CIS 5560

Cryptography
Lecture 10

Course website:

pratyushmishra.com/classes/cis-5560-s24/

Slides adapted from Dan Boneh and Vinod Vaikuntanathan

Announcements

- **HW 5 out after lecture**
    - Due **Tuesday**, Feb 27 at 1PM on Gradescope
    - Covers MACs,

Recap of last lecture

Message Authentication Codes (MACs)

A triple of algorithms (Gen, MAC, Ver):

- Gen(1^n): Produces a key $k \leftarrow \mathcal{K}$.
- MAC(k, m): Outputs a tag t (may be deterministic).
- Ver(k, m, t): Outputs Accept or Reject.

Correctness: $\Pr[\mathrm{Ver}(k, m, \mathrm{MAC}(k, m)) = 1] = 1$

Security: *Hard to forge.* Intuitively, it should be hard to come up with a new pair *(m', t')* such that Ver accepts.

EUF-CMA Security

Existentially Unforgeable against Chosen Message Attacks

$k \leftarrow K$

$\xrightarrow{m_1}$

$\xleftarrow{t_1 = \text{MAC}(k, m_1)}$

$\xrightarrow{m_2}$

$\xleftarrow{t_2 = \text{MAC}(k, m_2)}$

$\vdots$

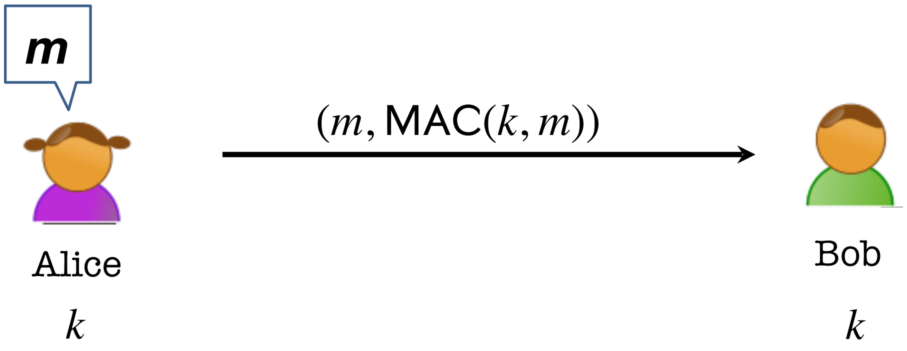
$\xrightarrow{(m,\ t)}$

Accept if $(m,\ t) \neq (m_i,\ t_i)$
for all i, and
Ver$(k, m, t) = 1$

Want: $\Pr((m,\ t) \leftarrow A^{MAC(k,\ \bullet)}(1^n),\ Ver(k,\ m,\ t) = 1,\ (m,\ t) \notin Q)) = negl(n)$.

where Q is the set of queries $\left\{ \left(m_i,\ t_i\right) \right\}_i$ that A makes.

Constructing a MAC



Gen(1^n): Produces a PRF key $k \leftarrow K$.
MAC(k, m): Output $f_k(m)$.
Ver(k, m, t): Accept if $f_k(m) = t$, reject otherwise.

Security: Our earlier unpredictability lemma about PRFs essentially proves that this is secure!

A Simple Lemma about Unpredictability

Let **F: K×X$\longrightarrow$Y** be a pseudorandom function.

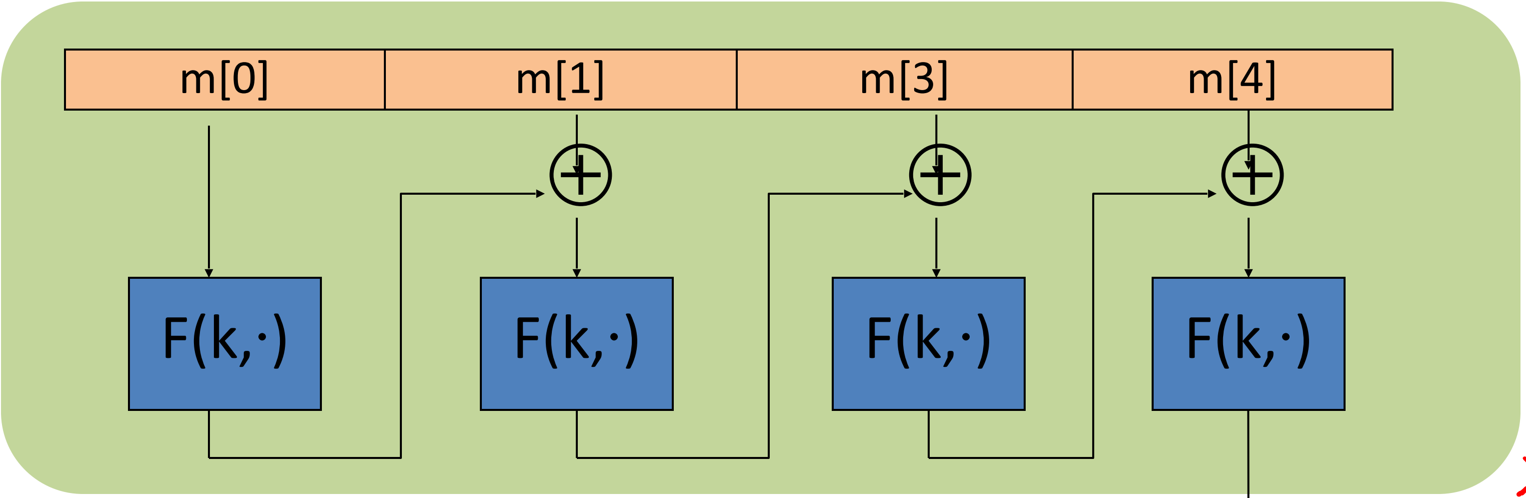
- Consider an adversary who requests and obtains $\mathsf{F}_k(x_1), \ldots, \mathsf{F}_k(x_q)$ for a polynomial $q = q(n)$.

- Can she predict $\mathsf{F}_k(x^\star)$ for some x^* of her choosing where $x^* \notin \{x_1, \ldots, x_q\}$? How well can she do it?

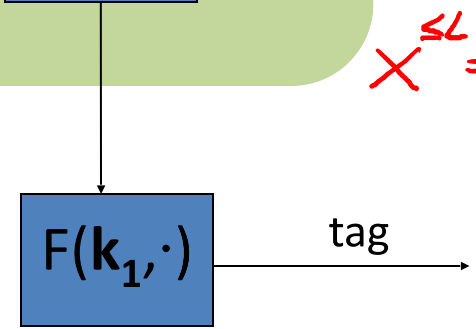
Lemma: If she succeeds with probability $\frac{1}{2^m} + 1/\mathrm{poly}(n)$, then she broke PRF security.

Construction: encrypted CBC-MAC

raw CBC



$X^{\leq L} = \bigcup_{i=1}^{L} X^i$

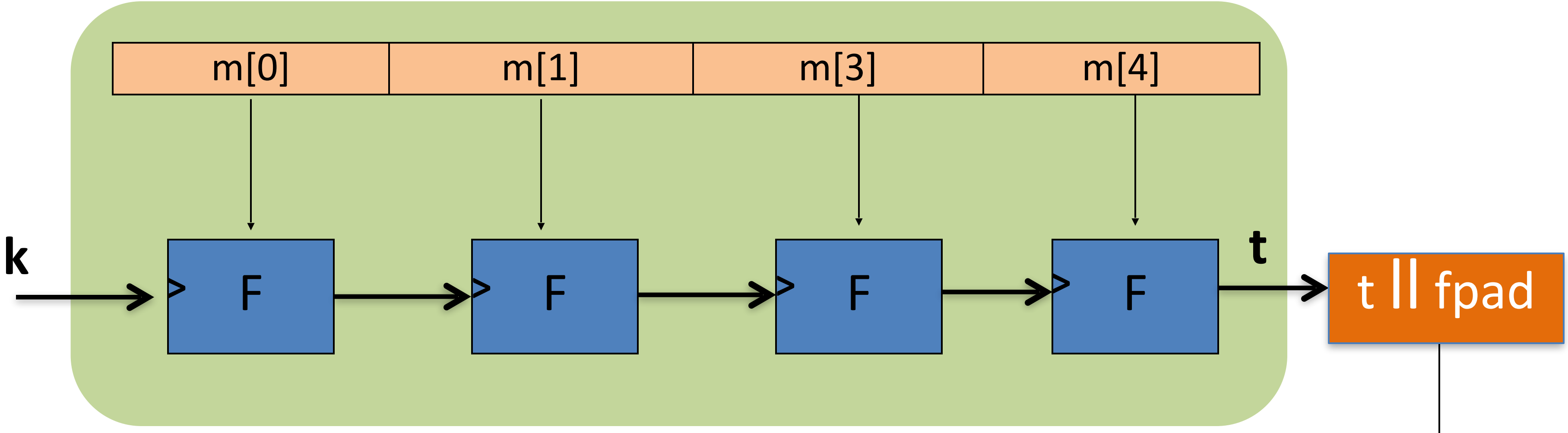


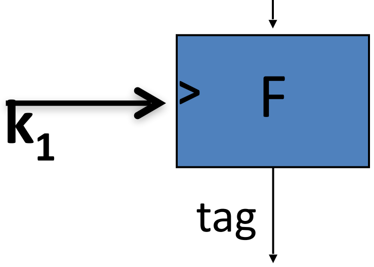
Let **F: K × X ⟶ X** be a PRP

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

Construction: NMAC (nested MAC)

cascade





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

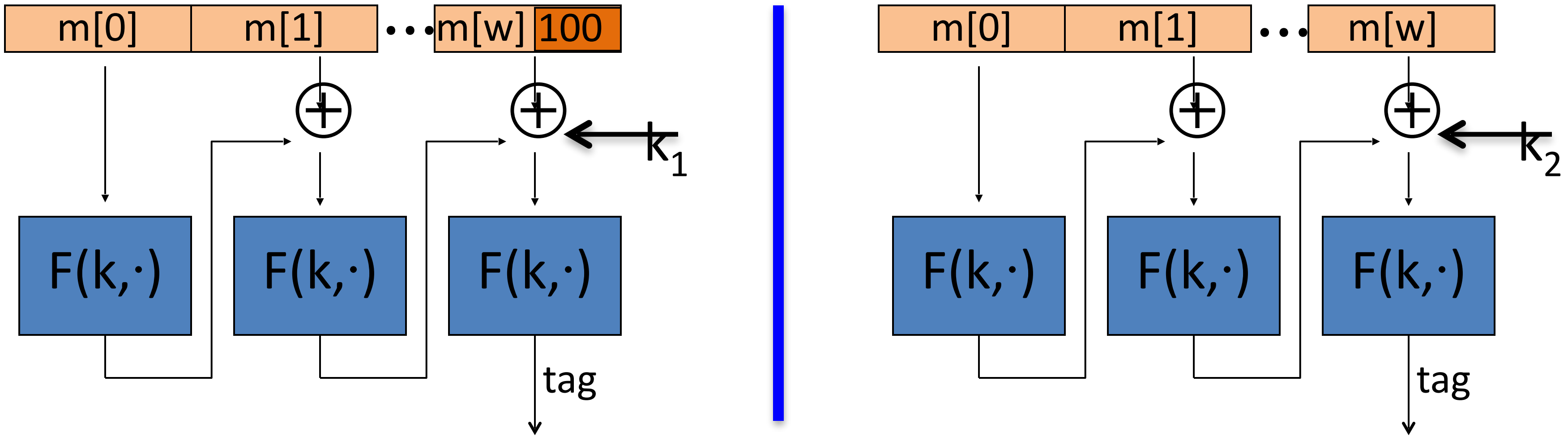
Define new PRF $\mathbf{F_{NMAC} : K^2 \times X^{\leq L} \longrightarrow K}$

CMAC (NIST standard)

(k_1, k_2) derived from k

Variant of CBC-MAC where 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)



Today's Lecture

- Collision-resistant Hash Functions (CRHFs)
- CRH → MACs
  - HMAC

Collision Resistance

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

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 efficient algs. A:

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

is "neg".

Example: SHA-256 (outputs 256 bits)

Formal Definition: Collision-Resistant Hash Functions

A compressing **family of functions** $\mathscr{H} = \{h : \{0,1\}^m \to \{0,1\}^n\}$ (where $m > n$) for which it is computationally hard to find collisions.

Def: $\mathscr{H}$ is collision-resistant if for every PPT algorithm A, there is a negligible function μ s.t.

$$\Pr_{h \leftarrow \mathscr{H}}\left[A\left(1^n, h\right) = \left(x, y\right) : x \neq y,\ h(x) = h\left(y\right)\right] = \mu(n)$$

MACs from Collision Resistance

Let MAC be a MAC for short messages over (K,M,T)   (e.g. AES)

Let H: $M^{big} \to M$ be a hash function

Def: $MAC^{big} = (MAC^{big}, Ver^{big})$ over (K, M^{big}, T) as:

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

<u>Thm</u>: If MAC is a secure MAC and H is collision resistant

then MAC^{big} is a secure MAC.

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

MACs from Collision Resistance

$\mathrm{MAC}^{big}(k, m) = \mathrm{MAC}(k, H(m))$;

$\mathrm{Ver}^{big}(k, m, t) = V(k, H(m), t)$

Collision resistance is necessary for security:

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

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

step 1: adversary asks for $t \leftarrow \mathrm{MAC}(k, m_0)$

step 2: output (m_1, t) as forgery

Collision resistance

Generic birthday attack

Generic attack on C.R. functions

Let H: M $\longrightarrow$ $\{0,1\}^n$ 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: $m_1, \ldots, m_{2^{n/2}}$ (distinct w.h.p)
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.

How well will this work?

The birthday paradox

Let $r_1, \ldots, r_n \in \{1, \ldots, B\}$ be IID integers.

Thm: When $n \approx \sqrt{B}$ then $\Pr[r_i = r_j \mid \exists i \neq j] \geq \frac{1}{2}$

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

$$\Pr[\exists i \neq j : r_i = r_j] = 1 - \Pr[\forall i \neq j : r_i \neq r_j] = 1 - \left(\frac{B-1}{B}\right)\left(\frac{B-2}{B}\right)\cdots\left(\frac{B-n+1}{B}\right) =$$

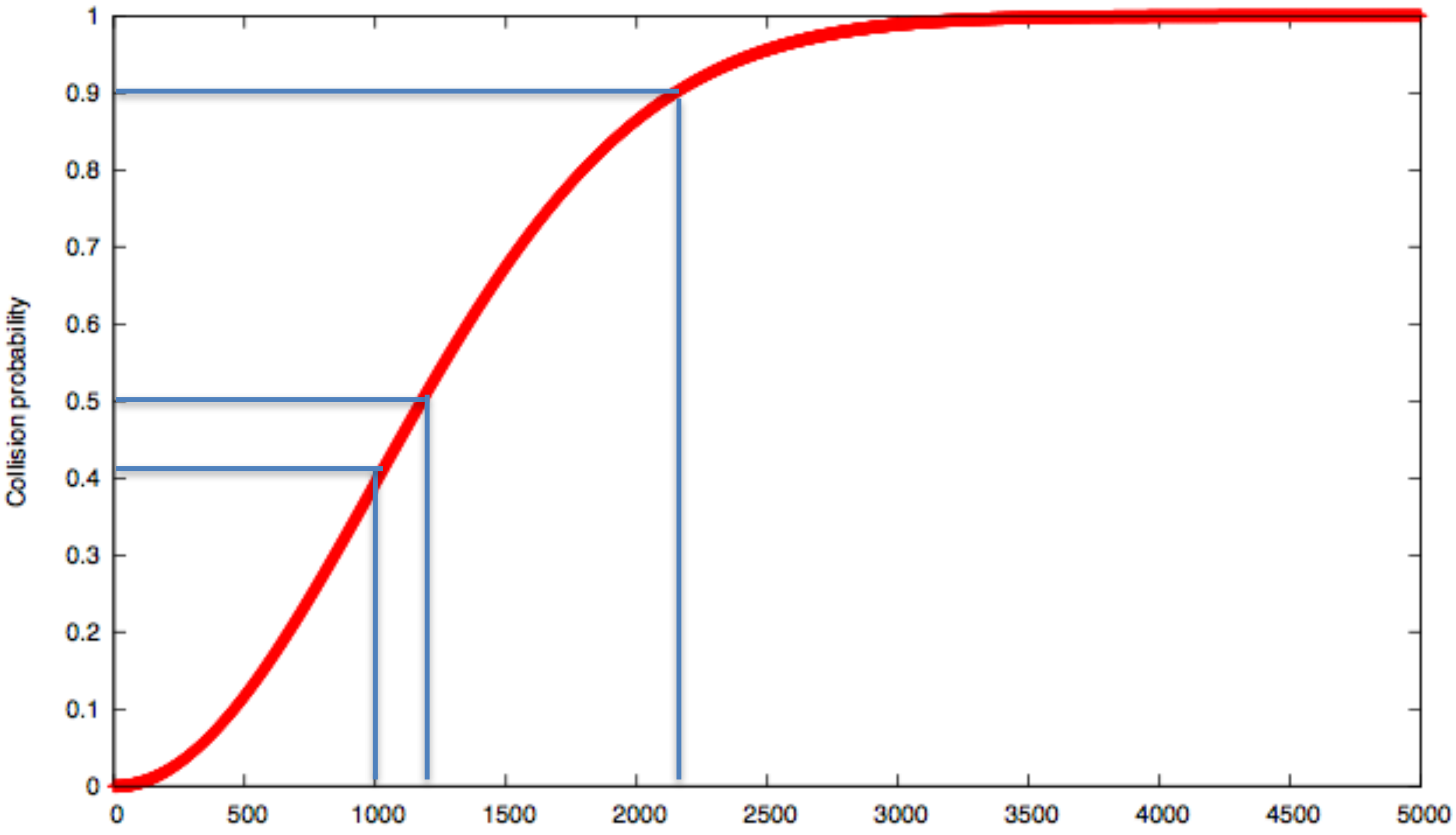
$$= 1 - \prod_{i=1}^{n-1}\left(1 - \frac{i}{B}\right) \geq 1 - \prod_{i=1}^{n-1} e^{-i/B} = 1 - e^{-\frac{1}{B}\sum_{i=1}^{n-1} i} \geq 1 - e^{-n^2/2B}$$

$1 - x \leq e^{-x}$

$$\geq 1 - e^{-0.72} = 0.53 > \frac{1}{2}$$

$\frac{n^2}{2B} = 0.72$

B=10^6



samples n

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 $t_i = H(m_i) \quad \in \{0,1\}^n$
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:

Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

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

* SHA-1 is broken; do not use!

The Merkle-Damgard Paradigm:

Collision resistance: review

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

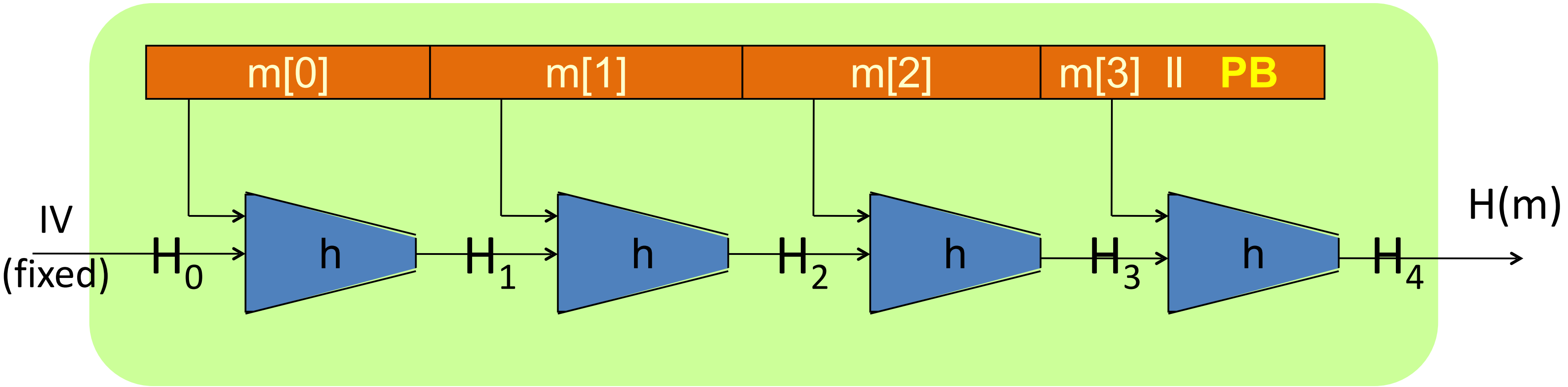
A **collision** for H is a pair $m_0, m_1 \in M$ such that:

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

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 × X ⟶ T** (compression function)

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

PB: padding block

If no space for PB add another block

Proof on Board

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.

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

$$IV = H_0', \quad H_1', \quad \ldots, \quad H'_r, \quad H'_{r+1} = H(M')$$

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

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 $H_t = H'_r$ and $M_t = M'_r$ and $PB = PB'$

$\Rightarrow t = r$

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

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:

(1) find collision on h, or

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

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

$\Rightarrow$ To construct C.R. function,

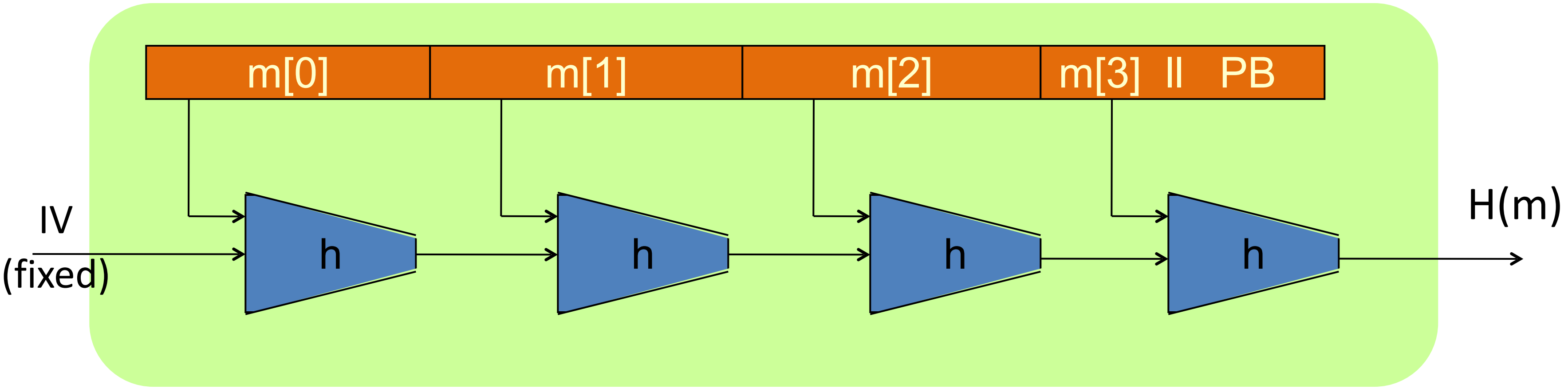
suffices to construct compression function

End of Segment

Collision resistance

Constructing Compression Functions

The Merkle-Damgard iterated construction



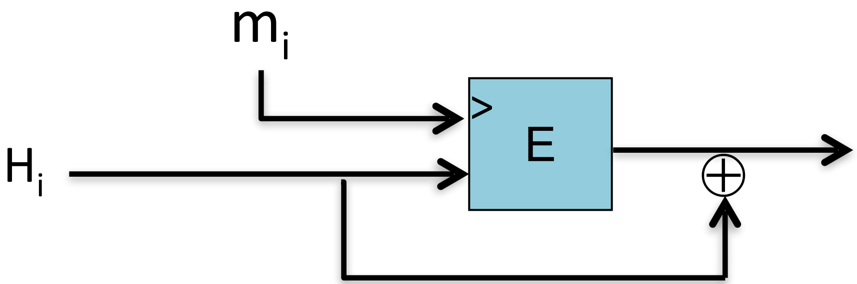
Thm: h collision resistant ⇒ H collision resistant

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

Compr. func. from a block cipher

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

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



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:

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

Other block cipher constructions

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

Miyaguchi-Preneel: **$h(H, m) = E(m, H) \oplus H \oplus m$** **(Whirlpool)**

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

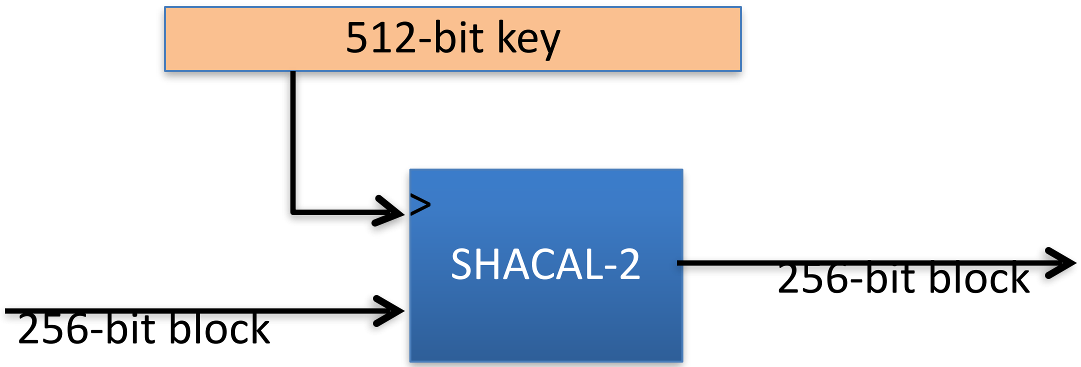
total of 12 variants like this

Other natural variants are insecure:

$h(H, m) = E(m, H) \oplus m$ (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 $1 \le u, v \le p$.

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

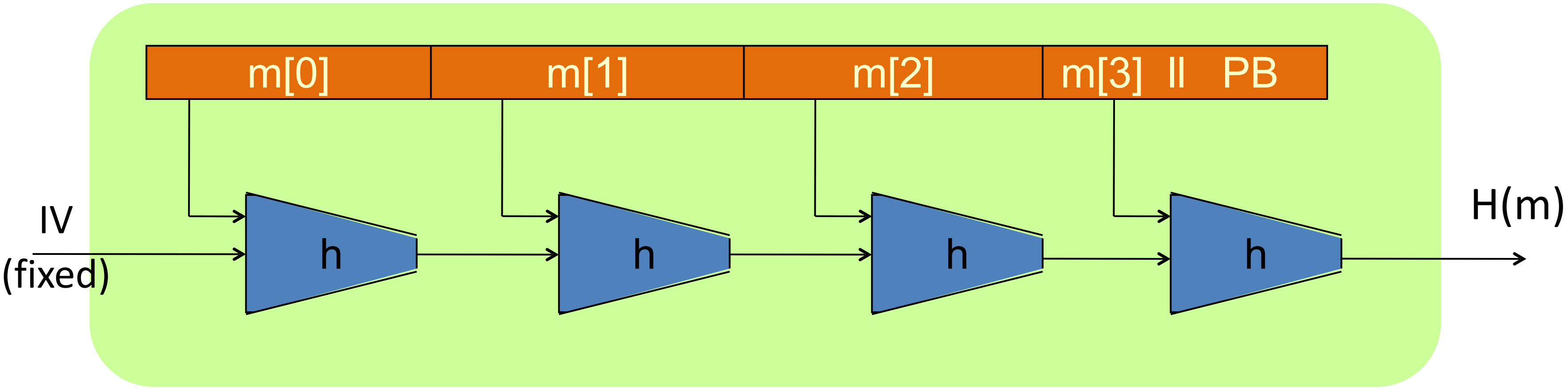
Fact: finding collision for h(.,.) is as hard as solving "discrete-log" modulo p.

Problem: slow.

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

$H: X^{\leq L} \longrightarrow T$ 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.

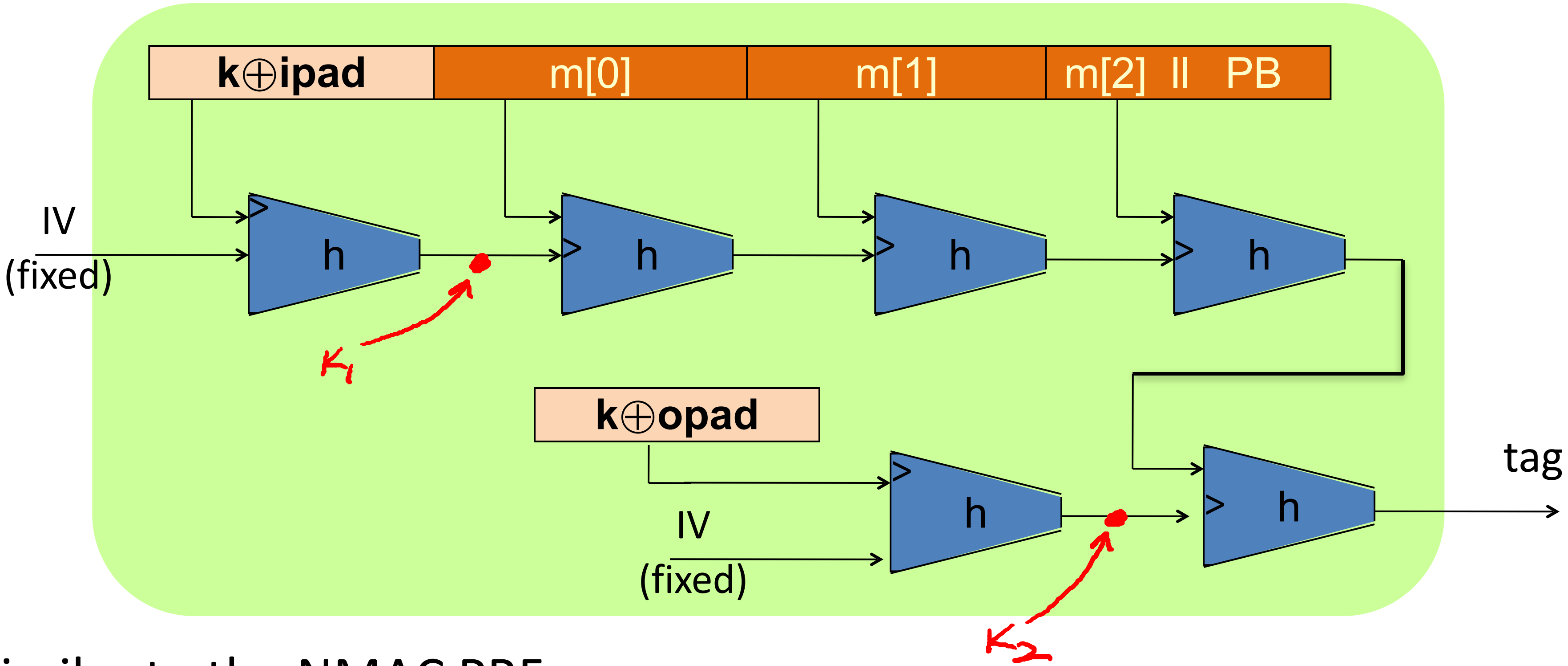
Standardized method: HMAC (Hash-MAC)

Most widely used MAC on the Internet.

Building a MAC out of a hash function H:

HMAC: $\mathsf{MAC}(k, m) = H(k \oplus \mathsf{opad} \mid\mid H(k \oplus \mathsf{ipad} \mid\mid m))$

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 ($q << |T|^{½}$)

In TLS: must support HMAC-SHA1-96

Timing attacks on MAC verification

Warning: verification timing attacks [L'09]

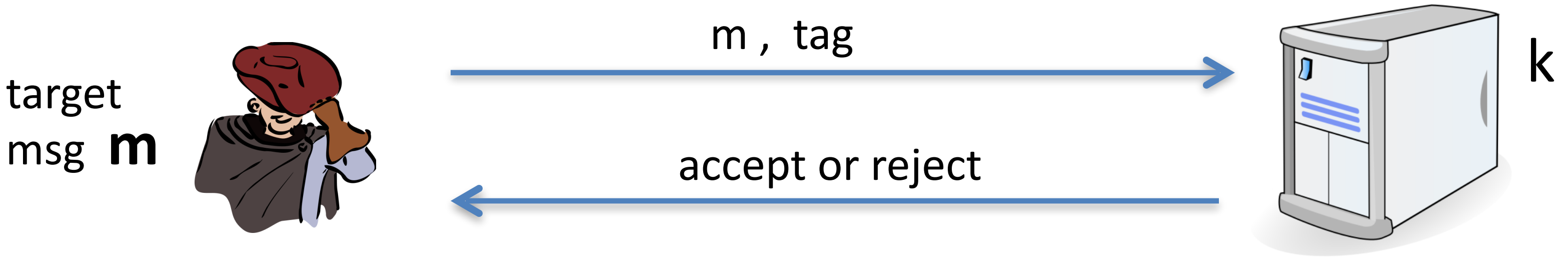
Example: Keyczar crypto library (Python) [simplified]

```
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

3	53	*	*	*	*

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 !