Hash Functions

Sushmita Ruj

Recap

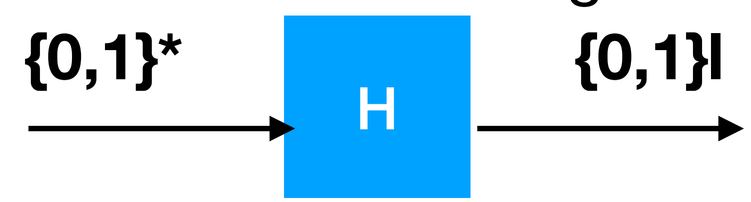
- Message integrity
- Message Authentication code
- Hash functions
- Merkle Trees

This Lecture

- Birthday paradox
- Hash Function Construction: Merkle Damgard, SHA
- Hash Function Construction: Sponge construction, SHA3
- HMAC
- Number Theory

What is a Hash Function?

- $H: \{0,1\}^* \rightarrow \{0,1\}^l$: Given variable length input produces a fixed length output (also called msg digest or fingerprint)



Properties of a hash function

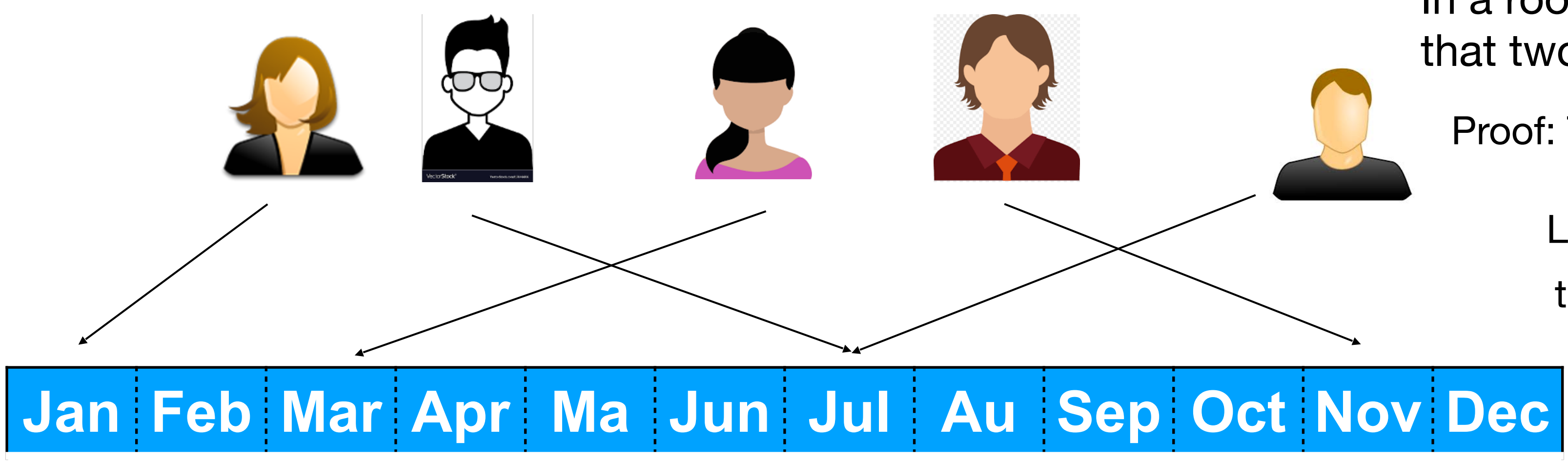
- Given x, $y=H(x)$ is easy to compute
- Given $y= H(x)$, x is computationally infeasible to compute (One-wayness)
- It is computationally infeasible to find two strings x,x' $(x\neq x')$, such that $H(x) = H(x')$ (collision resistance)
- Given $y=H(x)$, it is difficult to find $x\neq x'$, such that $H(x) = H(x')$ (second preimage resistance)

- Output cannot be too short: One can find collisions by random search ("birthday attack")
- For any input, the output should be "random"; cannot find (x,y), s.t. x is short and $y=H(x)$, except by picking x and evaluating $H(x)$.

Birthday Attack

Birthday Paradox

Birthday Paradox: In a group of 23 randomly chosen people, at least two will share a birthday with probability at least 1/2.



In a room there are Q people, what is the probability that two people will have the same birthday

Proof: Think of throwing Q balls in M bins. What is the probability that at least 1 bin contains 2 balls?

Let $\mathscr{X} = \{x_1, x_2, \cdots, x_Q\}$, E_i denote the event that the there are no balls $\{x_1, \cdots, x_{i-1}\}$ in the same bin as x_i

$Pr[E_1] = 1$

$Pr[E_2 \mid E_1] = (M-1)/M$

$Pr[E_3 \mid E_2, E_1] = (M-2)/M..$

$$Pr[E_1 \wedge E_2 \wedge \cdots E_Q] = Pr[E_1 \wedge E_2 \wedge \cdots E_{Q-1}] \,.\, Pr[E_Q \mid E_1 \wedge E_2 \wedge \cdots E_{Q-1}]$$

$$Pr[E_1 \wedge E_2 \wedge \cdots E_{Q-1}] \,.\, Pr[E_1 \wedge E_2 \wedge \cdots E_{Q-2}] \,.\, Pr[E_{Q-1} \mid E_1 \wedge E_2 \wedge \cdots E_{Q-2}] \ldots$$

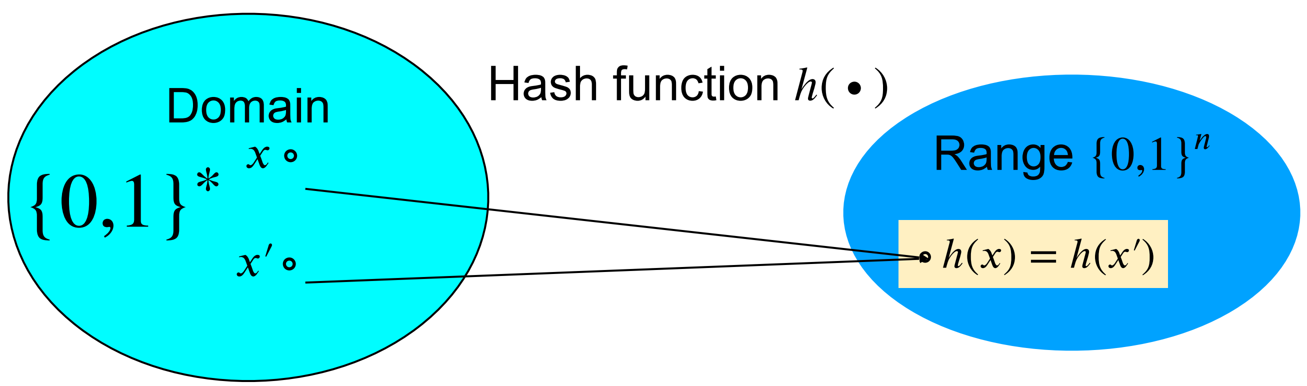
$$= \left(\frac{M-1}{M}\right)\left(\frac{M-1}{M}\right)\cdots,\left(\frac{M-Q+1}{M}\right)$$

Probability that there is at least one collision $\epsilon = 1 - Pr[E_1 \wedge E_2 \wedge \cdots E_Q] = 1 - e^{-Q(Q-1)/2M}$

If ϵ=1/2, Q=$1.17\sqrt{M}$ = 23

Q=O($\sqrt{M}$)

Generic Collision Attacks



- Easy: find collisions in 2^n time or with 2^{-n} probability
- **Birthday paradox**: birthday attack imposes a lower bound on the sizes of secure message digests
- 40-bit message digest would be very insecure, since a collision couldbe found with probability 1/2 with just over 2^{20} random hashes
- SHA-1, which was a standard for a number of years, has a message digest that is 160 bits
- SHA-3, utilizes hash functions having message digests of sizes between 224 and 512 bits in length

Performance & Security

AMD Opteron, 2.2 GHz (Linux)

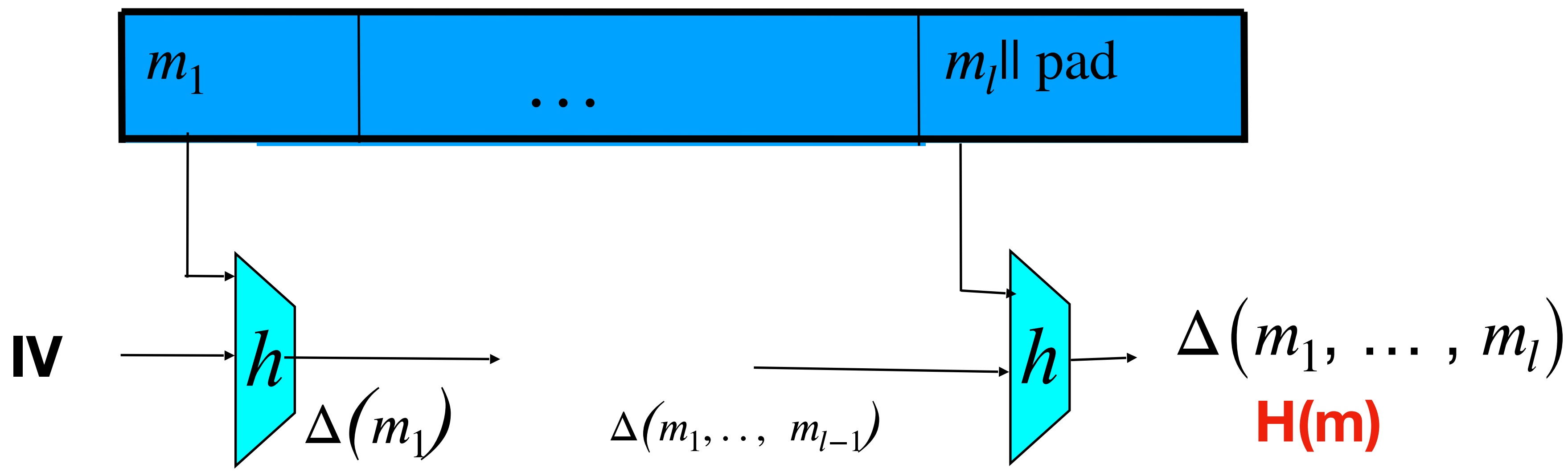
Hash Function	**Digest size (in bits)**	**Generic Attack time**	**Speed (MB/s)**
SHA-1	160	2^{80}	153
SHA-256	256	2^{128}	111
SHA-3	512	2^{256}	99

Attacks on SHA-1

- Read from Boneh-Shoup

Hash Function Constructions

Iterated Hash Function

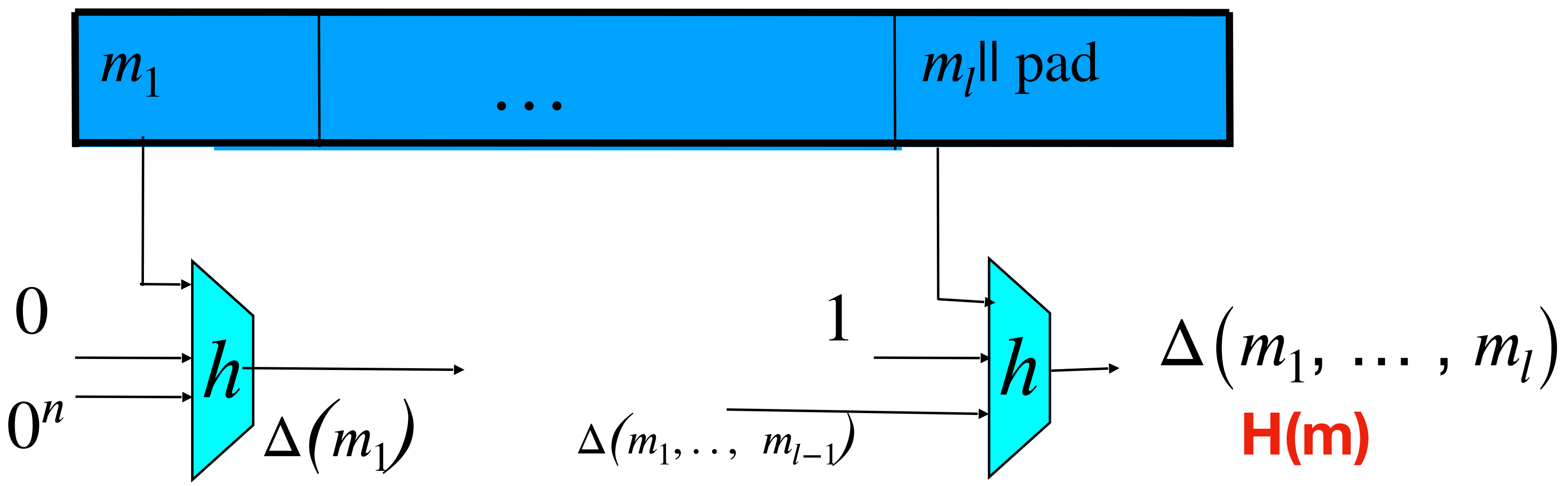


Given h: T × X ⟶ T (compression function)

we obtain a hash function H: $X^{\leq L}$ ⟶ T .

If Compression function is collision resistant, then H is collision resistant

Merkle-Damgard

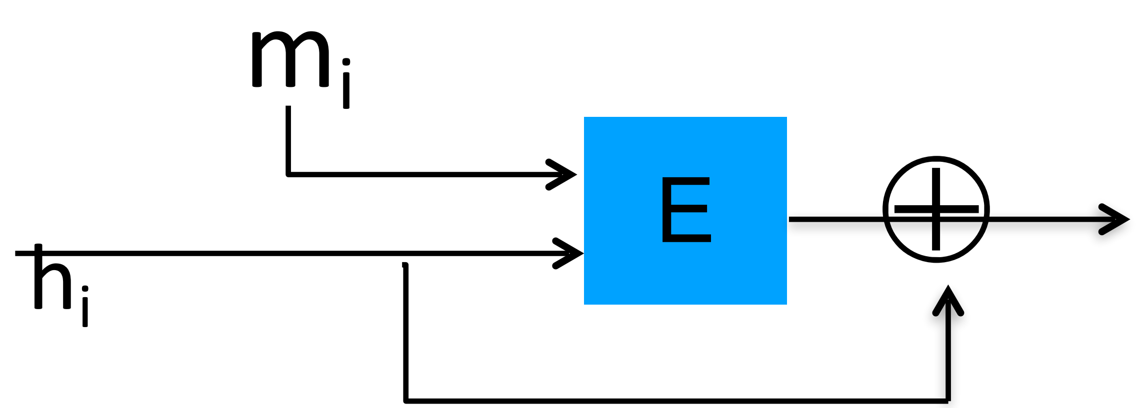


- The Merkle-Damgard construction of:
  - Collision-Resistant Digest function from CRHF
  - CRHF from compression function: $|m_i| = n$
- Idea: hash iteratively, message by message:
- $\Delta(m_1, \ldots, m_l) = h(\Delta(m_1, \ldots, m_{l-1}) \,||\, 1 \,||\, m_l)$; $\Delta(m_1) = h(0^{n+1} \,||\, m_1)$
- Lemma 4.2: if h is collision resistant, then H is collision resistant

Compression Functions

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.).

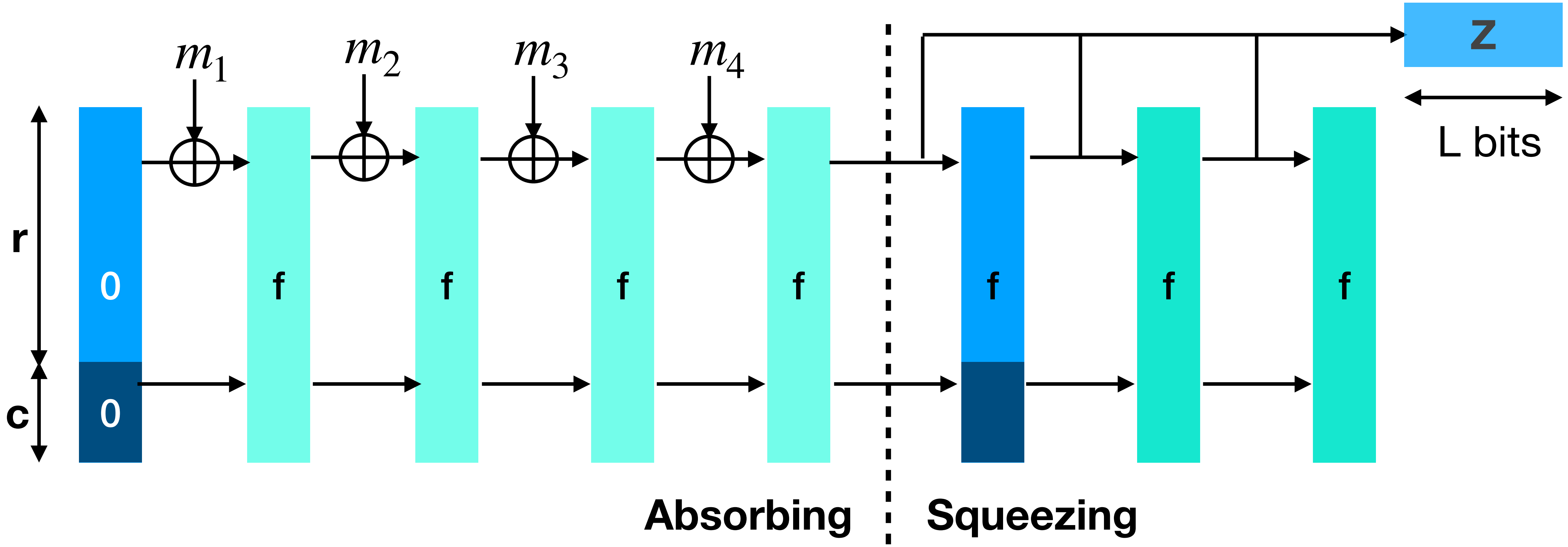
Finding a collision **h(H,m)=h(H',m')** takes **$O(2^{n/2})$** evaluations of (E,D).

Proof : Boneh-Shoup.

Choice of Compression Functions

- AES is not a good option for David-Meyer Compression function
- In AES, the same key is used which adds to its efficiency, here keys are different, so this will be slower.
- SHA-256 has custom made block cipher which is faster than simple AES based scheme
- Other constructions Matyas-Meyer-Oseas,Miyaguchi-Preneel (Read Boneh-Shoup for more information)
- Other Merkle-Damgard hash functions are MD4, MD5, Whirlpool, RIPEMD-160 (Used in Bitcoin because of its short size)

Sponge Construction



- h was a compression function, f is a permutation function, has no key
- r is the rate of the sponge, higher the r, faster is the function
- Security depends on c, larger c lead to better security
- It is not known how to reduce the collision resistance of the sponge to a concrete security property of the permutation
- Sponge constructions are flexible and used where collision resistance is not the main desirable property
- **Absorbing phase:** message blocks get mixed up
- **Squeezing phase:** Output is pulled out of the chaining variable
- Examples: SHA-3 family standardised by NIST, Keccak is the permutation used

HMAC

- Build MAC for long messages
- Use hash function to build MAC
- Hash-then-sign:
- m mesg, compute H(m), I= (TG, Ver), t=TG(k, H(m))
- If Adversary finds collision in H, then this is broken
- This needs a hash function and a MAC, so, better to have one algorithm
- Can we use a keyless hash function and convert into a key-ed function which is a secure MAC or Secure PRF?

Attempts

- Prepend the key: $F_{pre}(k,M) := H(k||M)$.
- Broken because, given $F_{pre}(k, M)$, one can easily compute $F_{pre}(k, M||PB\ ||M')$, for any M'. PB is the padding block for the message k||M
- Append the key: $F_{post}(k,M) := H(M||k)$
- If there are two distinct messages of the same length M_0 and M_1 such that $h(IV, M_0) = h(IV, M_1)$, then $F_{post}(k,M_0)= F_{post}(k,M_1)$

Hash MAC (HMAC)

Most widely used MAC on the Internet.

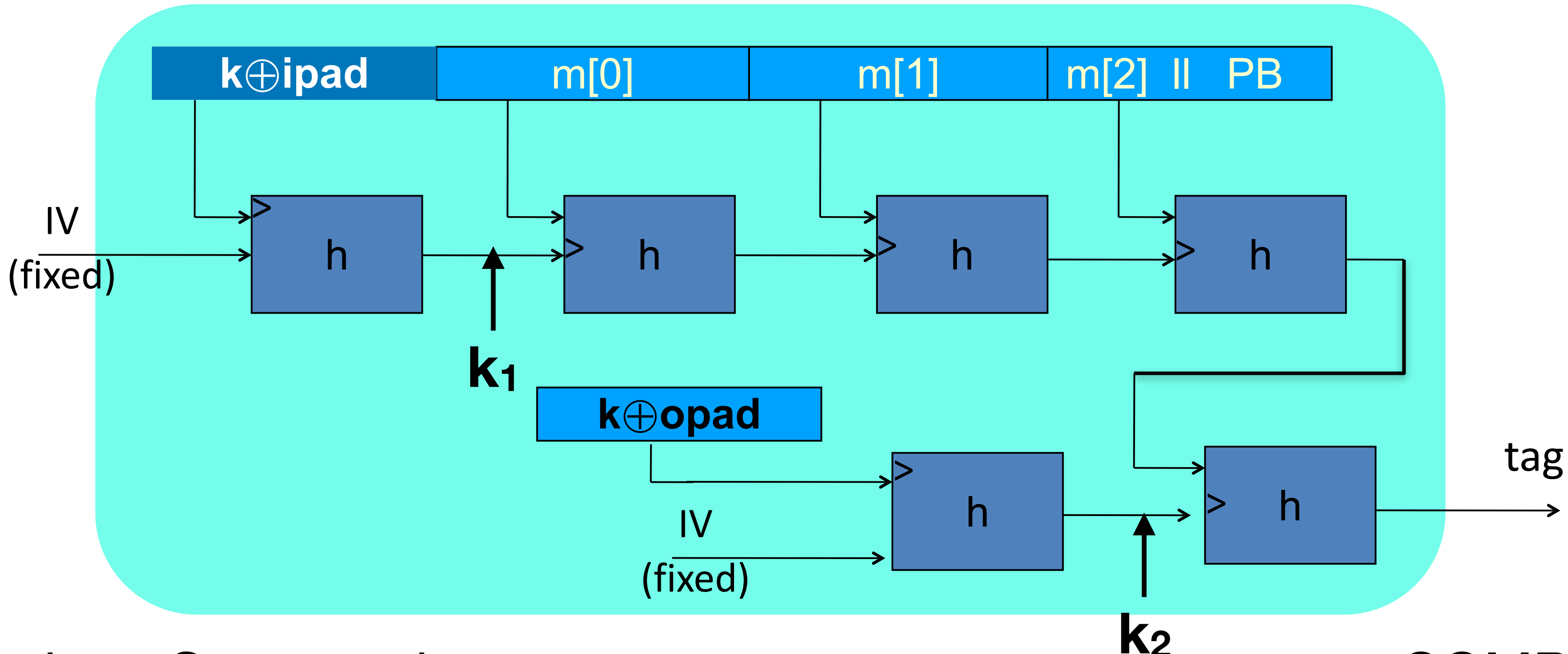
H: hash function.

example: SHA-256 ; output is 256 bits

Building a MAC out of a hash function:

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

ipad = the byte 0x36 repeated B times
opad = the byte 0x5C repeated B times
B is the size of the message.



Authenticated Encryption

- Mac-and-encrypt
- Mac-then-Encrypt
- Encrypt-then-MAC
- (Details explained during the lecture)

Basic Number Theory

Notations

<u>**Def:**</u> <u>$\mathbf{Z^*_N}$</u> = (set of invertible elements in Z_N) =

= $\{\ x \in\ Z^*_N\ :\ \gcd(x,N) = 1\ \}$

Examples:

1. for prime p, $Z^*_p = Z_p - \{0\} = \{1, 2, \ldots p-1\}$
2. $Z^*_{12} = \{ 1, 5, 7, 11\}$

For x in Z^*_N, can find x^{-1} using extended Euclid algorithm.

Modular Inversion

Over the rationals, inverse of 2 is ½ . What about in Z_N ?

<u>Def</u>: The **inverse** of x in is an element y in s.t. x.y =1
y is denoted x^{-1} .

Examples: 7^{-1} mod 11 = ?

Example: let N be an odd integer. The inverse of 2 in is

<u>Lemma</u>: x in Z_N has an inverse if and only if gcd(x,N) = 1

Proof:

gcd(x,N)=1 ⇒ ∃ a,b: $a \cdot x + b \cdot N = 1$, => a.x =1 => $x^{-1} = a$

If gcd(x,N) >1, a.x not= 1, so no inverse.

then say, gcd (x, N) = 2, so, for all a, gcd(a,N) is even.

GCD

Def: For ints. x,y: **gcd(x, y)** is the greatest common divisor of x,y

Example: gcd(24, 18) = 6

Fact: for all ints. x,y there exist ints. a,b such that

$$\mathbf{a \cdot x + b \cdot y = gcd(x,y)}$$

a,b can be found efficiently using the extended Euclid alg.

If gcd(x,y)=1 we say that x and y are **relatively prime**

Euclidean Algorithms

Algorithm 6.1: EUCLIDEAN ALGORITHM(a, b)

$r_0 \leftarrow a$
$r_1 \leftarrow b$
$m \leftarrow 1$
while $r_m \neq 0$
$$\textbf{do} \begin{cases} q_m \leftarrow \lfloor \frac{r_{m-1}}{r_m} \rfloor \\ r_{m+1} \leftarrow r_{m-1} - q_m r_m \\ m \leftarrow m+1 \end{cases}$$
$m \leftarrow m - 1$
return $(q_1, \ldots, q_m; r_m)$
comment: $r_m = \gcd(a, b)$

Extended Euclidean Algorithm

Algorithm 6.2: EXTENDED EUCLIDEAN ALGORITHM(a, b)

$a_0 \leftarrow a$
$b_0 \leftarrow b$
$t_0 \leftarrow 0$
$t \leftarrow 1$
$s_0 \leftarrow 1$
$s \leftarrow 0$
$q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor$
$r \leftarrow a_0 - qb_0$
while $r > 0$

$$\textbf{do} \begin{cases} temp \leftarrow t_0 - qt \\ t_0 \leftarrow t \\ t \leftarrow temp \\ temp \leftarrow s_0 - qs \\ s_0 \leftarrow s \\ s \leftarrow temp \\ a_0 \leftarrow b_0 \\ b_0 \leftarrow r \\ q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor \\ r \leftarrow a_0 - qb_0 \end{cases}$$

$r \leftarrow b_0$
return (r, s, t)
comment: $r = \gcd(a, b)$ and $sa + tb = r$

Thank you