



Collision resistance

Introduction

Recap: message integrity

So far, four MAC constructions:

PRFs { **ECBC-MAC, CMAC** : commonly used with AES (e.g. 802.11i)
NMAC : basis of HMAC (this segment)
PMAC: a parallel MAC

randomized
MAC { **Carter-Wegman MAC**: built from a fast one-time MAC

This module: MACs from collision resistance.

Collision Resistance

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

A **collision** 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 **collision resistant** if for all (explicit) “eff” algs. A :

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

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 $H: M^{\text{big}} \rightarrow M$

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

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

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

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

MACs from Collision Resistance

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

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 $t \leftarrow S(k, m_0)$

step 2: output (m_1, t) as forgery

Protecting file integrity using C.R. hash

Software packages:



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

End of Segment



Collision resistance

Generic birthday attack

Generic attack on C.R. functions

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

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

Algorithm:

1. Choose $2^{n/2}$ random messages in M : $m_1, \dots, m_{2^{n/2}}$ (distinct w.h.p)
2. For $i = 1, \dots, 2^{n/2}$ 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.

How well will this work?

The birthday paradox

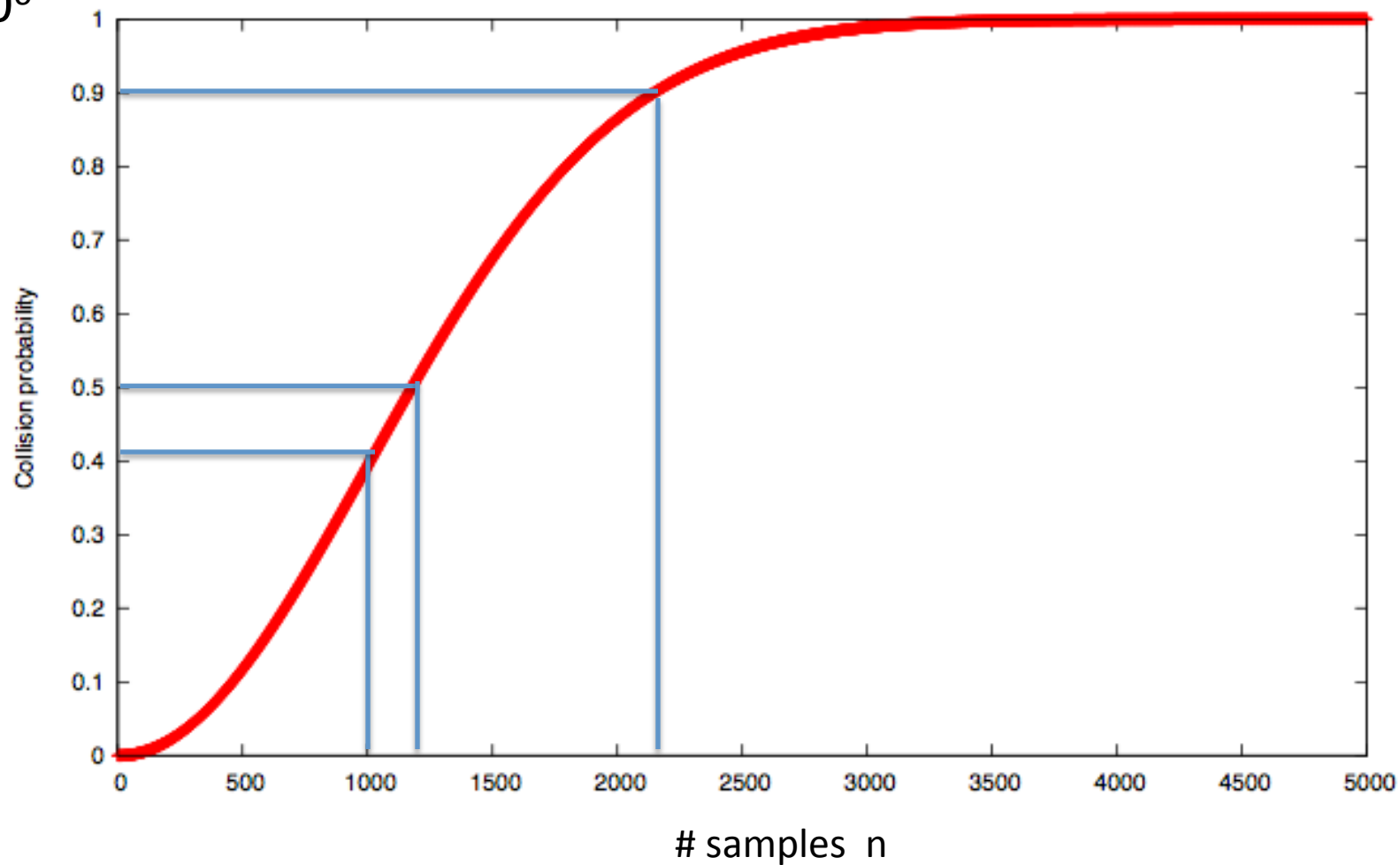
Let $r_1, \dots, r_n \in \{1, \dots, B\}$ be indep. identically distributed integers.

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

Proof: (for uniform indep. r_1, \dots, r_n)

$$\begin{aligned}\Pr[\exists i \neq j: r_i = r_j] &= 1 - \Pr[\forall i \neq j: r_i \neq r_j] = 1 - \left(\frac{B-1}{B}\right)\left(\frac{B-2}{B}\right) \cdots \left(\frac{B-n+1}{B}\right) = \\ &= 1 - \prod_{i=1}^{n-1} \left(1 - \frac{i}{B}\right) \geq 1 - \prod_{i=1}^{n-1} e^{-i/B} = 1 - e^{-\frac{1}{B} \sum_{i=1}^{n-1} i} \geq 1 - e^{-n^2/2B} \\ &\quad \uparrow \quad \quad \quad \uparrow \\ &\quad 1-x \leq e^{-x} \quad \quad \quad \frac{n^2}{2B} = 0.72 \end{aligned}$$
$$\geq 1 - e^{-0.72} = 0.53 > \frac{1}{2}$$

$B=10^6$



Generic attack

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

1. Choose $2^{n/2}$ random elements in M : $m_1, \dots, m_{2^{n/2}}$
2. For $i = 1, \dots, 2^{n/2}$ 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: $O(2^{n/2})$ (space $O(2^{n/2})$)

Sample C.R. hash functions:

Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

	<u>function</u>	<u>digest size (bits)</u>	<u>Speed (MB/sec)</u>	<u>generic attack time</u>
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 finder for SHA-1 requires 2^{51} hash evaluations

Quantum Collision Finder

	Classical algorithms	Quantum algorithms
Block cipher $E: K \times X \rightarrow X$ exhaustive search	$O(K)$	$O(K ^{1/2})$
Hash function $H: M \rightarrow T$ collision finder	$O(T ^{1/2})$	$O(T ^{1/3})$

End of Segment



Collision resistance

The Merkle-Damgard
Paradigm

Collision resistance: review

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

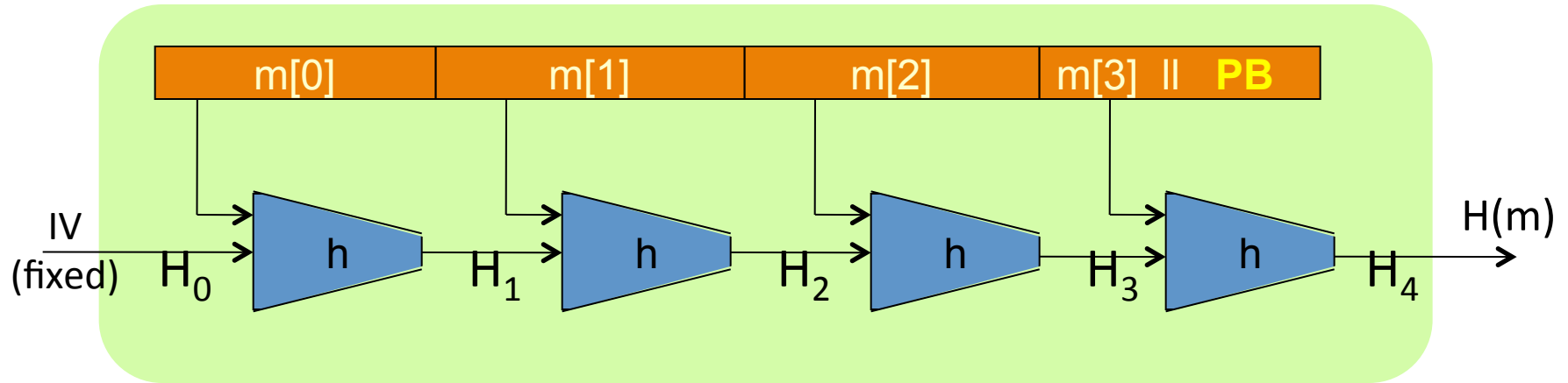
A **collision** for H is a pair $m_0, m_1 \in M$ such that:

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

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 $h: T \times X \rightarrow T$ (compression function)

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

PB: padding block

1000...0 || msg len

64 bits

If no space for PB
add another block

MD collision resistance

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

Proof: collision on $H \Rightarrow$ collision on h

Suppose $H(M) = H(M')$. We build collision for h .

$$IV = H_0, H_1, \dots, H_t, H_{t+1} = H(M)$$

$$IV = H'_0, H'_1, \dots, H'_r, H'_{r+1} = H(M')$$

$$h(H_t, M_t \parallel PB) = H_{t+1} = H'_{r+1} = h(H'_r, M'_r \parallel PB')$$

$$\text{If } \left[\begin{array}{l} H_t \neq H'_r \text{ or} \\ M_t \neq M'_r \text{ or} \\ PB \neq PB' \end{array} \right]$$

\Rightarrow we have a collision on h .

STOP

Otherwise,

Suppose $\underline{H_t = H'_r}$ and $\underline{M_t = M'_r}$ and $PB = PB'$

$\Rightarrow t = r$

Then: $\boxed{h(H_{t-1}, M_{t-1}) = H_t = H'_t = h(H'_{t-1}, M'_{t-1})}$

If $\left[\begin{array}{c} H_{t-1} \neq H'_{t-1} \\ \text{or} \\ M_{t-1} \neq M'_{t-1} \end{array} \right]$ then we have a collision on h . STOP.

otherwise, $H_{t-1} = H'_{t-1}$ and $M_t = M'_t$ and $M_{t-1} = M'_{t-1}$.

Iterate all the way to beginning and either:

$\boxed{\text{(1) find collision on } h, \text{ or}}$

$\text{(2) } \forall i: M_i = M'_i \Rightarrow M = M'$

cannot happen
because M, M'
are collision
on H .

⇒ To construct C.R. function,
suffices to construct compression function

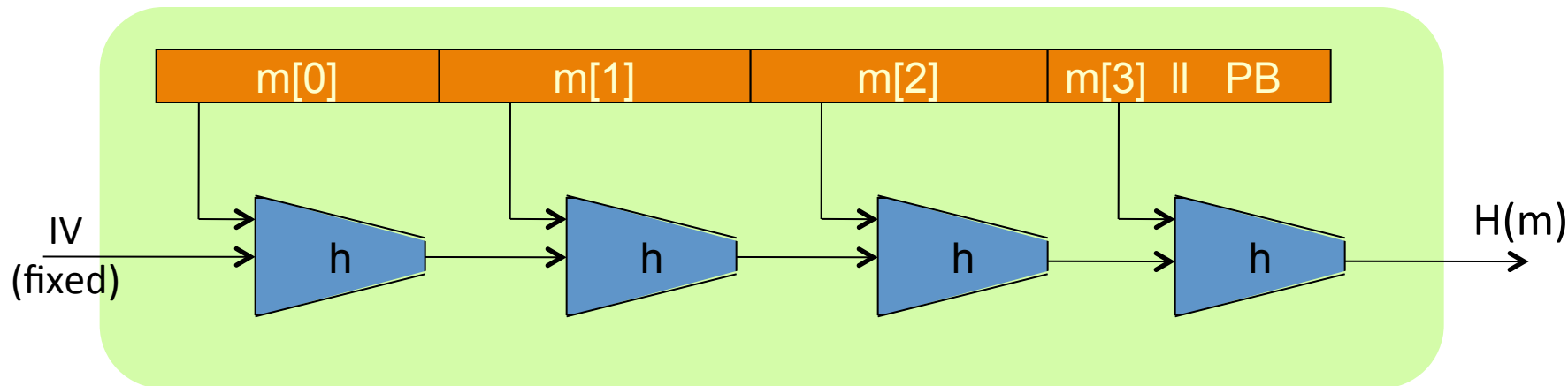
End of Segment



Collision resistance

Constructing Compression
Functions

The Merkle-Damgard iterated construction



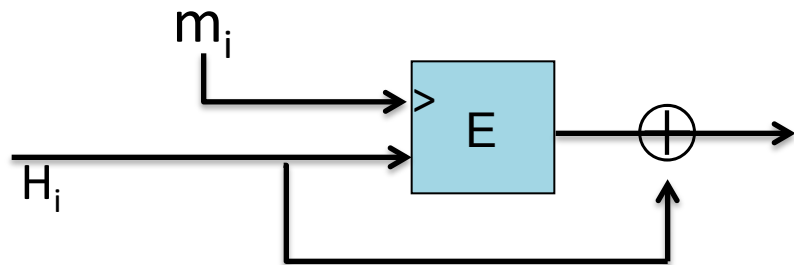
Thm: h collision resistant $\Rightarrow H$ collision resistant

Goal: construct compression function $h: T \times X \rightarrow T$

Compr. func. from a block cipher

$E: K \times \{0,1\}^n \rightarrow \{0,1\}^n$ a block cipher.

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



Thm: 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 !!

Suppose we define $h(H, m) = E(m, H)$

Then the resulting $h(.,.)$ is not collision resistant:

to build a collision (H, m) and (H', m')

choose random (H, m, m') and construct H' as follows:

- ☐ $H' = D(m', E(m, H)) \iff E(m', H') = E(m, H)$
- ☐ $H' = E(m', D(m, H))$
- ☐ $H' = E(m', E(m, H))$
- ☐ $H' = D(m', D(m, H))$

Other block cipher constructions

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

Miyaguchi-Preneel:

$$h(H, m) = E(m, H) \oplus H \oplus m$$

(Whirlpool)

$$h(H, m) = E(H \oplus m, m) \oplus m$$

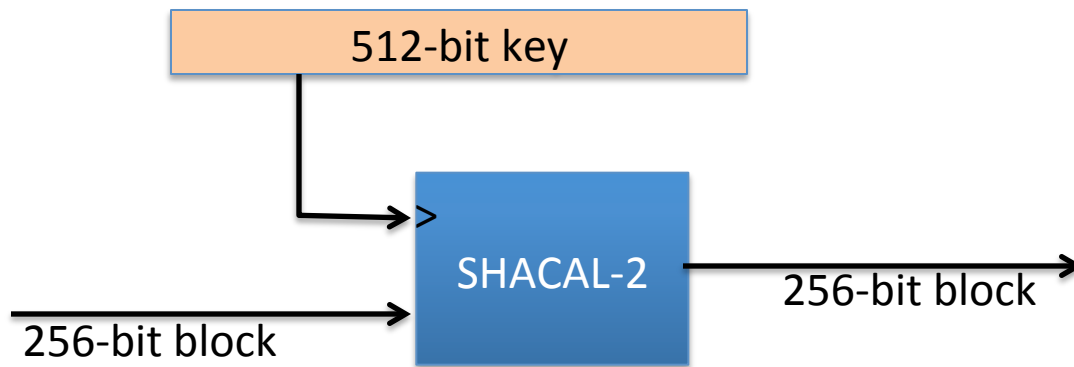
total of 12 variants like this

Other natural variants are insecure:

$$h(H, m) = E(m, H) \oplus m \quad (\text{HW})$$

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 $m, h \in \{0, \dots, p-1\}$ define $h(H, m) = u^H \cdot v^m \pmod{p}$

Fact: finding collision for $h(.,.)$ is as hard as solving “discrete-log” modulo p .

Problem: slow.

End of Segment

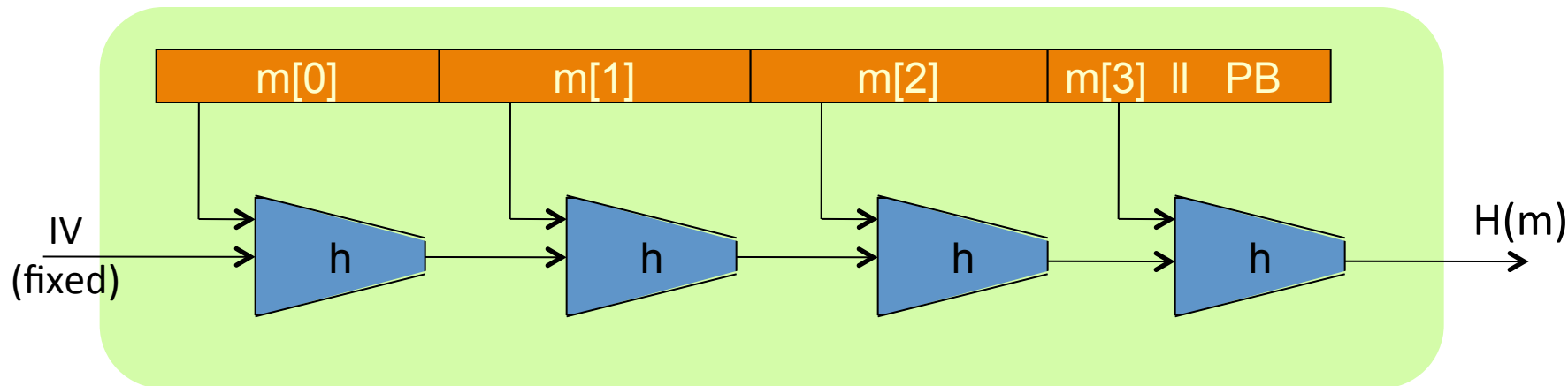


Collision resistance

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

Attempt #1: $S(k, m) = H(k \parallel m)$

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.

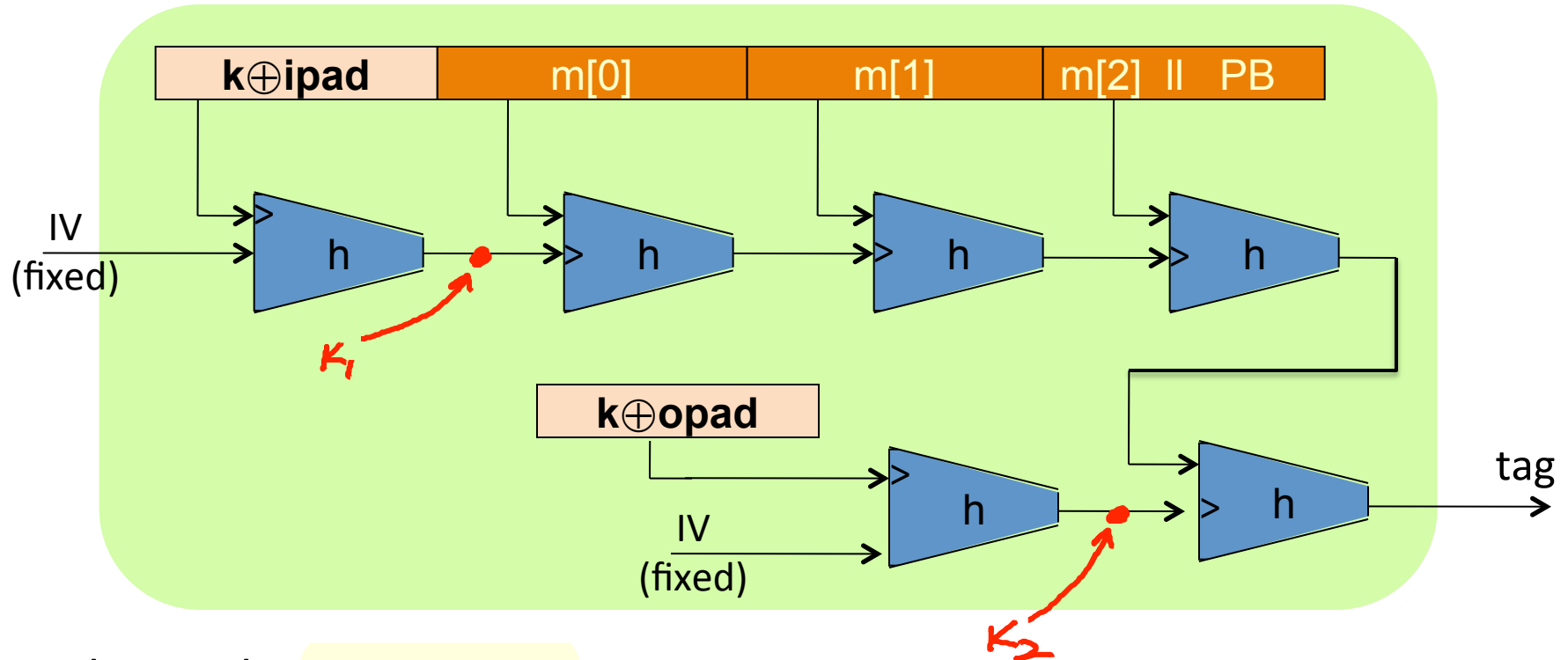
H: hash function.

example: SHA-256 ; output is 256 bits

Building a MAC out of a hash function:

$$\text{HMAC: } S(k, m) = H(k \oplus \text{opad} \parallel H(k \oplus \text{ipad} \parallel m))$$

HMAC in pictures



Similar to the NMAC PRF.

main difference: the two keys k_1, k_2 are dependent

HMAC properties

Built from a black-box implementation of SHA-256.

HMAC is assumed to be a secure PRF

- Can be proven under certain PRF assumptions about $h(.,.)$
- Security bounds similar to NMAC
 - Need $q^2/|T|$ to be negligible ($q \ll |T|^{1/2}$)

In TLS: must support HMAC-SHA1-96

End of Segment



Collision resistance

Timing attacks on MAC
verification

Warning: verification timing attacks [L'09]

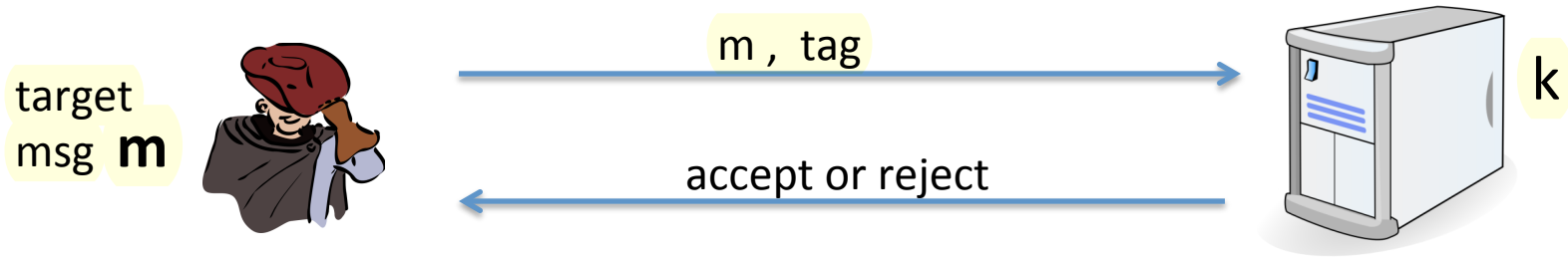
Example: Keyczar crypto library (Python) [simplified]

```
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 !

End of Segment