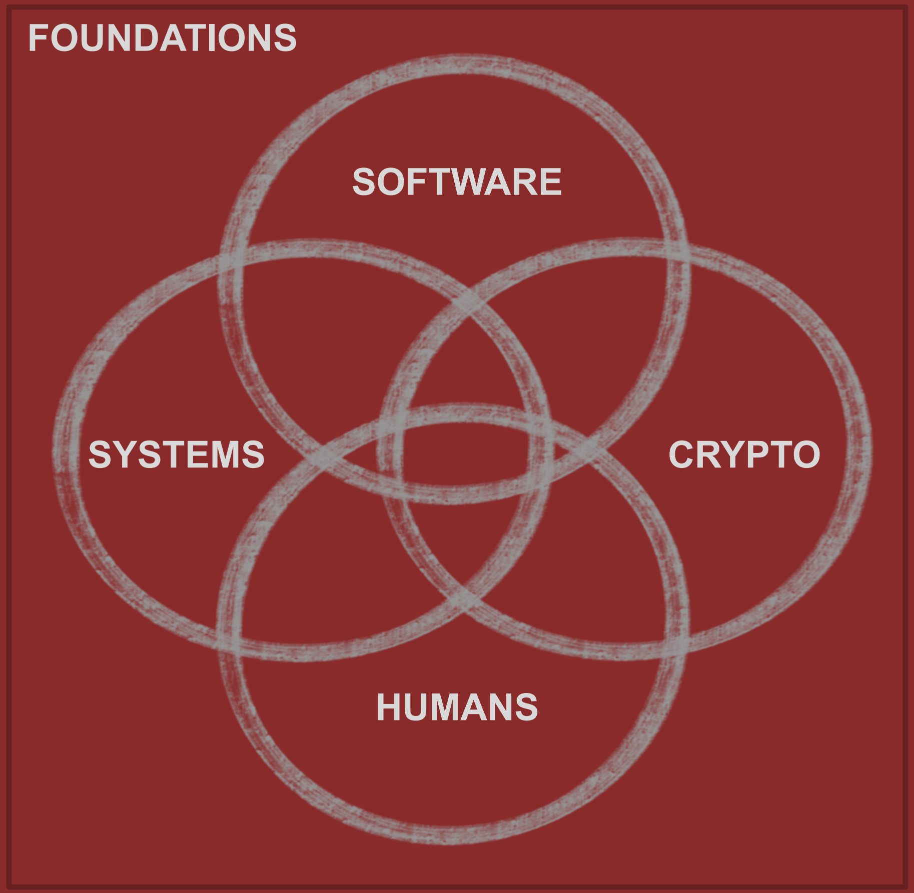
Διάλεξη #16 - Hash Functions

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στην Ασφάλεια

Θανάσης Αυγερινός



Huge thank you to David Brumley from Carnegie Mellon University for the guidance and content input while developing this class (lots of slides from Dan Boneh @ Stanford and some from Adrian Perrig)

Ανακοινώσεις / Διευκρινίσεις

- Η εργασία #2 κλείνει αύριο, μην ξεχάσουμε το write up!
- Η εργασία #3 θα ανοίξει αυτήν την εβδομάδα
- Η καταγραφή της Παρασκευής δεν πέτυχε :(- θα κάνουμε αναφορές
    - Δείτε την διάλεξη του Dan Boneh (Week 3)
- Οι βαθμοί των εργασιών θα ανακοινωθούν στις 16 Ιουνίου
- Το τελικό διαγώνισμα θα είναι στις 28 Ιουνίου

Ερωτήσεις:

1. Γιατί είναι το μήνυμα μέρος του MAC?
2. Παράδειγμα όπου μια συνάρτηση είναι second pre-image resistant αλλά όχι strongly collision resistant?

Την προηγούμενη φορά

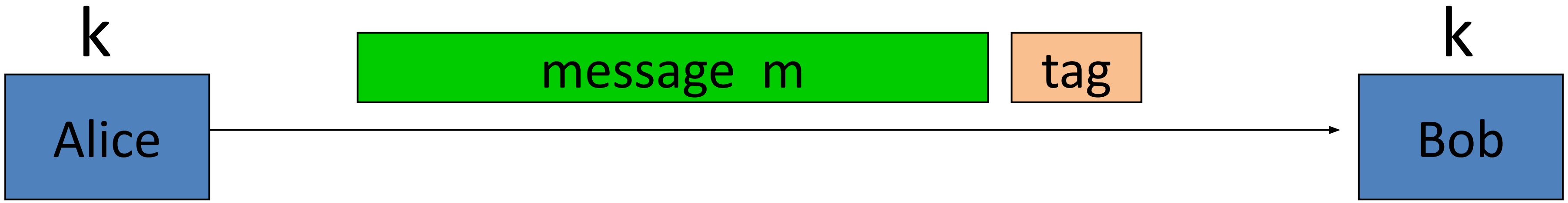
- Message Integrity
  - Message Authentication Codes (MACs)
  - CBC-MAC, NMAC, CMAC
- Introduction to Hashing

Σήμερα

- Hashes Intro
- Hash Constructions
- HMAC
- Hash Tricks/Datastructures
- Authenticated Encryption (AuthEnc)

Integrity Reminder

Message integrity: MACs



Generate tag (Sign):

tag ← S(k, m)

Verify tag:

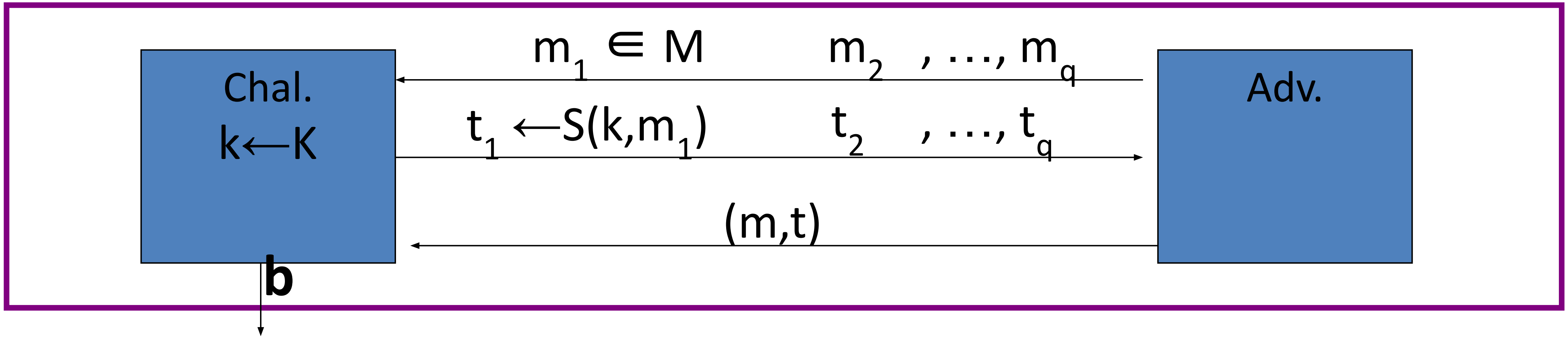
V(k, m, tag) $\stackrel{?}{=}$ `yes'

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'

Secure MACs

- For a MAC $I=(S,V)$ and adv. A define a MAC game as:

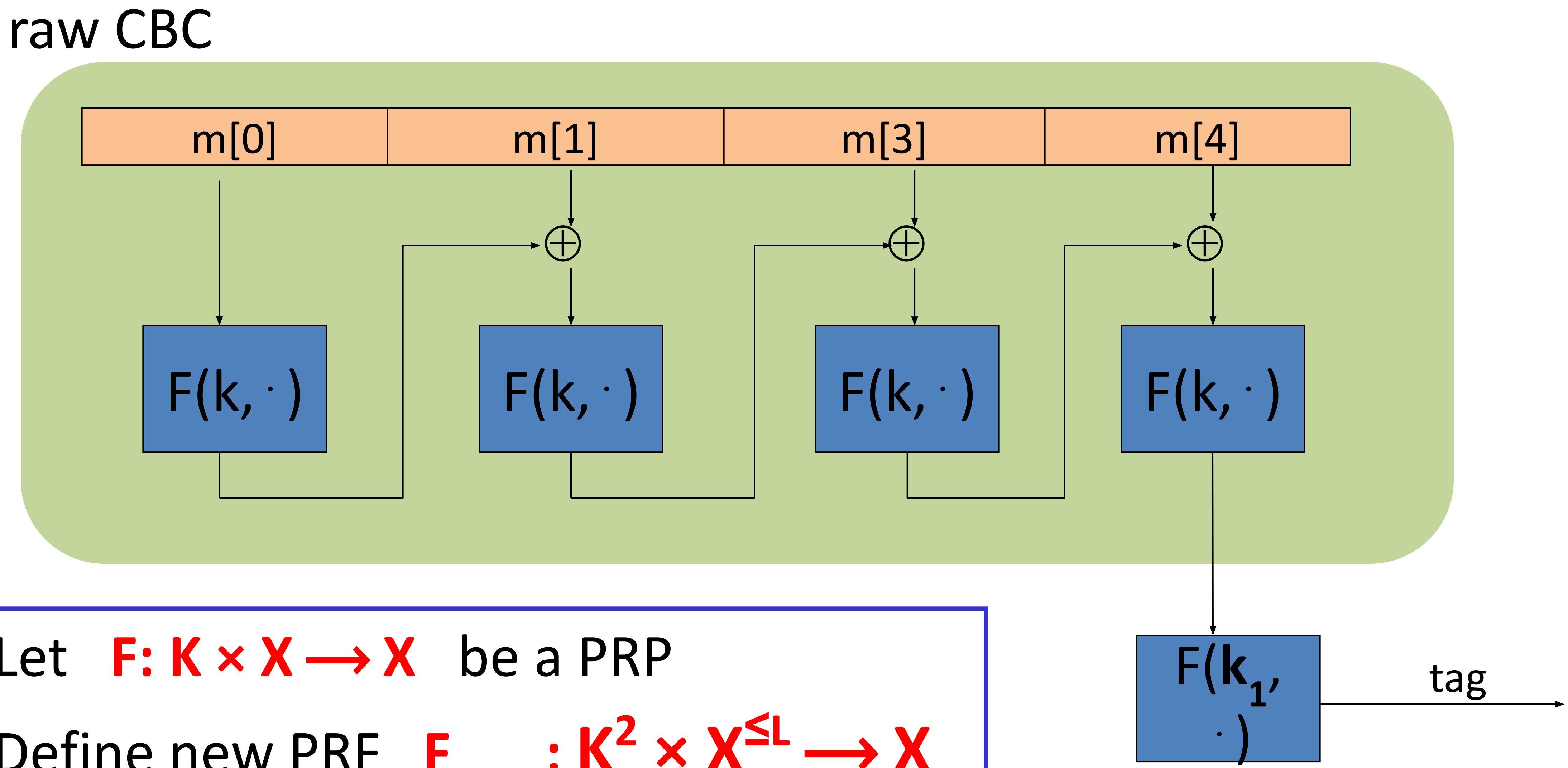


b=1 if $V(k,m,t) =$ `yes' and $(m,t) \notin \{ (m_1,t_1) , \ldots , (m_q,t_q) \}$
b=0 otherwise

Def: $I=(S,V)$ is a **secure MAC** if for all "efficient" A:

$$\text{Adv}_{\text{MAC}}[A,I] = \Pr[\text{Chal. outputs } 1] \quad \text{is "negligible."}$$

Construction 1: encrypted CBC-MAC

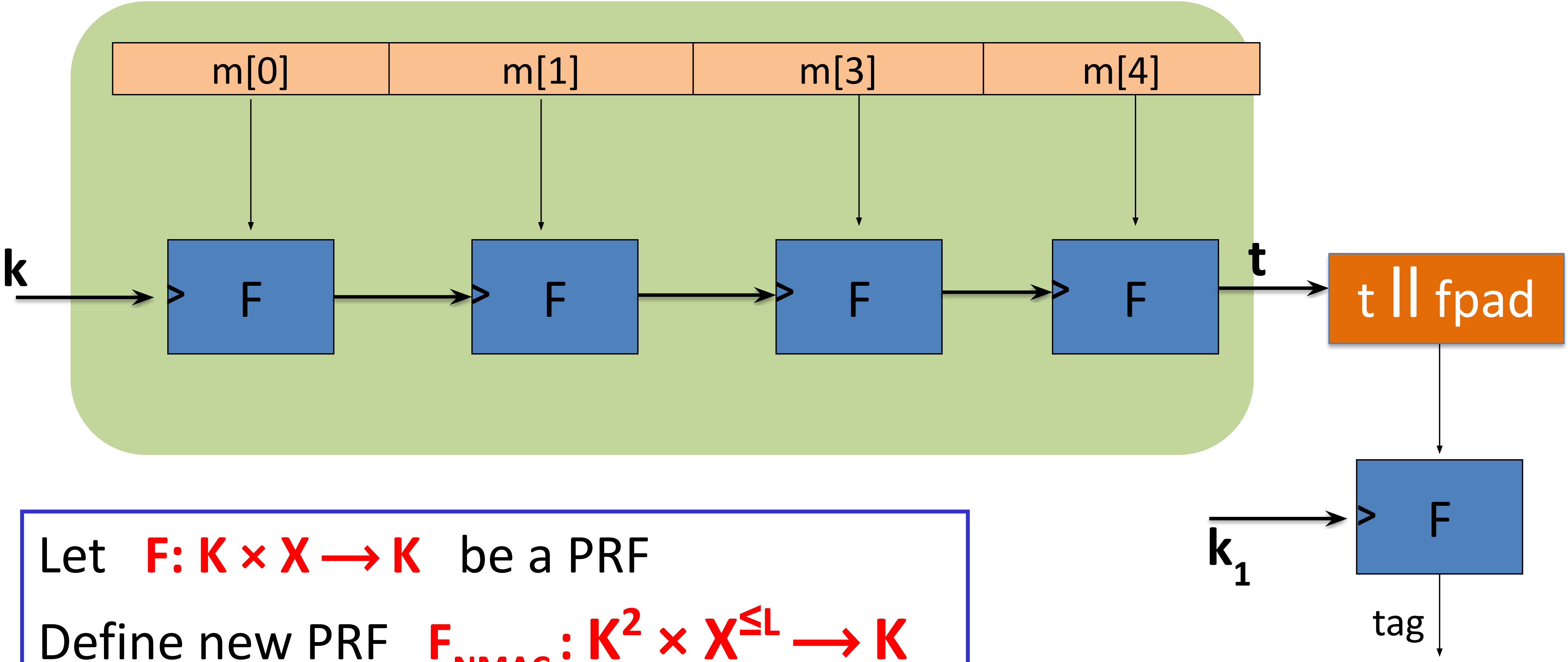


Let **F: K × X ⟶ X** be a PRP

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

Construction 2: NMAC (nested MAC)

cascade



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

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

Quiz Question

Why get the message included in the MAC computation? Let's use MAC = $E(k_1, k_2)$ and it is clearly not invertible or forgeable.

Hashes and Resistance

Cryptographic Hash Functions

A Cryptographic Hash Function (CHF) is an algorithm that maps an arbitrary binary string to a string of n bits. $H : \{0, 1\}^* \rightarrow \{0,1\}^n$

- Message space much larger than output space

$$H: M \rightarrow T, |M| >> |T|$$

- Given the output, we want the input to remain secret and also make it hard for other inputs to get the same output (collision).
- Applications: everywhere (from storing passwords,

Hash Function Properties

Let H: M -> T, |M| >> |T|

- Pre-image resistance. H is pre-image resistant if given a hash value h, it should be difficult to find any message m such that H(m) = h. In other words, P[H(random m) = h] = 1/|T|.
- Second pre-image resistance (weak collision resistance). H is second-preimage resistant if given a message m_1, it should be difficult to find a different m_2 such that $H(m_1) = H(m_2)$.
- (Strong) Collision resistance. H is collision resistant if it is difficult to find any two different messages m_1 and m_2 such that $H(m_1) = H(m_2)$.

Collision Resistance =>
Second-preimage Resistance

Second-preimage Resistance => Preimage Resistance?

*only true under certain conditions (|M| >> |T|)

Collision Resistance Definition

Let $H: M \to T$ 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:

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

is "neg".

Example: SHA-256 (outputs 256 bits)

Generic attack on C.R. functions

Let H: M $\longrightarrow$ $\{0,1\}^n$ 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, …, $2^{n/2}$ 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?

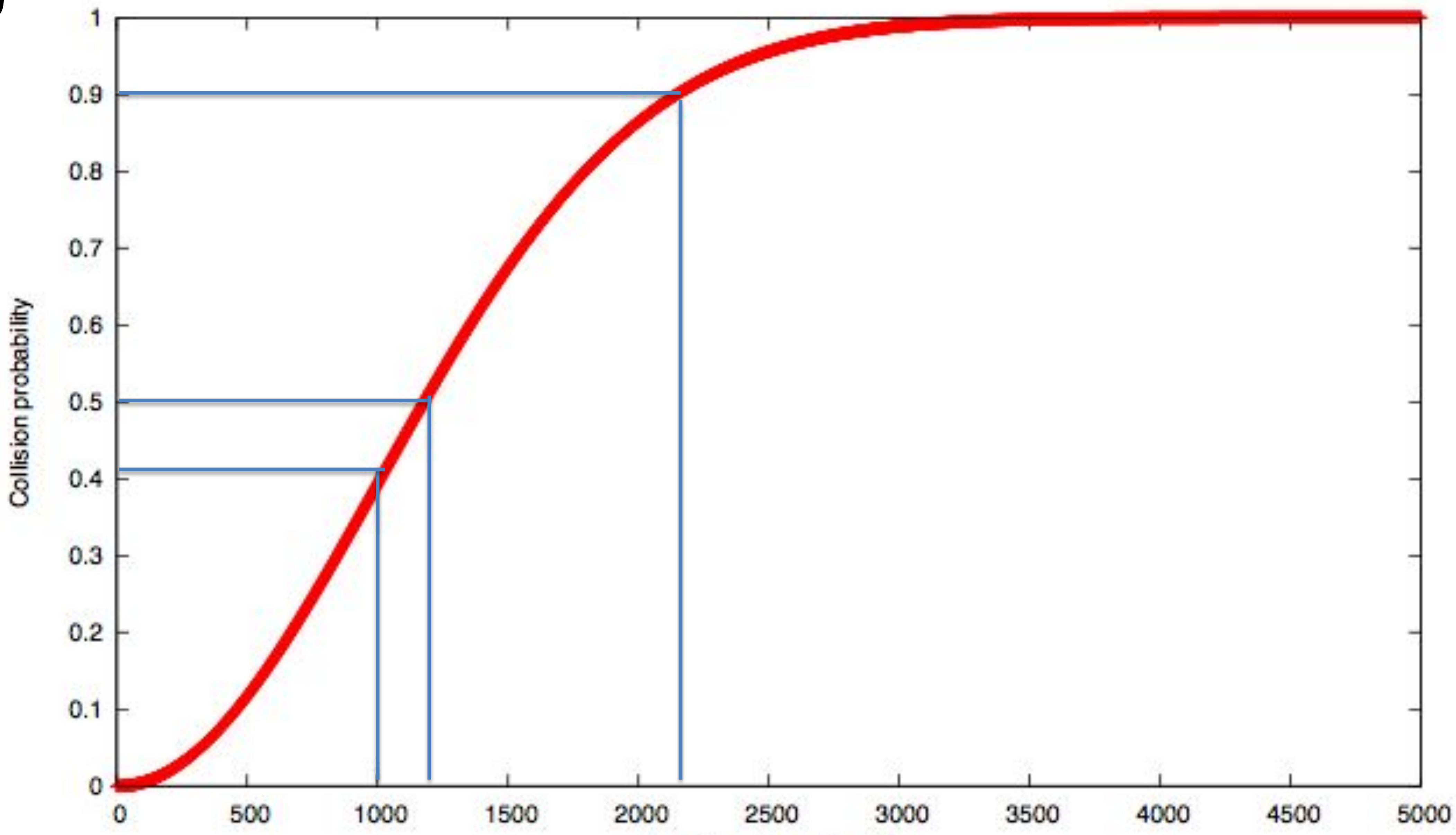
The birthday paradox

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

<u>Thm</u>: when $\mathbf{n} = 1.2 \times \mathbf{B^{1/2}}$ then $\Pr[\ \exists\, i \neq j:\ r_i = r_j\ ] \geq ½$

Proof: (for <u>uniform</u> indep. $r_1, \ldots, r_n$)

$B=10^6$



samples n

Generic attack

$H: M \longrightarrow \{0,1\}^n$.   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: 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}

* best known collision finder for SHA-1 requires 2^{51} hash evaluations

https://shattered.io/

Quantum Collision Finder

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

Hash Constructions: Merkle-Damgård

Collision resistance

Let H: M ⟶T be a hash function (|M| >> |T|)

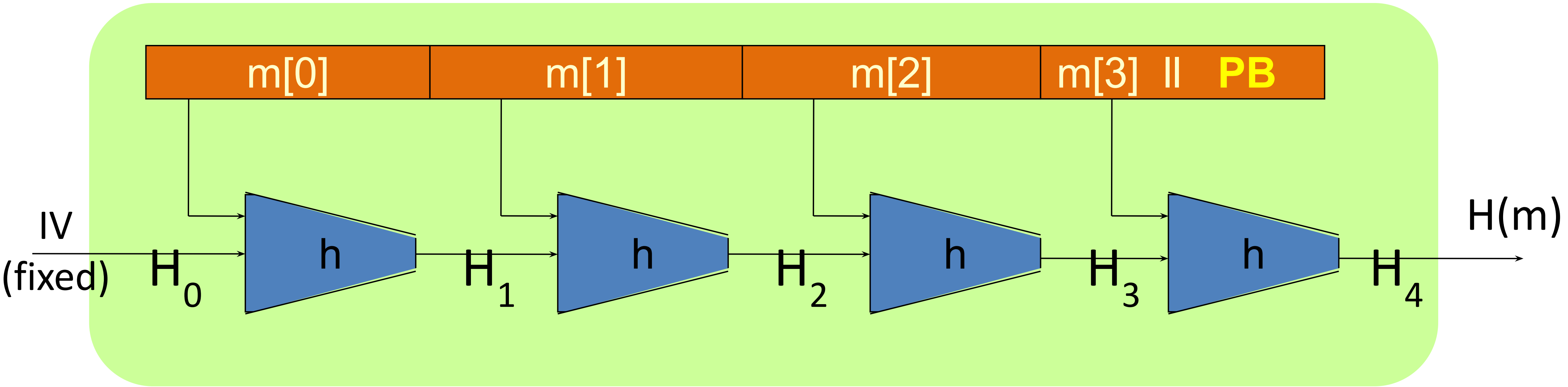
A **collision** for H is a pair m_0 , m_1 ∈ M such that:

$H(m_0) = H(m_1)$ **and** $m_0 \neq m_1$

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

PB: padding block

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.

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

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

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

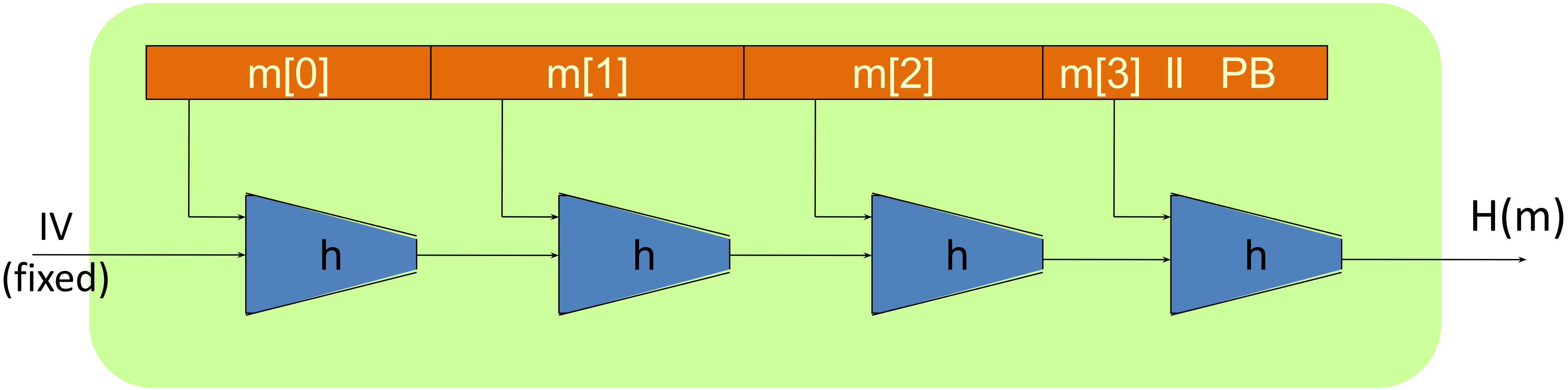
Suppose $H_t = H'_r$ and $M_t = M'_r$ and $PB = PB'$

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

⇒ To construct C.R. function,

suffices to construct compression function

The Merkle-Damgard iterated construction



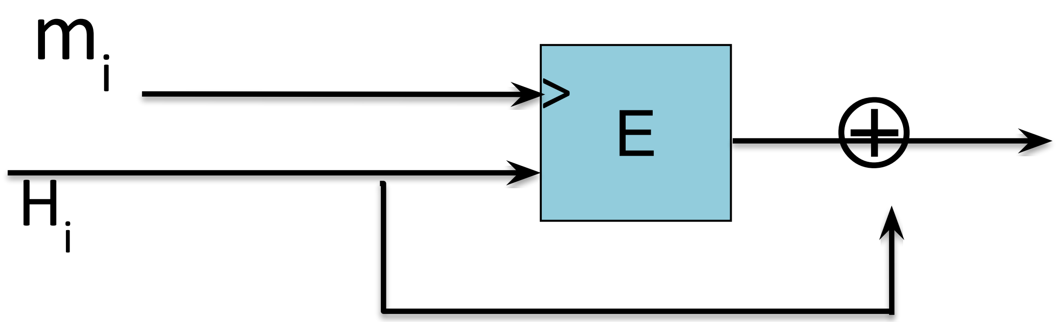
Thm: h collision resistant ⇒ H collision resistant

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

Compr. func. from a block cipher

E: K× {0,1}^{n} ⟶ {0,1}^{n} a block cipher.

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

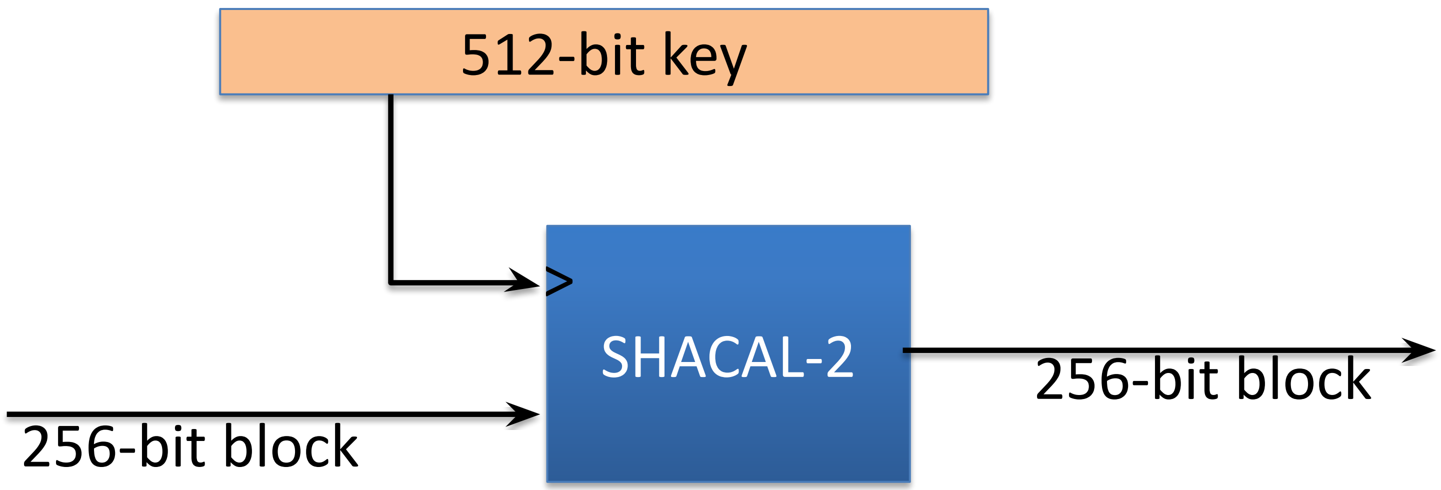
to build a collision (H,m) and (H',m')

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

- ○ H'=D(m', E(m,H))
- ○ H'=E(m', D(m,H))
- ○ H'=E(m', E(m,H))
- ○ H'=D(m', D(m,H))

Case study: SHA-256

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



Hashes and Passwords

Hash Functions are typically ***Fast***

> 10^6 / s on modern hardware

Some Hash Functions Are Slow

PBKDF2 is ~5 orders of magnitude (100,000x) slower than a standard hash function (e.g., MD5). It is also the main recommendation for storing passwords (RFC 8018 / 2017).

1. Why is a hash function used for storing passwords?

2. Is slowness an advantage or disadvantage?

Careful with storing passwords

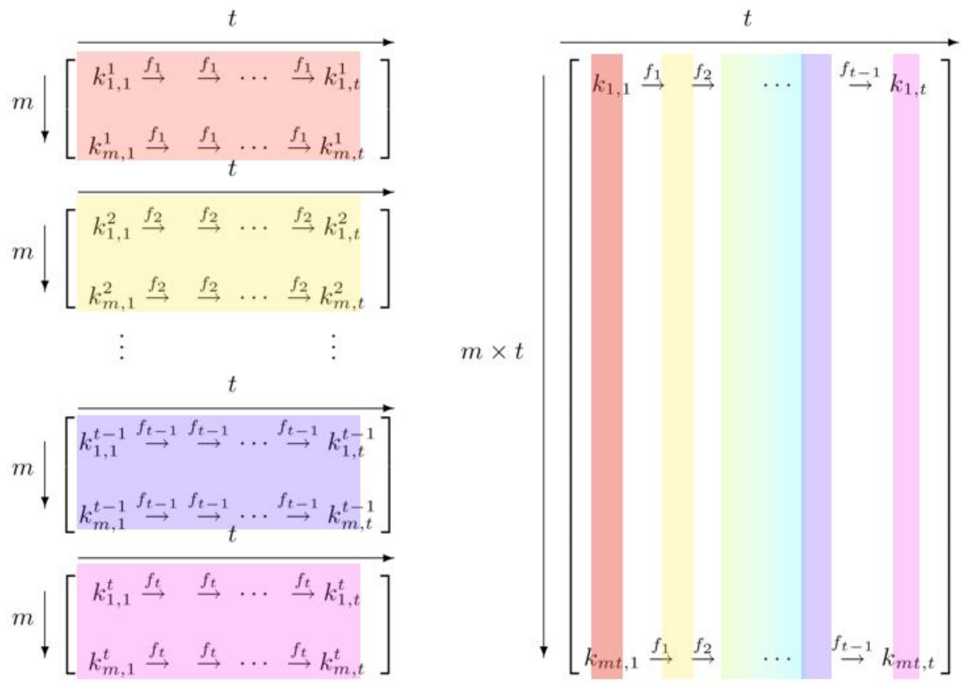
https://hashcat.net/hashcat/

COMPONENT	PERCENTILE RANK	# COMPATIBLE PUBLIC RESULTS	H/S (AVERAGE)
NVIDIA GeForce RTX 4090	96th	28	152416197859 +/- 6710203881
MSI NVIDIA GeForce RTX 4090	96th	5	151733333333 +/- 6137634362
Zotac NVIDIA GeForce RTX 2080 Ti	90th	3	130494289583 +/- 210426329
NVIDIA GeForce RTX 4080	87th	14	94912651871 +/- 2409779704
Gigabyte NVIDIA GeForce RTX 3070	82nd	4	80013741667 +/- 228221564
NVIDIA GeForce RTX 3090 Ti	81st	4	75184916667 +/- 3931783726
Gigabyte NVIDIA GeForce RTX 4070 Ti	81st	4	74021816667 +/- 1638545931
Mid-Tier	75th		< 70991033333
NVIDIA GeForce RTX 3090	75th	39	70867552587 +/- 3204264127
AMD Radeon RX 7900 XTX	73rd	4	69163372857 +/- 1246558898
NVIDIA GeForce RTX 3080 Ti	72nd	14	67781101282 +/- 413951882
AMD Radeon RX 7900 XT	67th	5	61566224762 +/- 753630529
NVIDIA GeForce RTX 3080	66th	19	60284979323 +/- 871234740
AMD Radeon RX 6900 XT	63rd	9	58596096296 +/- 1383732339
AMD Radeon RX 6800 XT	56th	6	52565837302 +/- 1856012111
NVIDIA GeForce RTX 2080 SUPER	54th	3	43272383333
NVIDIA GeForce RTX 3070 Ti	52nd	12	42679897024 +/- 218910662

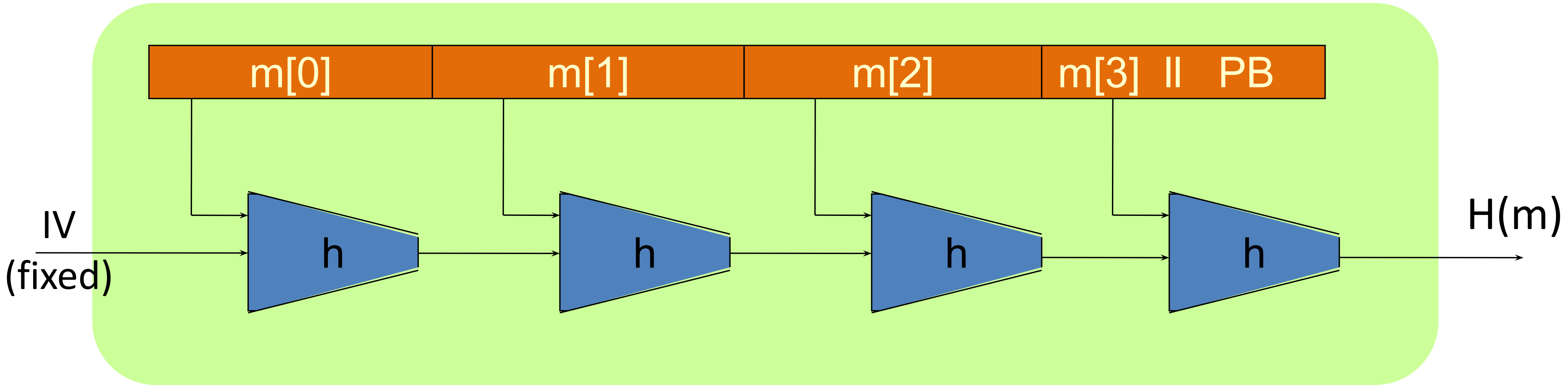
We just recovered the MD5 hash of a password: d50ba4dd3fe42e17e9faa9ec29f89708 . Can we get the original password?

Rainbow Tables

A **rainbow table** is a precompute table for caching the outputs of a hash function. Typically used for cracking password hashes. A common defense against this attack is to compute the hashes using a key derivation function that adds a "salt" to each password before hashing it, with different passwords receiving different salts, which are stored in plain text along with the hash.



The Merkle-Damgard iterated construction



Thm: h collision resistant ⇒ 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

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)

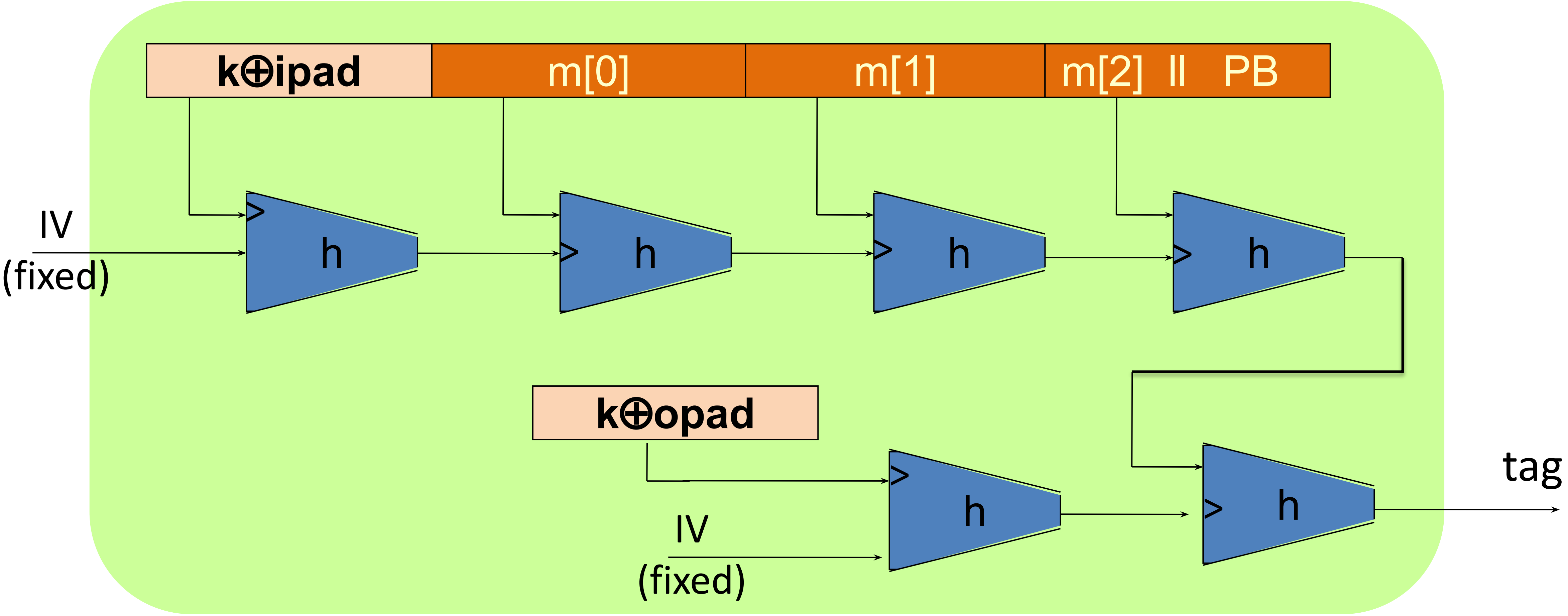
Most widely used MAC on the Internet.

H: hash function.
example: SHA-256 ; output is 256 bits

Building a MAC out of a hash function:

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

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 ($q \ll |T|^{½}$)

In TLS: must support HMAC-SHA1-96

Warning: verification timing attacks [L'09]

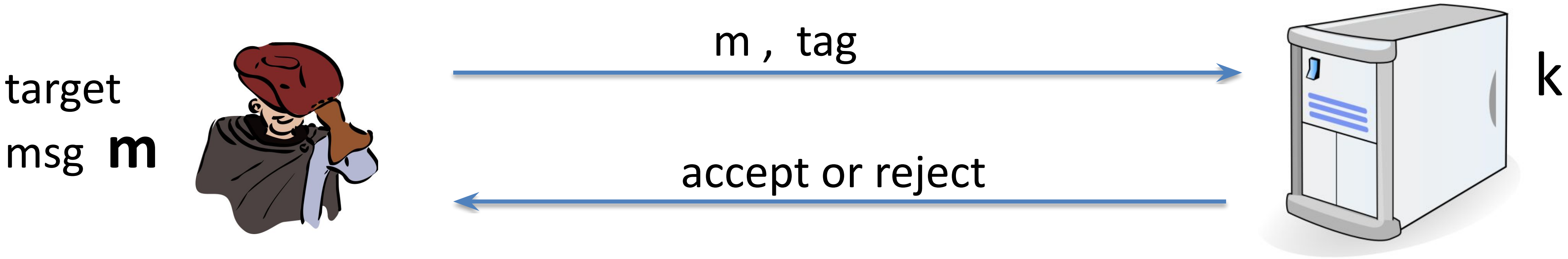
Example: Keyczar crypto library (Python) [simplified]

```
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]

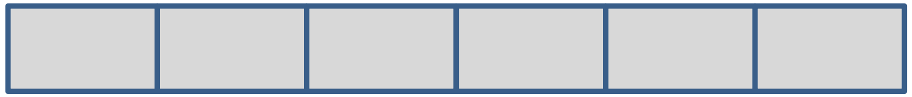


Timing attack: to compute tag for target message m do:

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

Hash Tricks and Datastructures

Commitment Scheme

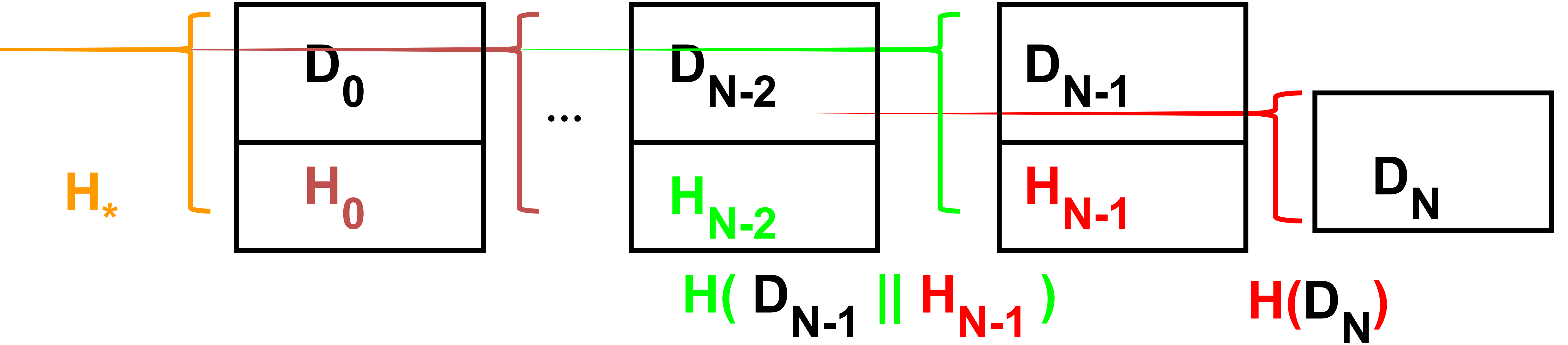
One-Way Chain Application (Lists)

- One-time password system
- Goal
  - Use a different password at every login
  - Server cannot derive password for next login
- Solution: one-way chain
  - Pick random password P_L
  - Prepare sequence of passwords $P_i = F(P_{i+1})$
  - Use passwords $P_0, P_1, \ldots, P_{L-1}, P_L$
  - Server can easily authenticate user

$$p_3 \xleftarrow{F} p_4 \xleftarrow{F} p_5 \xleftarrow{F} p_6 \xleftarrow{F} p_7$$

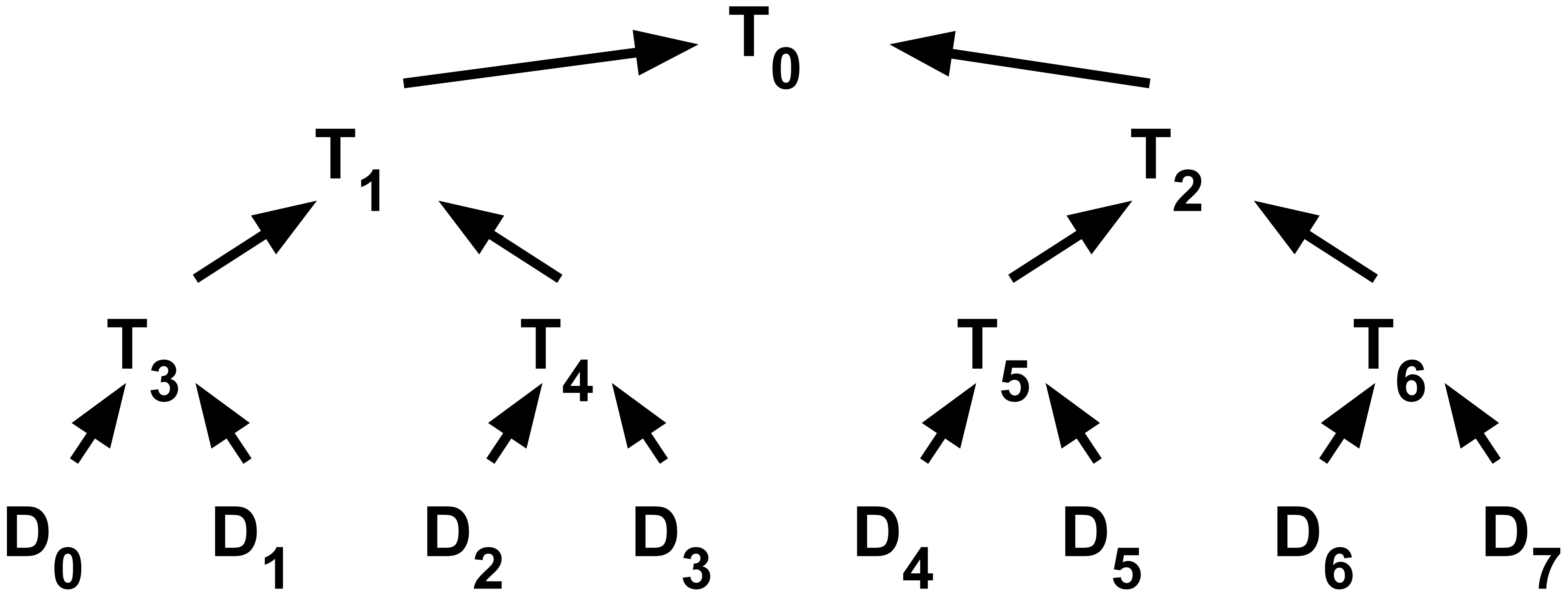
Chained Hashes

- More general construction than one-way hash chains
- Useful for authenticating a sequence of data values D_0 , D_1 , …, D_N
- H_* authenticates entire chain



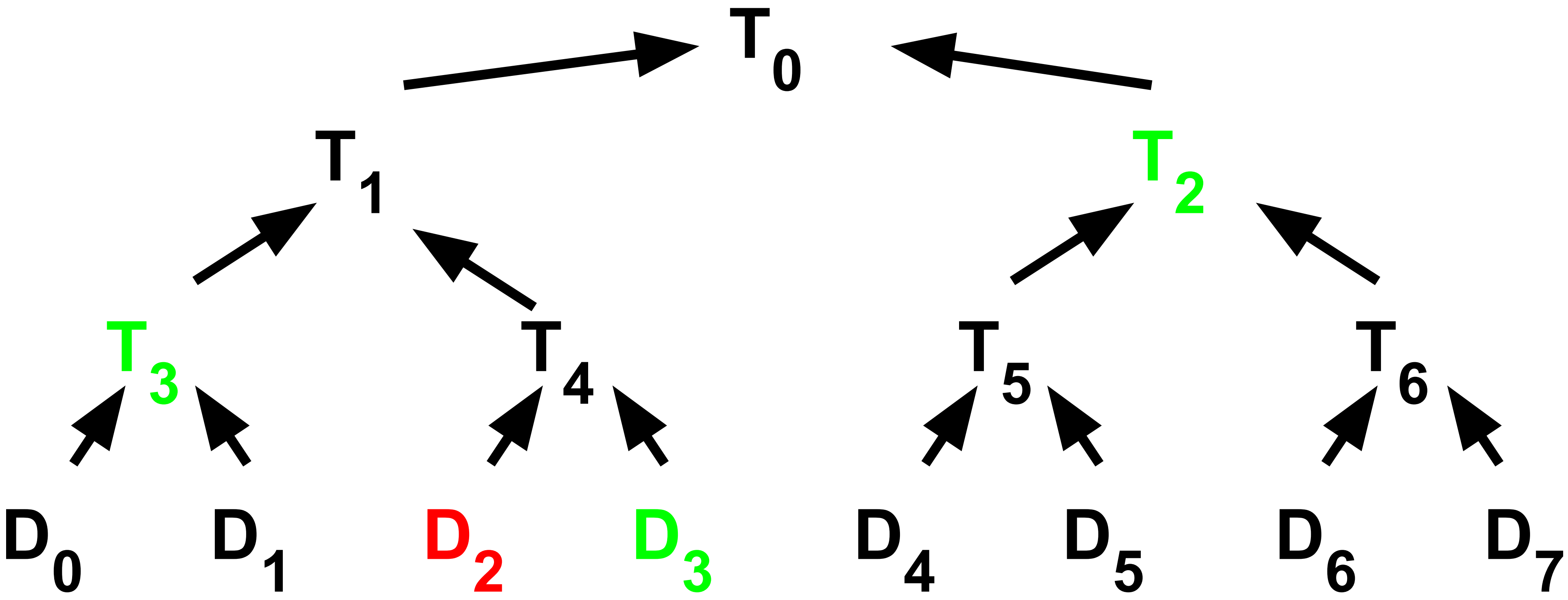
Merkle Hash Trees

- Authenticate a sequence of data values D_0 , D_1 , …, D_N
- Construct binary tree over data values



Merkle Hash Trees II

- Verifier knows T_0
- How can verifier authenticate leaf D_i ?
- Solution: recompute T_0 using D_i
- Example authenticate D_2 , send D_3 T_3 T_2
- Verify $T_0 = H(H(T_3 || H(D_2 || D_3)) || T_2)$



Ευχαριστώ και καλή μέρα εύχομαι!

Keep hacking!