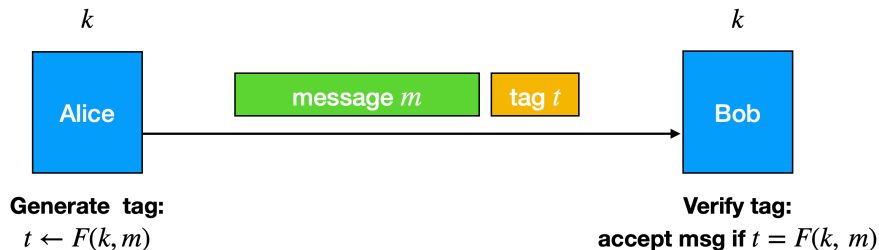
Outline

- 1 Message Authentication Codes
- 2 MACs based on Pseudo Random Functions
- 3 MAC padding

Secure PRF \Rightarrow Secure MAC

For a PRF $F: K \times X \rightarrow Y$ define a MAC $I_F = (S, V)$ as:

- $S(k, m) := F(k, m)$
- $V(k, m, t)$: output 'yes' if $t = F(k, m)$ and 'no' otherwise.



A bad example

Suppose $F: K \times X \rightarrow Y$ is a secure PRF with $Y = \{0, 1\}^{10}$.

Is the derived MAC I_F a secure MAC system?

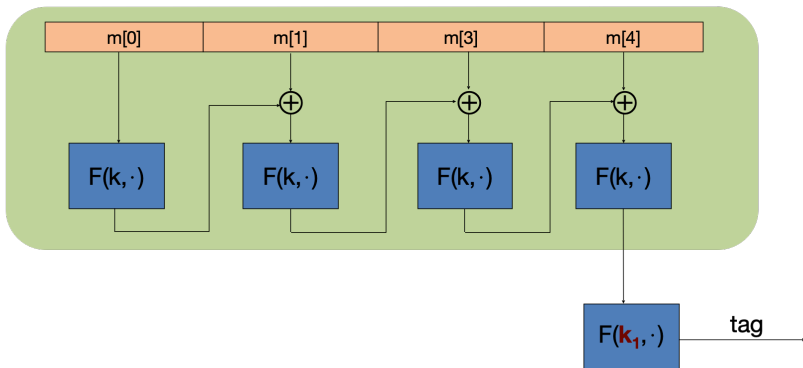
- Yes, the MAC is secure because the PRF is secure
- No tags are too short: anyone can guess the tag for any msg
- It depends on the function F

Examples

- **AES**: a MAC for 16-byte messages.
- **Main question**: how to convert Small-MAC into a Big-MAC?
- Two main constructions used in practice:
 - CBC-MAC (banking – ANSI X9.9, X9.19, FIPS 186-3)
 - HMAC (Internet protocols: SSL, IPsec, SSH,...)
 - Both convert a small-PRF into a big-PRF.

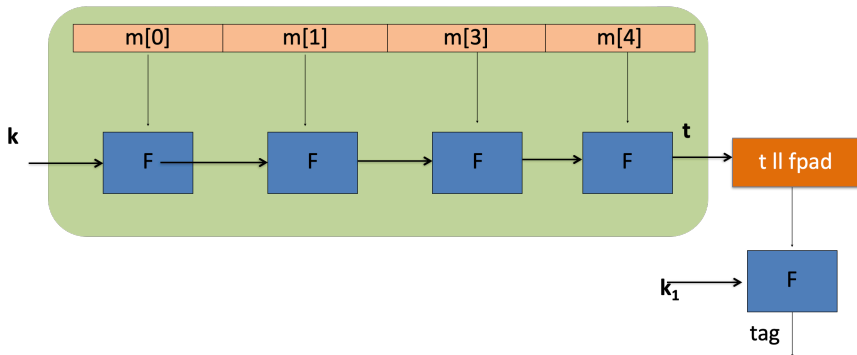
Construction 1: encrypted CBC-MAC

raw CBC



Construction 2: NMAC

cascade



Question

Why the last encryption step in NMAC?

NMAC: suppose we define a MAC $I = (S, V)$ where

$$S(k, m) = \text{cascade}(k, m)$$

- 1 This MAC is secure
- 2 This MAC can be forged without any chosen msg queries
- 3 This MAC can be forged with one chosen msg query
- 4 This MAC can be forged, but only with two msg queries

Question

Why the last encryption step in ECBC-MAC?

Suppose we define a MAC $I_{RAW} = (S, V)$ where

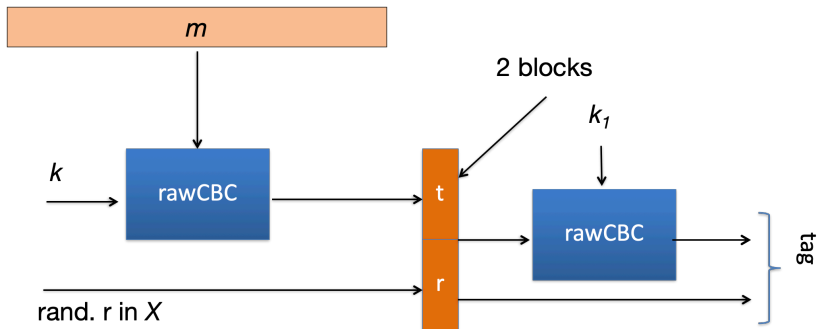
$$S(k, m) = \text{rawCBC}(k, m)$$

Then I_{RAW} is easily broken using a 1-chosen msg attack.

Adversary works as follows:

- Choose an arbitrary one-block message $m \in X$
- Request tag for m . Get $t = F(k, m)$
- Output t as MAC forgery for the 2-block message $(m, t \oplus m)$

Better security: a randomized construction



Comparison

ECBC-MAC is commonly used as an AES-based MAC

- CCM encryption mode (used in 802.11i)
- NIST standard called CMAC

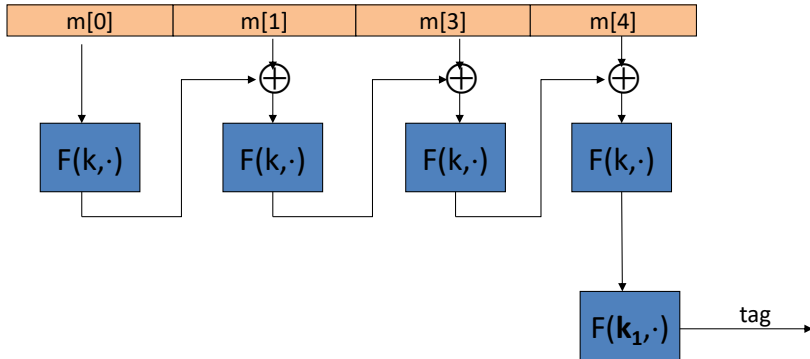
NMAC not usually used with AES or 3DES

- Main reason: need to change AES key on every block
requires re-computing AES key expansion
- But NMAC is the basis for a popular MAC called HMAC

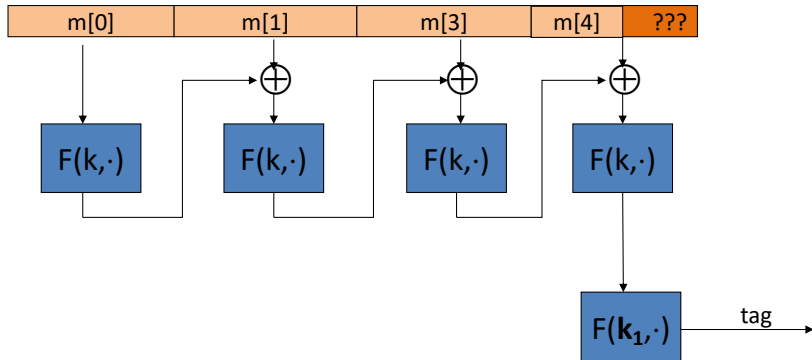
Outline

- ① Message Authentication Codes
- ② MACs based on Pseudo Random Functions
- ③ MAC padding

Recall: ECBC-MAC



What if message length is not multiple of block-size?



CBC MAC padding

Bad idea: pad m with 0's



Is the resulting MAC secure?

- ① Yes, the MAC is secure
- ② It depends on the underlying MAC
- ③ No, given tag on msg m attacker obtains tag on $m\|0$

CBC MAC padding

For security, padding must be invertible !

$$m_0 \neq m_1 \quad \Rightarrow \quad \text{pad}(m_0) \neq \text{pad}(m_1)$$

ISO: pad with $1000 \dots 0$. Add new dummy block if needed.

- The '1' indicates beginning of pad.

$m[0]$	$m[1]$
--------	--------

 \Rightarrow

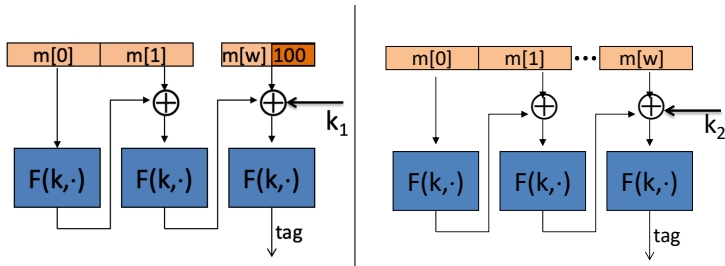
$m[0]$	$m[1]$	$1000 \dots$
--------	--------	--------------

CMAC

NIST standard

Variant of CBC-MAC where $\text{key} = (k, k_1, k_2)$

- No final encryption step (extension attack thwarted by last keyed xor)
- No dummy block (ambiguity resolved by use of k_1 or k_2)





25
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you!

