CS-3002: Information Security

Lecture # 5: Message Integrity, MACs, Collision Resistant HMAC

Prof. Dr. Sufian Hameed
Department of Computer Science
FAST-NUCES

Message Integrity

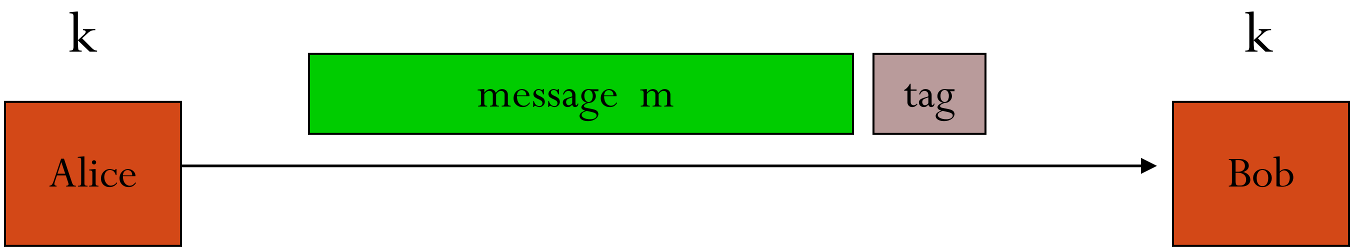
Goal: **integrity**, no confidentiality.

- There are Scenarios where integrity is important and confidentiality is not required

Examples:

- Protecting public binaries on disk
  - To prevent malicious manipulation
- Protecting banner ads on web pages.

Message integrity: MACs



Generate tag:

$\mathbf{tag \leftarrow S(k, m)}$

Verify tag:

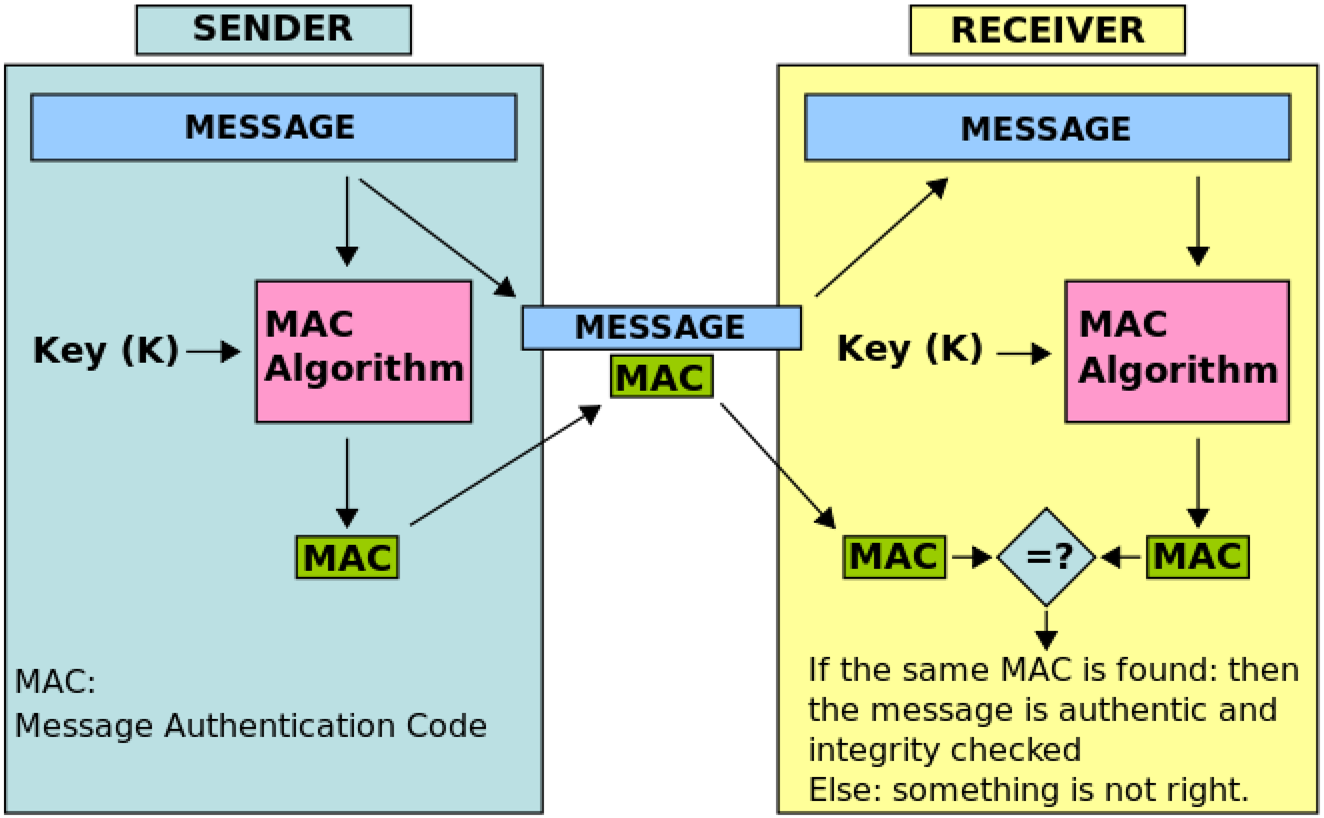
$\mathbf{V(k, m, tag) \stackrel{?}{=} \text{`yes'}}$

- MAC is short information
  - Provide integrity and authenticity assurances on the message.
  - Integrity assurances detects accidental and intentional message changes
  - Authenticity assurances affirms the message's origin

Def: **MAC** I = (S,V) defined over (K,M,T) is a pair of algs:

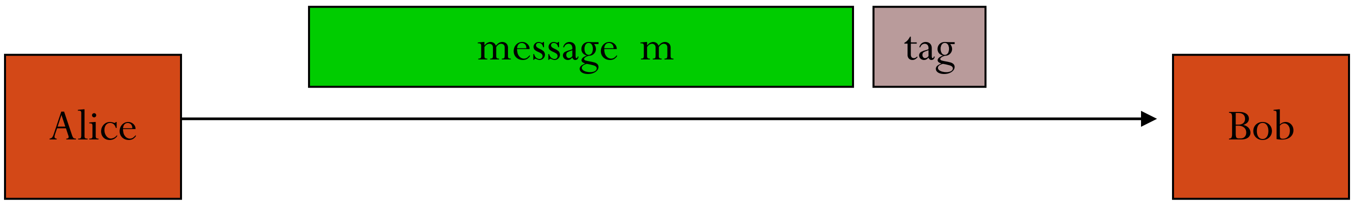
- S(k,m) outputs t in T
- V(k,m,t) outputs \`yes’ or \`no’

MAC



*Source = Wiki

Integrity requires a secret key ?



Generate tag:

$\text{tag} \leftarrow \text{CRC}(m)$

Verify tag:

$V(m, \text{tag}) \stackrel{?}{=} \text{`yes'}$

- Attacker can easily modify message m and re-compute CRC.

- CRC designed to detect **random**, not malicious errors.

Secure MACs

MAC:

- Signing Alg. **$S(k,m) \longrightarrow t$** and
- Verification alg. **$V(k,m,t) \longrightarrow 0,1$**

Attacker's power: **chosen message attack**

- for $m_1, m_2, \ldots, m_q$ attacker is given $t_i \leftarrow S(k, m_i)$

Attacker's goal: **existential forgery**

- produce some **new** valid message/tag pair (m,t).

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

$\Rightarrow$ attacker cannot produce a valid tag for a new message

$\Rightarrow$ given (m,t) attacker cannot even produce (m,t') for $t' \neq t$

Lets Make Secure MACs

A bad example

Suppose **$F: K \times X \longrightarrow Y$** 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

$Adv[A, I_F] = 1/1024$

Security

<u>Thm</u>: If **F: K×X⟶Y** is a secure PRF and 1/|Y| is negligible (i.e. |Y| is large) then I_F is a secure MAC.

$\Rightarrow$ I_F is secure as long as |Y| is large, say $|Y| = 2^{80}$.

Examples

- AES (a secure PRF): 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

Truncating MACs based on PRFs

Easy lemma: suppose $F: \mathbf{K} \times \mathbf{X} \longrightarrow \{0,1\}^n$ is a secure PRF.

Then so is $\mathbf{F_t(k,m)} = \underbrace{\mathbf{F(k,m)[1\ldots t]}}_{\text{first t-bit of output}}$ for all $1 \leq t \leq n$

$\Rightarrow$ if (S,V) is a MAC based on a secure PRF outputting n-bit tags

the truncated MAC outputting w bits is secure

… as long as $1/2^w$ is still negligible (say $w \geq 64$)

CBC-MAC and NMAC

MACs and PRFs

Recall: secure PRF **F** $\Rightarrow$ secure MAC, as long as $|Y|$ is large

$$S(k, m) = F(k, m)$$

Our goal:

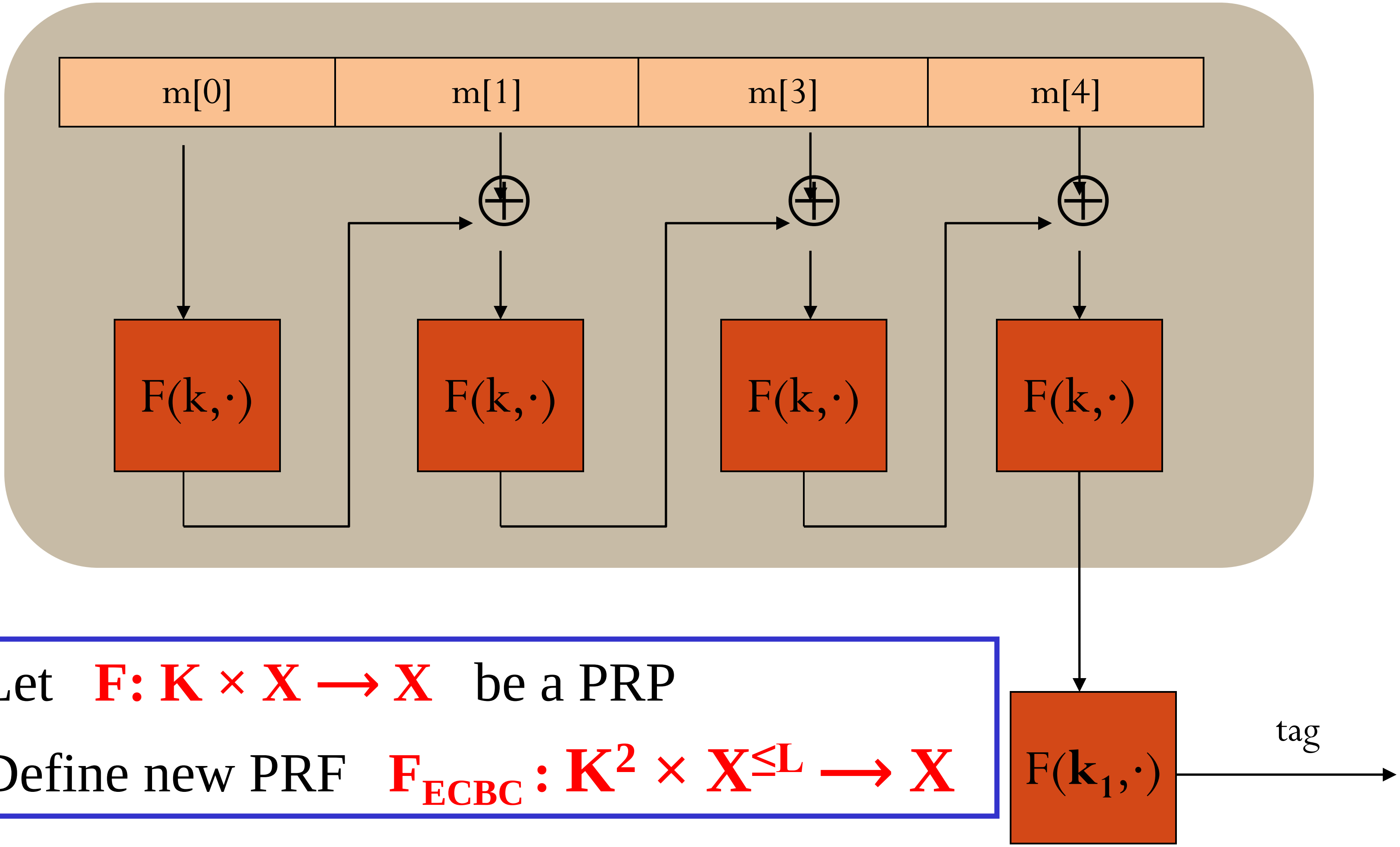
given a PRF for short messages (AES)

construct a PRF for long messages

From here on let $X = \{0,1\}^n$ (e.g. n=128)

Construction 1: encrypted CBC-MAC

raw CBC

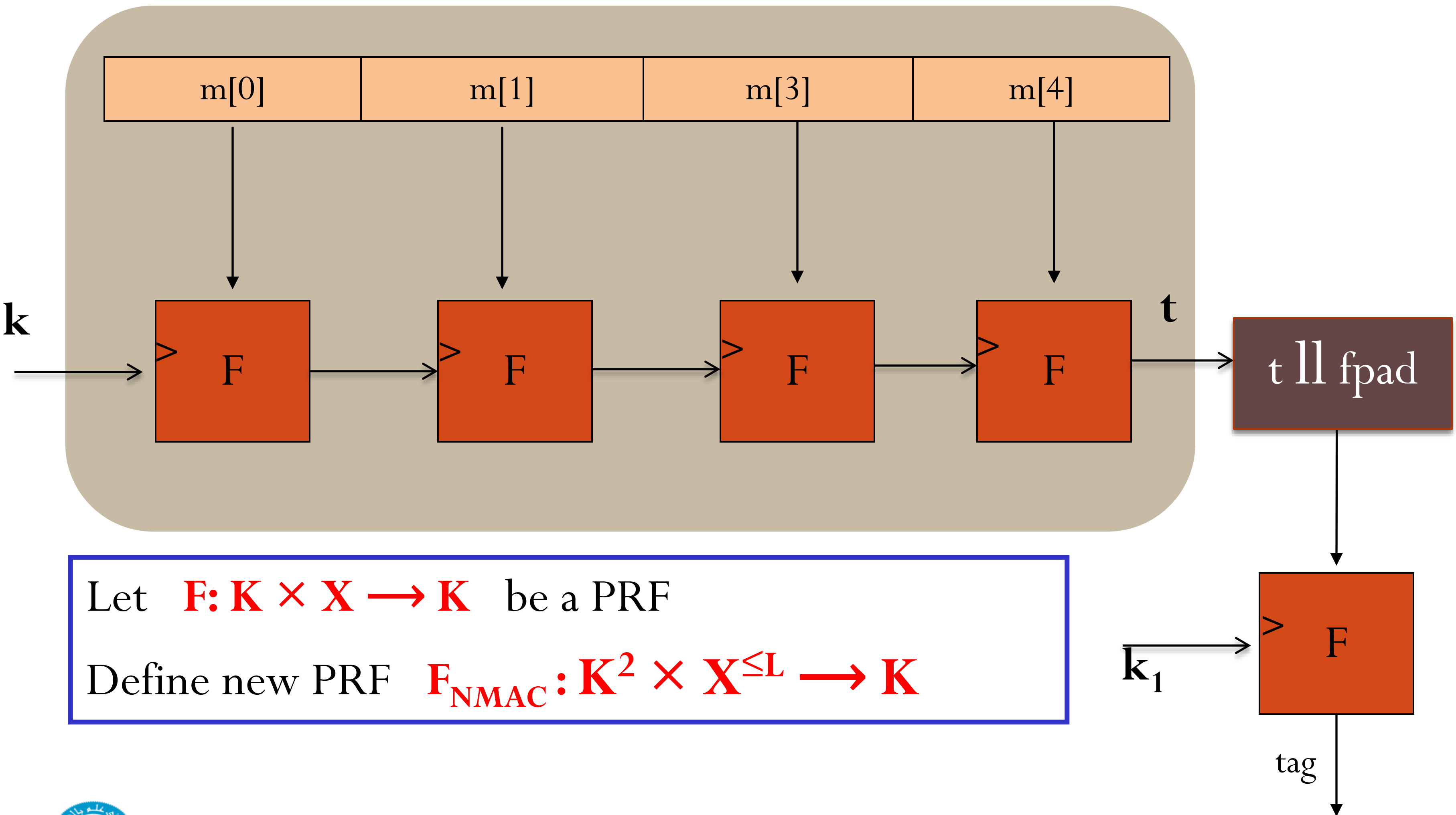


Let $\mathbf{F: K \times X \longrightarrow X}$ be a PRP

Define new PRF $\mathbf{F_{ECBC} : K^2 \times X^{\leq L} \longrightarrow X}$

Construction 2: NMAC (nested MAC)

cascade



Let **$F: K \times X \longrightarrow K$** be a PRF

Define new PRF **$F_{NMAC}: K^2 \times X^{\leq L} \longrightarrow K$**

Why the last encryption step in ECBC-MAC and NMAC?

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

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

This MAC is secure

This MAC can be forged without any chosen msg queries

This MAC can be forged with one chosen msg query

This MAC can be forged, but only with two msg queries

$\text{Cascade}(k,m) \Rightarrow \text{cascade}(k, m \| w)$ for any w

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 $(m, t \oplus m)$

Indeed: $\text{rawCBC}(k, (m, t\oplus m)) = F(k, F(k,m)\oplus(t\oplus m)) =$
$F(k, t\oplus(t\oplus m)) = t$

Comparison

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

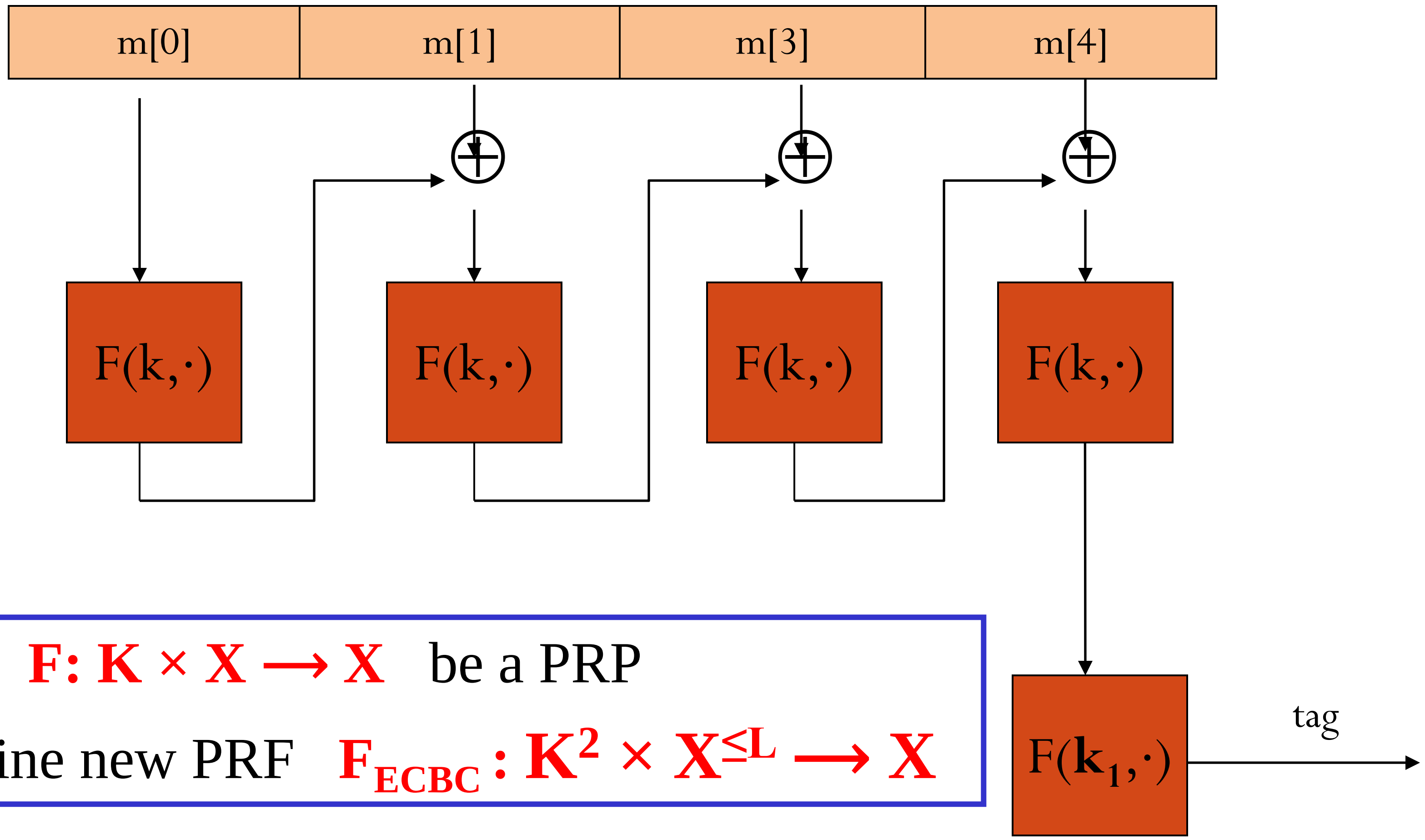
- CCM encryption mode (used in 802.11i)
- NIST standard called CMAC

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 (next)

MAC Padding

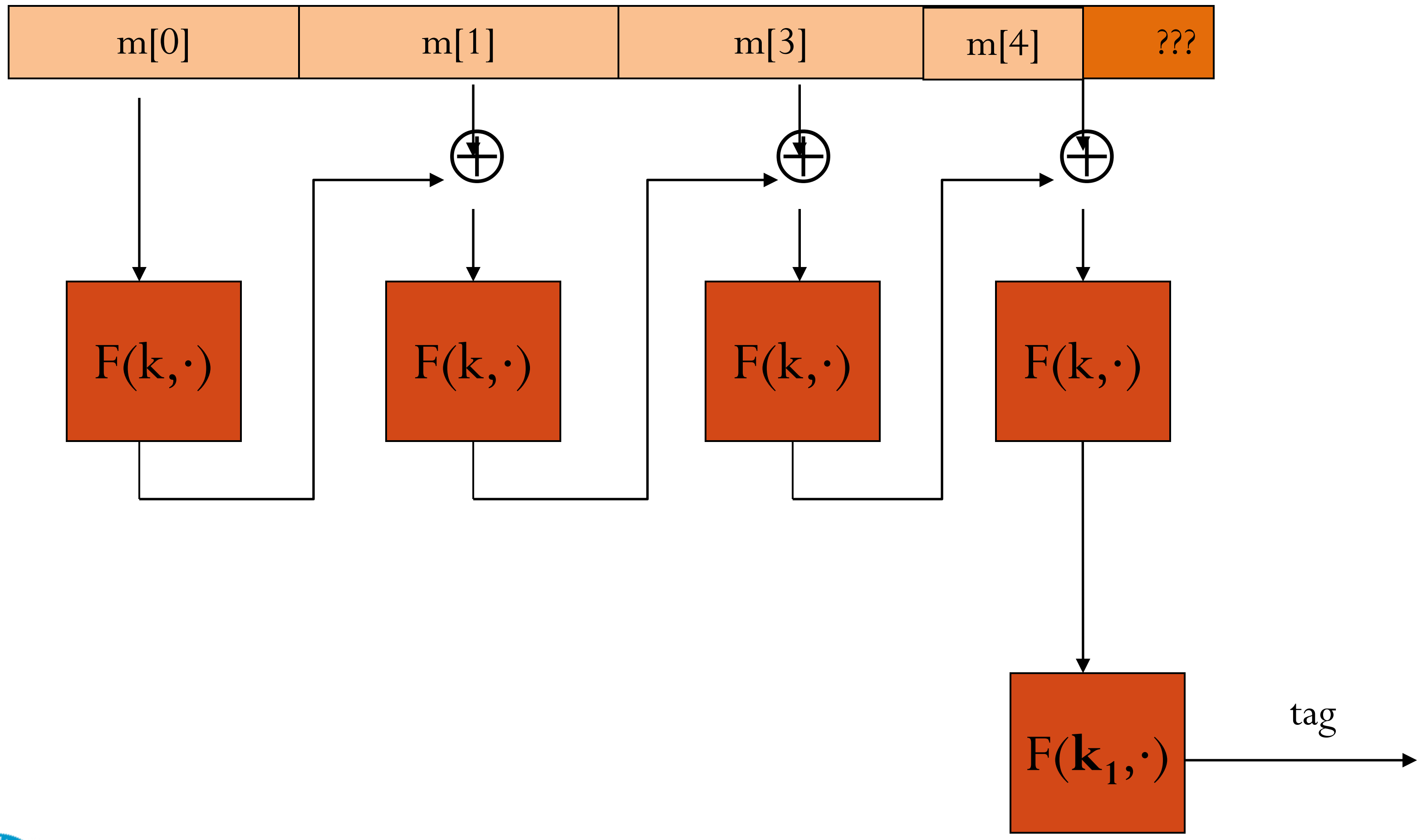
Recall: ECBC-MAC



Let $\mathbf{F}: \mathbf{K} \times \mathbf{X} \longrightarrow \mathbf{X}$ be a PRP

Define new PRF $\mathbf{F}_{\mathbf{ECBC}} : \mathbf{K}^2 \times \mathbf{X}^{\leq L} \longrightarrow \mathbf{X}$

What if msg. len. is not multiple of block-size?



CBC MAC padding

Bad idea: pad m with 0’s

m[0]	m[1]		m[0]	m[1]	0000

Is the resulting MAC secure?

Yes, the MAC is secure

It depends on the underlying MAC

No, given tag on msg **m** attacker obtains tag on **m||0**

Problem: pad(m) = pad(m||0)

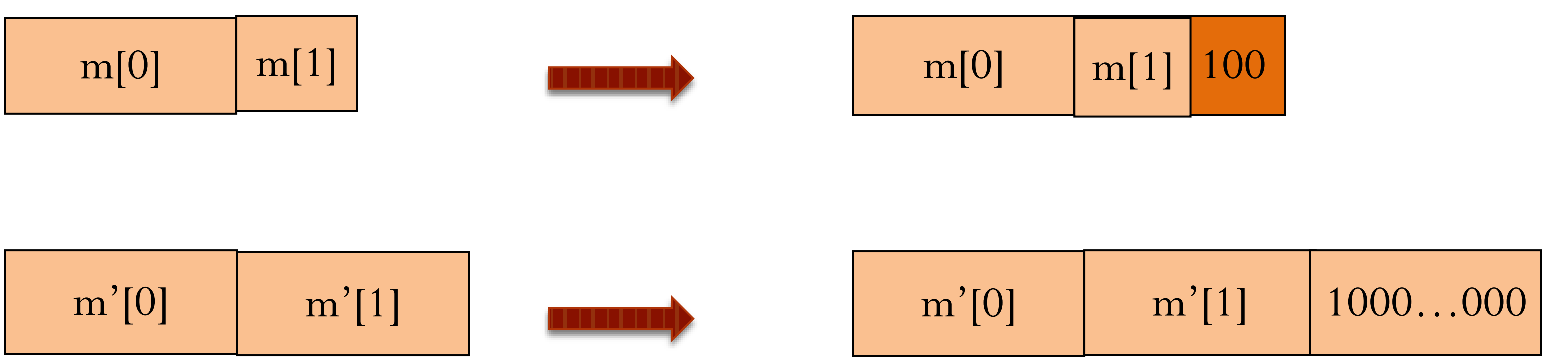
CBC MAC padding

For security, padding must be invertible !

$$m_0 \neq m_1 \quad \Rightarrow \quad pad(m_0) \neq pad(m_1)$$

<u>ISO</u>: pad with "1000…00". Add new dummy block if needed.

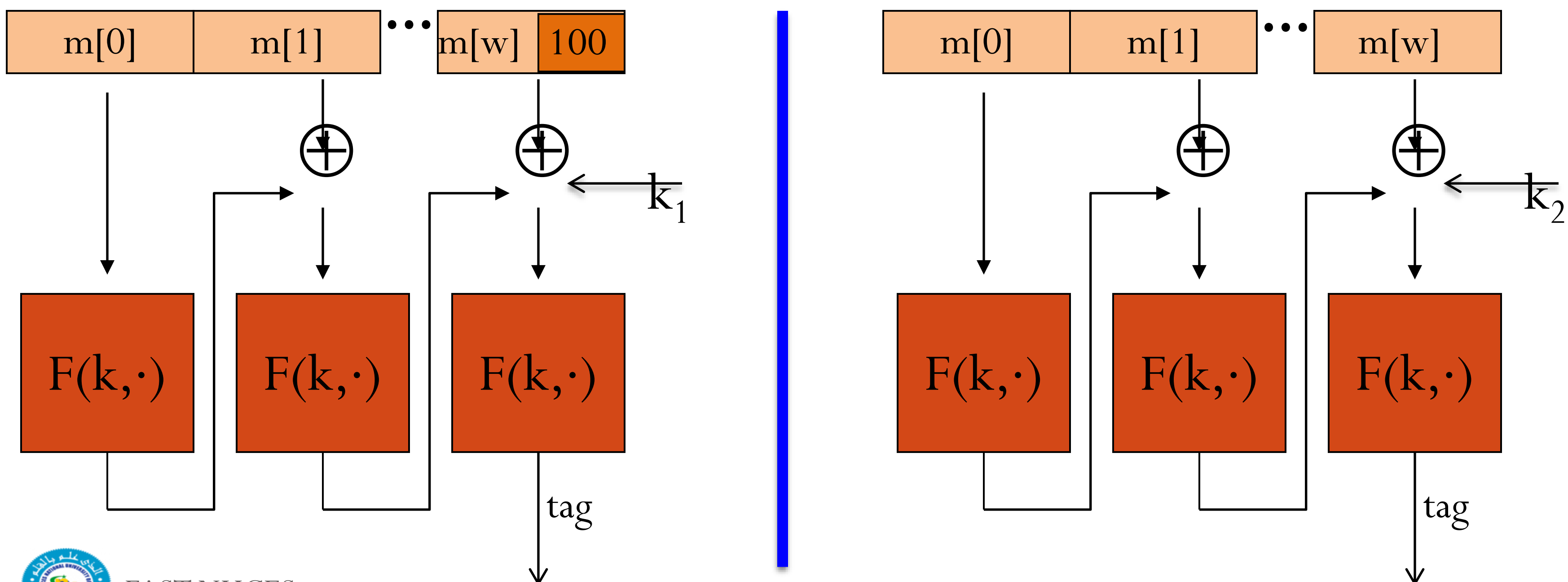
- The "1" indicates beginning of pad.



CMAC (NIST standard)

Variant of CBC-MAC where key = (k, k_1, k_2) (k_1, k_2) derived from k

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



Parallel- MAC (PMAC)

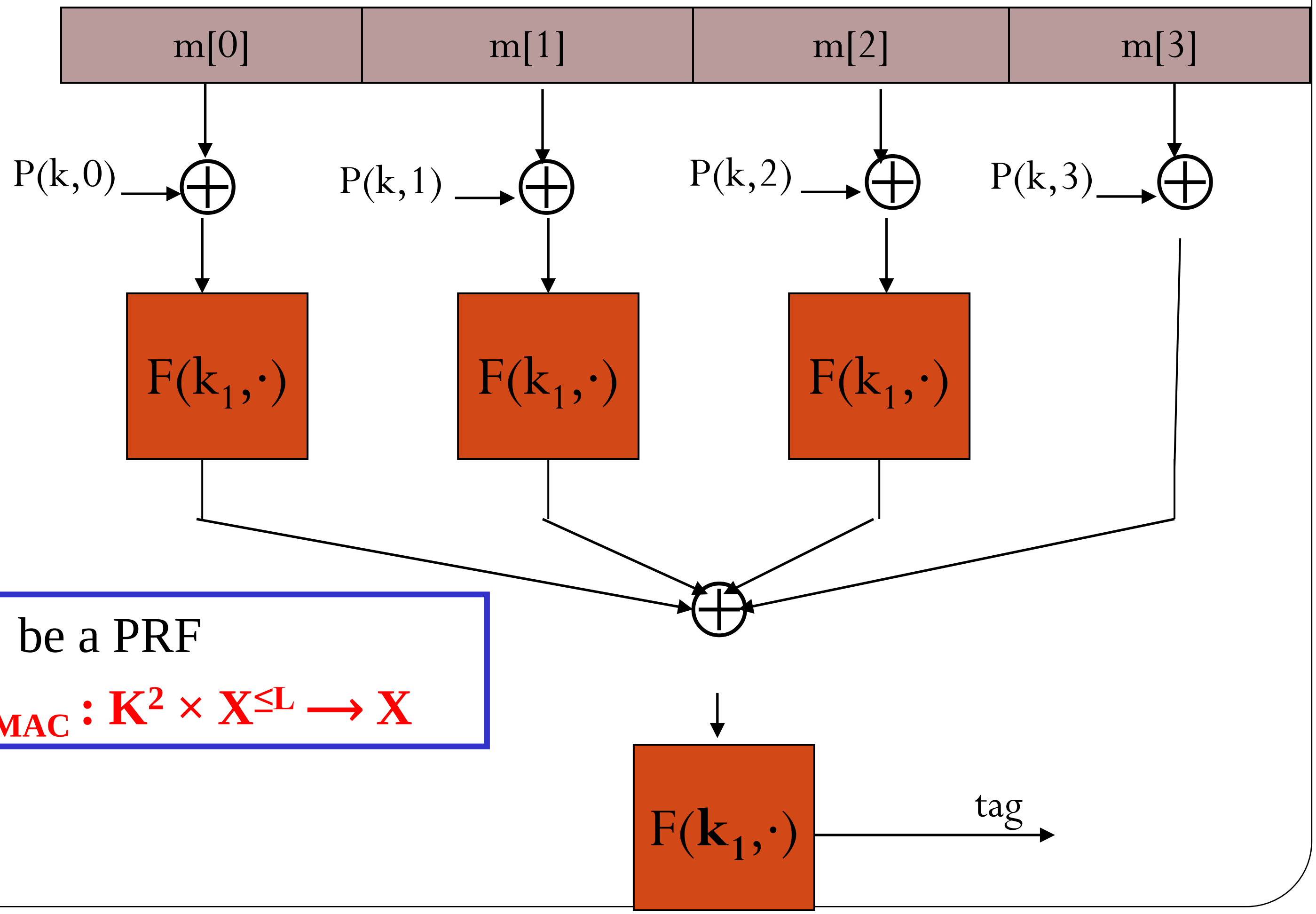
- ECBC and NMAC are sequential.
- Can we build a parallel MAC from a small PRF ??

Construction 3: PMAC – parallel MAC

P(k, i): an easy to compute function

key = (k, k_1)

Padding similar to CMAC



Let $\mathbf{F: K \times X \longrightarrow X}$ be a PRF

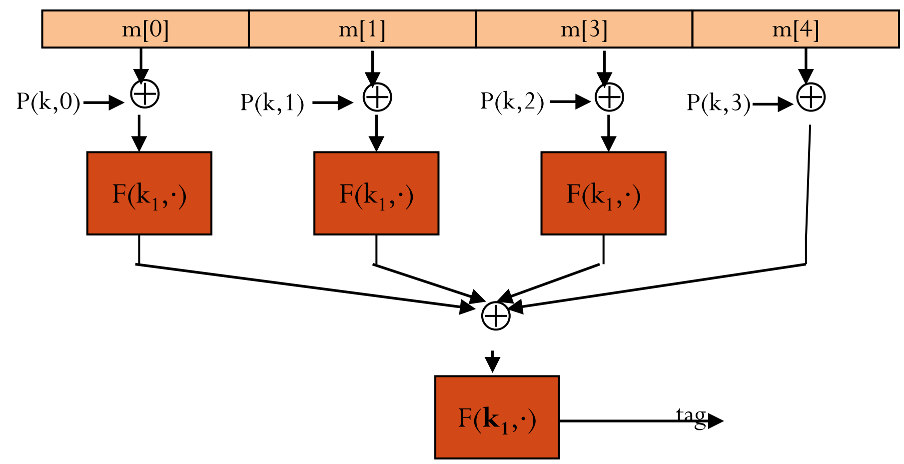
Define new PRF $\mathbf{F_{PMAC} : K^2 \times X^{\leq L} \longrightarrow X}$

PMAC is incremental

→ m'[1]

Suppose F is a PRP.

When m[1] ⟶ m'[1]
can we quickly update tag?



no, it can't be done

do $F^{-1}(k_1,tag) \oplus F(k_1, m'[1] \oplus P(k,1))$

do $F^{-1}(k_1,tag) \oplus F(k_1, m[1] \oplus P(k,1)) \oplus F(k_1, m'[1] \oplus P(k,1))$

do $tag \oplus F(k_1, m[1] \oplus P(k,1)) \oplus F(k_1, m'[1] \oplus P(k,1))$

Then apply $F(k_1, \cdot)$

Construction 4: HMAC (Hash-MAC)

Most widely used MAC on the Internet.

… but, we first we need to discuss hash function.

Further reading

- J. Black, P. Rogaway: CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. J. Cryptology 18(2): 111-131 (2005)
- K. Pietrzak: A Tight Bound for EMAC. ICALP (2) 2006: 168-179
- J. Black, P. Rogaway: A Block-Cipher Mode of Operation for Parallelizable Message Authentication. EUROCRYPT 2002: 384-397
- M. Bellare: New Proofs for NMAC and HMAC: Security Without Collision-Resistance. CRYPTO 2006: 602-619
- Y. Dodis, K. Pietrzak, P. Puniya: A New Mode of Operation for Block Ciphers and Length-Preserving MACs. EUROCRYPT 2008: 198-219

Recap: message integrity

So far, four MAC constructions:

ECBC-MAC, CMAC : commonly used with AES (e.g. 802.11i)

NMAC : basis of HMAC (this segment)

PMAC: a parallel MAC

This module: MACs from collision resistance.

Collision Resistance

Let H: M $\rightarrow$T be a hash function (|M| >> |T|)

A **<u>collision</u>** for H is a pair m_0 , $m_1 \in$ M such that:

$$H(m_0) = H(m_1) \quad \text{and} \quad m_0 \neq m_1$$

A function H is **<u>collision resistant</u>** if for all (explicit) “eff” algs. A:

$\mathrm{Adv}_{\mathrm{CR}}[A,H]$ = Pr[A outputs collision for H]

is “neg”.

Example: SHA-256 (outputs 256 bits)

MACs from Collision Resistance

Let $I = (S,V)$ be a MAC for short messages over (K,M,T) (e.g. AES)

Let $H: M^{big} \to M$

Def: $I^{big} = (S^{big}, V^{big})$ over (K, M^{big}, T) as:

$$\mathbf{S^{big}(k,m) = S(k,H(m)) \quad ; \quad V^{big}(k,m,t) = V(k,H(m),t)}$$

Thm: If I is a secure MAC and H is collision resistant then I^{big} is a secure MAC.

Example: $S(k,m) = AES_{2\text{-block-cbc}}(k, \text{SHA-256}(m))$ is a secure MAC.

MACs from Collision Resistance

$S^{big}(k, m) = S(k, H(m))$; $V^{big}(k, m, t) = V(k, H(m), t)$

Collision resistance is necessary for security:

Suppose adversary can find $m_0 \neq m_1$ s.t. $H(m_0) = H(m_1)$.

Then: **S^{big}** is insecure under a 1-chosen msg attack

step 1: adversary asks for $t \longleftarrow S(k, m_0)$

step 2: output (m_1, t) as forgery

Protecting file integrity using C.R. hash

Software packages:

package name F_1 | package name F_2 | ••• | package name F_n

read-only public space

$H(F_1)$ $H(F_2)$ $H(F_n)$

When user downloads package, can verify that contents are valid

H collision resistant $\Rightarrow$
attacker cannot modify package without detection

no key needed (public verifiability), but requires read-only space

Generic Birthday Attack

Generic attack on C.R. functions

Let H: $M \rightarrow \{0,1\}^n$ be a hash function ($|M| >> 2^n$)

Generic alg. to find a collision **in time** $\mathbf{O(2^{n/2})}$ hashes

Algorithm:

1. Choose $\mathbf{2^{n/2}}$ random messages in M: $m_1, \ldots, m_{2^{n/2}}$ (distinct w.h.p)
2. For $i = 1, \ldots, 2^{n/2}$ compute $t_i = H(m_i) \in \{0,1\}^n$
3. Look for a collision ($t_i = t_j$). If not found, go back to step 1.

How well will this work?

The birthday paradox

Let $r_1, \ldots, r_n \in \{1,\ldots,B\}$ be indep. identically distributed integers.

Thm: when $\mathbf{n} = 1.2 \times \mathbf{B}^{1/2}$ then $\Pr[\, \exists i \neq j:\ r_i = r_j \,] \geq \frac{1}{2}$

- **The Birthday Problem**
  - Let there be n people in a room
  - For what value of n two people will share the same birthday or What is the probability of two people sharing the same birthday?
- **Link to hash functions**
  - Collisions more likely for pairwise matching

Generic Attack

$H: M \to \{0,1\}^n$. Collision finding algorithm:

1. Choose $\mathbf{2^{n/2}}$ random elements in M: $m_1, \ldots, m_{2^{n/2}}$
2. For i = 1, …, $2^{n/2}$ compute $t_i = H(m_i) \in \{0,1\}^n$
3. Look for a collision $(t_i = t_j)$. If not found, got back to step 1.

Expected number of iteration ≈ 2

Running time: $\mathbf{O(2^{n/2})}$ (space $O(2^{n/2})$)

Sample C.R. hash functions:

AMD Opteron, 2.2 GHz (Linux)

	function	digest size (bits)	Speed (MB/sec)	generic attack time
NIST standards	SHA-1	160	153	2^{80}
	SHA-256	256	111	2^{128}
	SHA-512	512	99	2^{256}
	Whirlpool	512	57	2^{256}

best known collision (theoratical) finder for SHA-1 requires 2^{51} hash evaluations. Other than that there are no known collisions

The Merkle-Damgard Paradigm

Collision resistance: Review

Let H: M $\rightarrow$T be a hash function (|M| >> |T|)

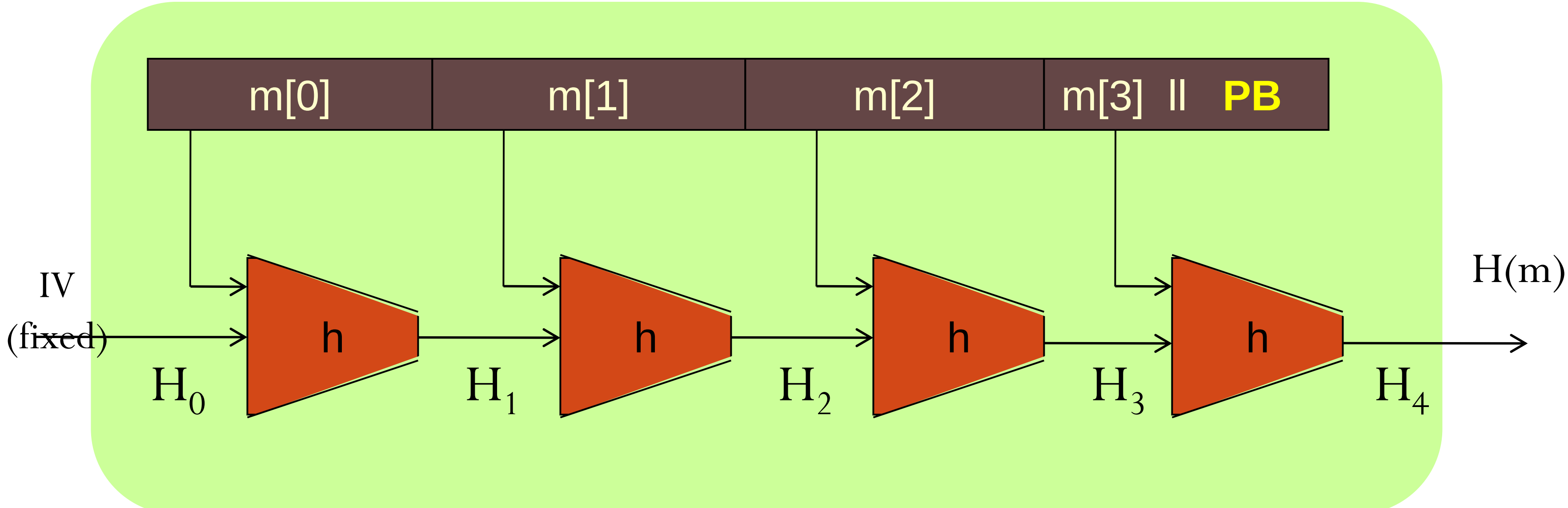
A **collision** for H is a pair m_0 , $m_1 \in$ M such that:

$$\mathbf{H(m_0) = H(m_1) \quad and \quad m_0 \neq m_1}$$

Goal: collision resistant (C.R.) hash functions

Step 1: given C.R. function for **short** messages,
construct C.R. function for **long** messages

The Merkle-Damgard iterated construction



Given **$h: T \times X \longrightarrow T$** (compression function)

we obtain **$H: X^{\leq L} \longrightarrow T$**. H_i - chaining variables

If no space for PB
add another block

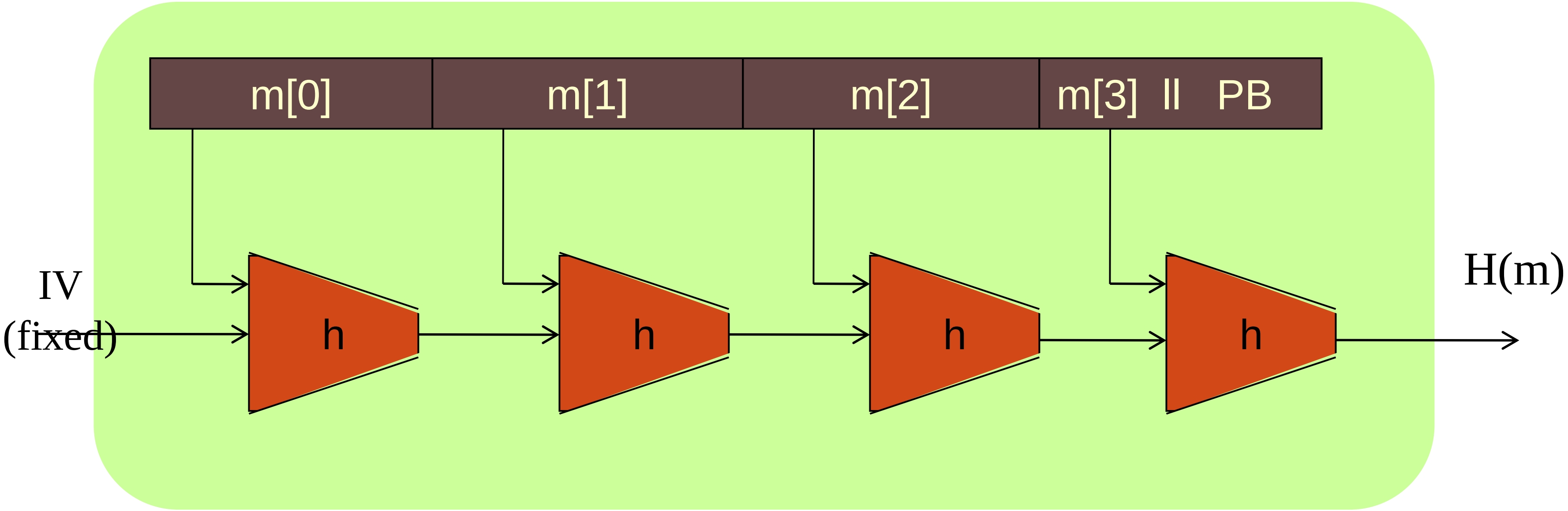
MD collision resistance

Thm: if h is collision resistant then so is H.

collision on H $\Rightarrow$ collision on h

Constructing Compression Function

The Merkle-Damgard iterated construction



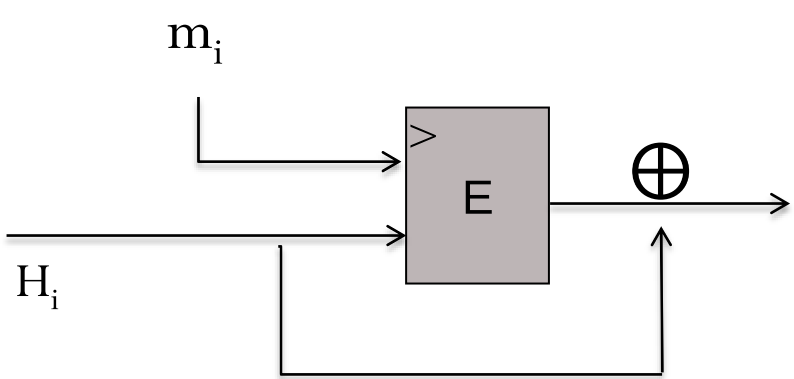
Thm: h collision resistant $\Rightarrow$ H collision resistant

Goal: construct compression function **h: T × X ⟶ T**

Compression. func. from a block cipher

E: K× $\{0,1\}^n \longrightarrow \{0,1\}^n$ a block cipher

The **Davies-Meyer** compression function: **$h(H, m) = E(m, H) \oplus H$**



<u>Thm</u>: Suppose E is an ideal cipher (collection of |K| random perms.).
Finding a collision **h(H,m)=h(H',m')** takes **$O(2^{n/2})$** evaluations of (E,D).

Best possible !!

Other block cipher constructions

Let $\mathbf{E: \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}^n}$ for simplicity

Miyaguchi-Preneel: $\mathbf{h(H, m) = E(m, H)\oplus H\oplus m}$

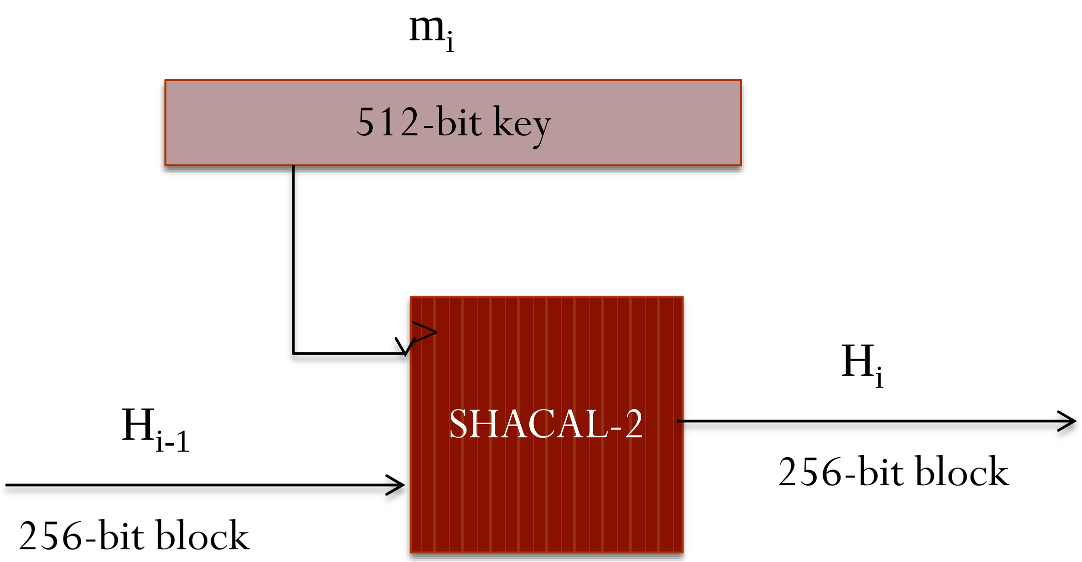
(Whirlpool)

$\mathbf{h(H, m) = E(H\oplus m, m)\oplus m}$

total of 12 variants like this

Case study: SHA-256

- Merkle-Damgard function
- Davies-Meyer compression function
- Block cipher: SHACAL-2



Provable compression functions

Choose a random 2000-bit prime p and random $1 \leq u, v \leq p$

.

For $m,h \in \{0,\ldots,p\text{-}1\}$ define $\mathbf{h(H,m) = u^H \cdot v^m \quad (mod\ p)}$

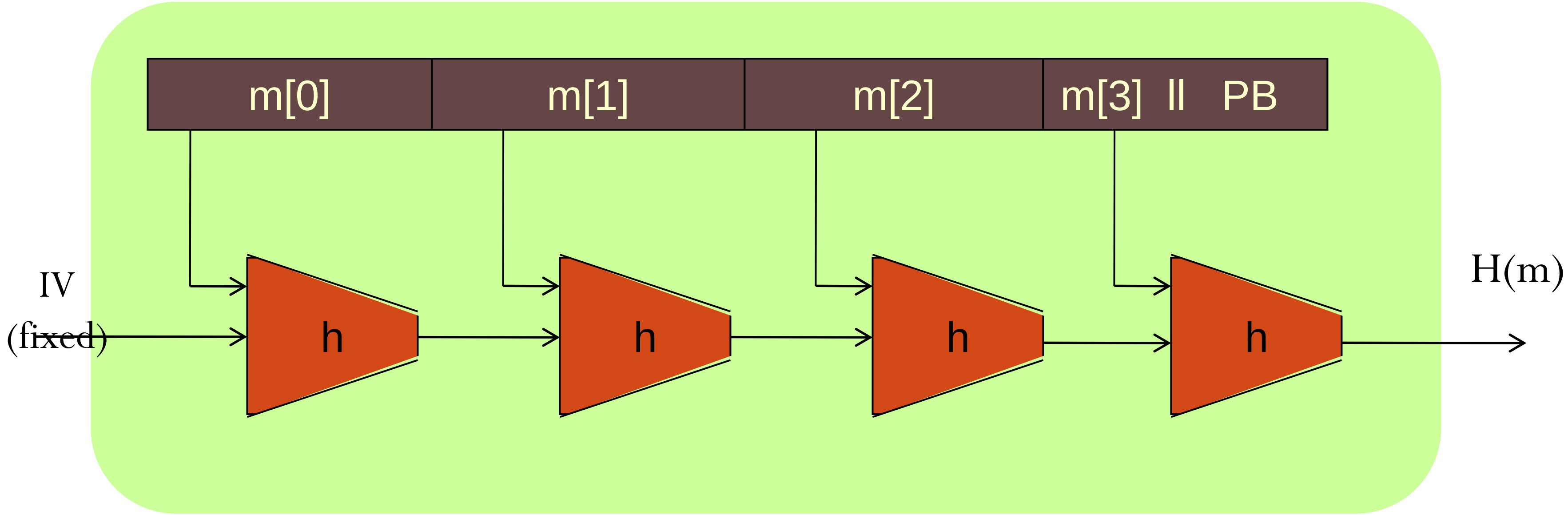
Compression is 2:1

<u>Fact:</u> finding collision for h(.,.) is as hard as
solving "discrete-log" modulo p.

Problem: slow

HMAC: a MAC from SHA-256

The Merkle-Damgard iterated construction



Thm: h collision resistant $\Rightarrow$ H collision resistant

Can we use H(.) to directly build a MAC?

MAC from a Merkle-Damgard Hash Function

$H: X^{\leq L} \longrightarrow T$ a C.R. Merkle-Damgard Hash Function

<u>Attempt #1</u>: **$S(k, m) = H(k \parallel m)$**

This MAC is insecure because:

Given $H(k \parallel m)$ can compute $H(w \parallel k \parallel m \parallel PB)$ for any w.

Given $H(k \parallel m)$ can compute $H(k \parallel m \parallel w)$ for any w.

Given $H(k \parallel m)$ can compute $H(k \parallel m \parallel PB \parallel w)$ for any w.

Anyone can compute $H(k \parallel m)$ for any m.

Standardized method: HMAC (Hash-MAC)

Most widely used MAC on the Internet.

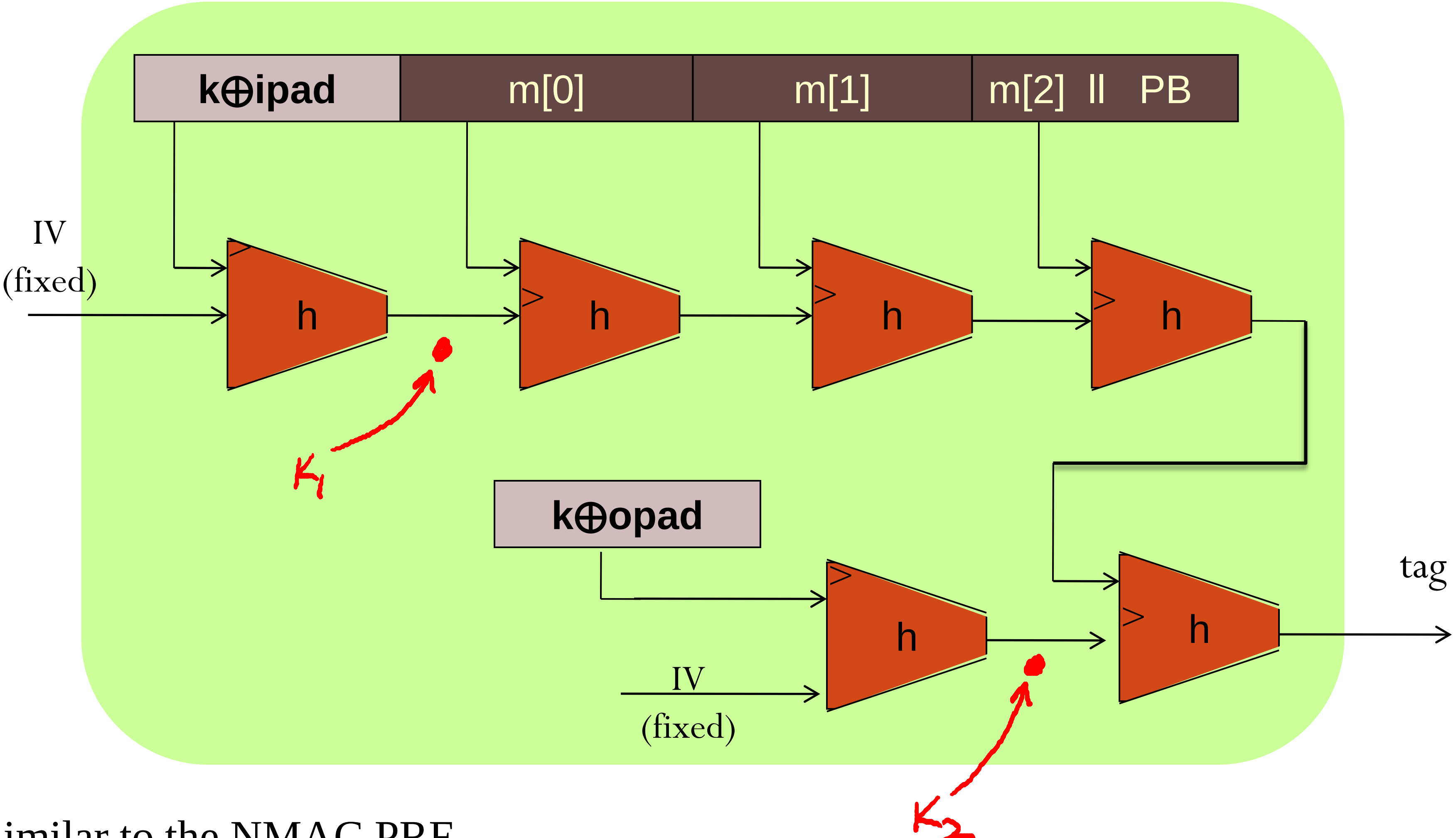
H: hash function.

example: SHA-256 ; output is 256 bits

Building a MAC out of a hash function:

HMAC: $S(k, m) = H\big(k \oplus opad \ \| \ \mathbf{H(k \oplus ipad \ \| \ m)} \big)$

HMAC in pictures



Similar to the NMAC PRF.

main difference: the two keys k_1, k_2 are dependent

Timing Attacks on MAC Verification

Warning: verification timing attacks [L'09]

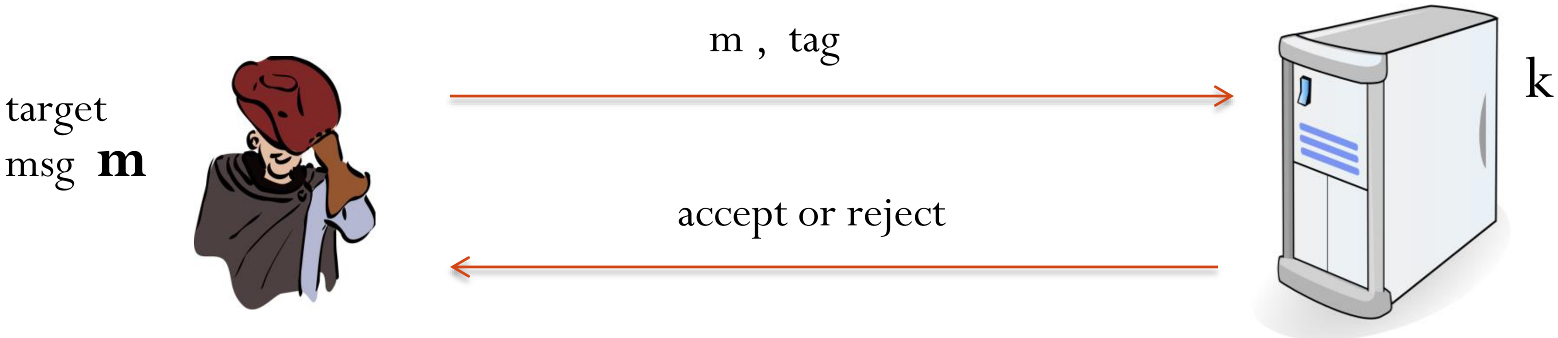
Example: Keyczar crypto library (Python) [simplified]

```
def Verify(key, msg, sig_bytes):
    return HMAC(key, msg) == sig_bytes
```

The problem: '==' implemented as a byte-by-byte comparison

- Comparator returns false when first inequality found

Warning: verification timing attacks [L'09]



Timing attack: to compute tag for target message m do:

Step 1: Query server with random tag

Step 2: Loop over all possible first bytes and query server.

stop when verification takes a little longer than in step 1

Step 3: repeat for all tag bytes until valid tag found

Defense #1

Make string comparator always take same time (Python) :

```
return false if  sig_bytes  has wrong length
result = 0
for x, y in zip( HMAC(key,msg) , sig_bytes):
      result |= ord(x) ^ ord(y)
return result == 0
```

Can be difficult to ensure due to optimizing compiler.

Defense #2

Make string comparator always take same time (Python) :

```
def Verify(key, msg, sig_bytes):
    mac = HMAC(key, msg)
    return HMAC(key, mac) == HMAC(key, sig_bytes)
```

Attacker doesn't know values being compared

Lesson

Don’t implement crypto yourself !

Acknowledgements

Material in this lecture are taken from the slides prepared by:

- Prof. Dan Boneh (Stanford)