Week 13

Cryptographic Hash fun^n.

- provides assurance of **data integrity** → to detect modification of transmitted message (corrupted or not)
- $y = h(x)$, h is a hash fun^n. → message digest
- y stored in a secure place, → short fingerprint of data x
  ~~if x is changed~~
  x is not.

if x is changed to x' and y is not old digest of x', then a message verification is done by computing $y' = h(x')$ & checking that $y' \neq y$.

(y must be stored in a secure place).

- → unkeyed hash fun^n. (e.g. MDC)
- → keyed hash fun^n. (e.g. MAC).

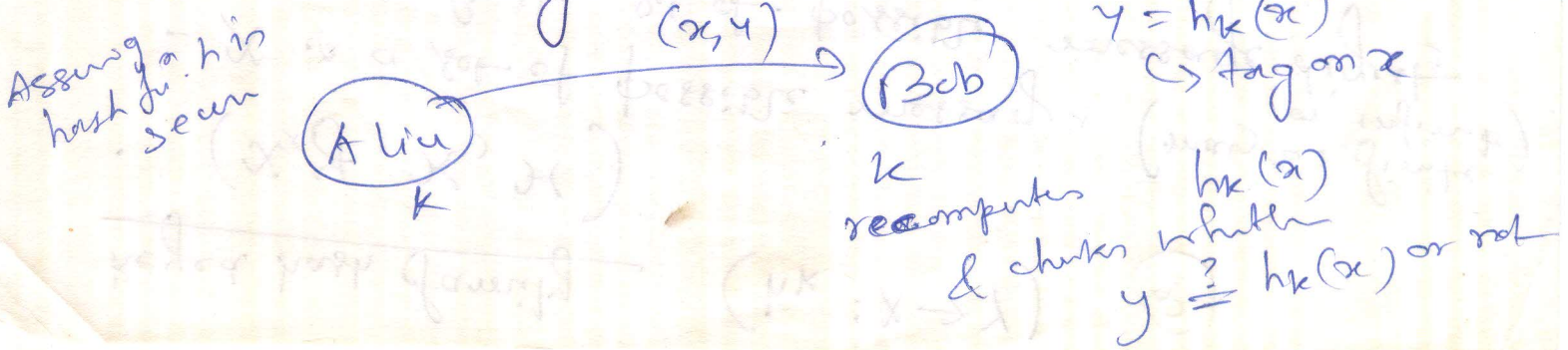
(both x & y ~~may be~~ can be transmitted).

$y = h_K(x)$, K is a secret key shared between Alice & Bob.

↙ tag or authentication tag on x

MDC → ~~Mof~~ modification detection code
or
manipulation detection code
or
message integrity codes (MIC).

MAC → message authentication code.



Keyed hash family $(h_K : X \to Y)$

- $(X, Y, \mathcal{K}, \mathcal{H})$
  - X is a set of possible messages (may be finite or infinite).
  - Y is a finite set of possible message digests or authentication tags
  - $\mathcal{K}$, the key space, is a finite set of possible keys.
  - for each $K \in \mathcal{K}$, there exists a hash fun^n. $h_K \in \mathcal{H}$. Each $h_K : X \to Y$.
- $h_K \to$ compression fun^n. when both X & Y are finite. $|X| > |Y|$.
- $(x, y) \in X \times Y$ is a valid pair if $h_K(x) = y$.
- prevent adversary to construct such valid pairs.

Unkeyed hash function $(h : X \to Y)$

- Keyed hash fun^n with $|\mathcal{K}| = 1$. i.e. only one possible key

$h : X \to Y$.

$|X| = N$, $|Y| = M$

$F^{X \times Y}$ → set of all funs from X to Y.

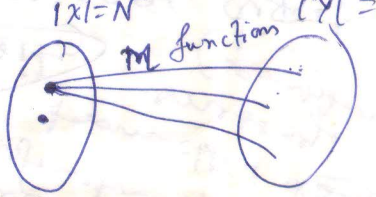
$|F^{X \times Y}| = M^N$

any hash family $f \in F^{X \times Y}$ is called an (M, N) hash family

Security of a Hash functions (unkeyed)

~~Preimage~~

- Difficult to solve the following three problems for a cryptographically secure hash fun^n.
  - preimage
  - second preimage
  - collision.

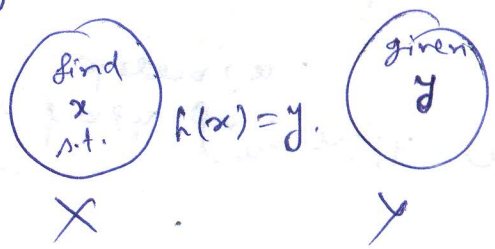


Preimage problem

Instance: A hash fun^n $h: X \to Y$ & an element $y \in Y$

find: $x \in X$ s.t. $h(x) = y$.

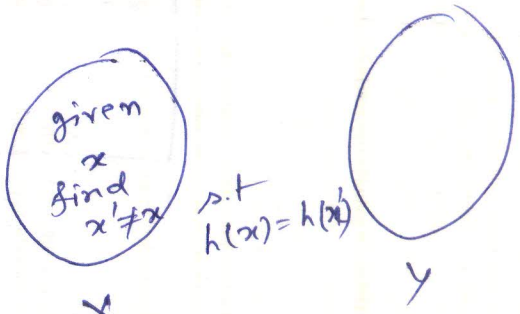
- Hash fun^n for which preimage cannot be efficiently solved is called one-way or preimage resistant.



Second preimage problem

Instance: A hash fun^n $h: X \to Y$ & an element $x \in X$

find: $x' \in X$ s.t. $x' \neq x$ and $h(x') = h(x)$.



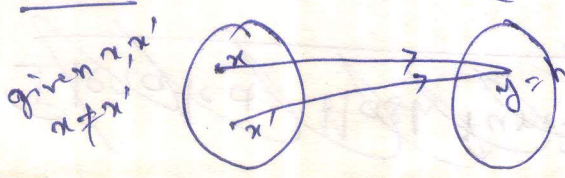
- Hash fun^n for which second preimage cannot be efficiently solved is called second pre-image resistant or weak collision resistant.

Collision Problem

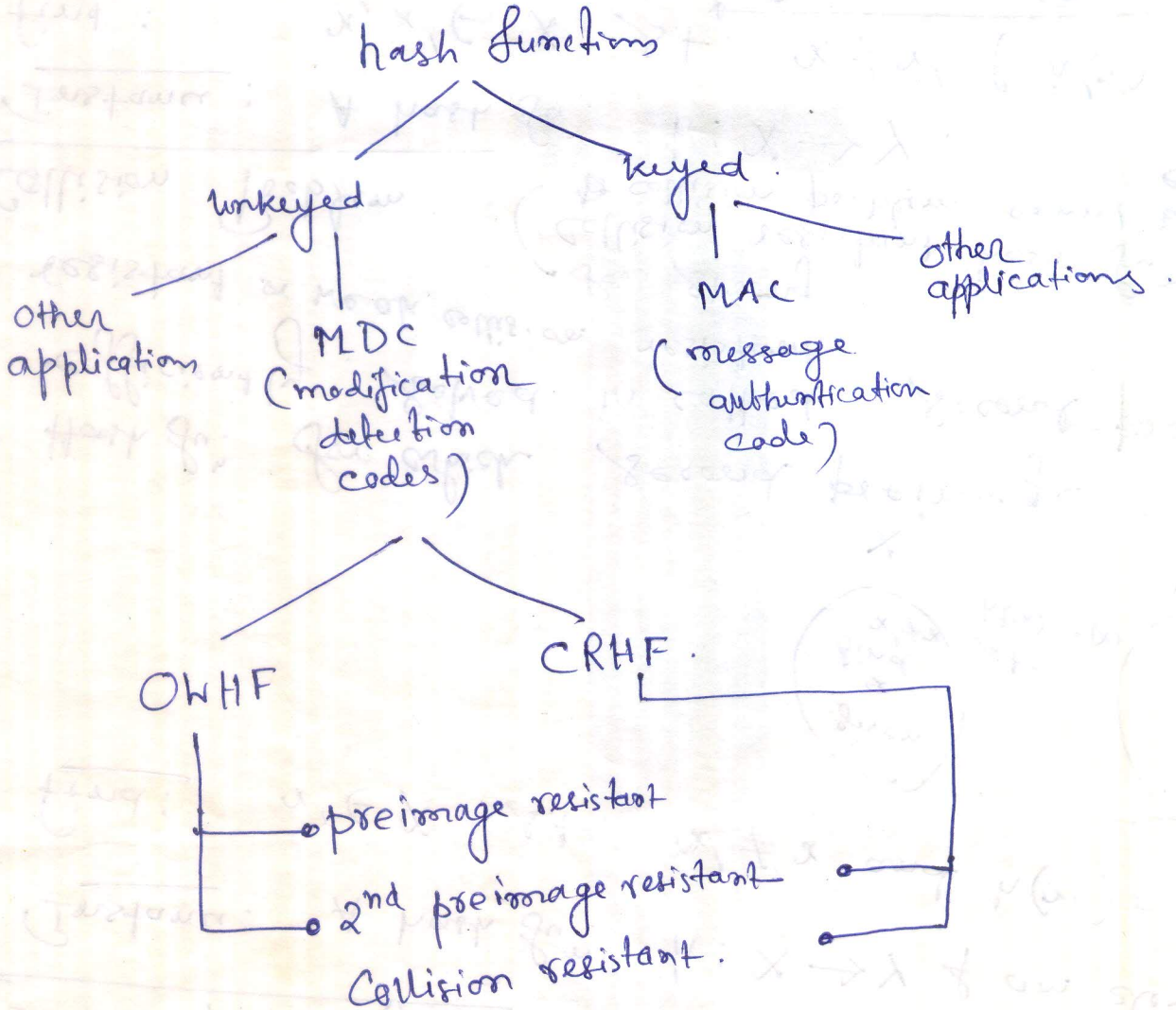
(for strong collision resistant hash fun^n, collision problem cannot be solved efficiently).

Instance: A hash fun^n $h: X \to Y$.

find: $x, x' \in X$ s.t. $x \neq x'$ & $h(x) = h(x')$.



Iterated Hash functions



Iterated Hash functions

- Compression function → hash fu^ns with a finite domain.
  say, 'compress'
- extending 'compress' to a hash fu^n h with an infinite domain.
- h → iterated hash fu^n.

- Compress : $\{0,1\}^{m+t} \longrightarrow \{0,1\}^m$. $(t \geq 1)$.
- Construct an iterated hash fu^n.

$$h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \longrightarrow \{0,1\}^l ,$$

based on the compression fu^n compress

Preprocessing Step

Given an input x, $|x| \geq m+t+1$, construct a String y, using a public algorithm, such that-

$$|y| \equiv 0 \pmod t$$

$y = y_1 \| y_2 \| \cdots \| y_r$, where $|y_i| = t$, $1 \leq i \leq r$.

processing step

Let IV be a public initial value which is a bit string of length m. $IV \in \{0,1\}^m$.

The compute

$z_0 \leftarrow IV$ (z_0: m bit, y_1: t bit)

$z_1 \leftarrow$ Compress $(z_0 \| y_1)$

$z_2 \leftarrow$ Compress $(z_1 \| y_2)$.

$\vdots$

$z_r \leftarrow$ compress $(z_{r-1} \| y_r)$

Output transformation (optional).

- $g : \{0,1\}^m \rightarrow \{0,1\}^l$, a public fu^n.

define $h(x) = g(z_r)$. (l-bit).

- if output transformation is not desired, then define $h(x) = z_r$.

- **padding in pre-processing step.** (mapping $x \to y$ must be injective).

$$y = x \| \text{pad}(x).$$

- pad(x) is a padding fun^n.
  - → incorporates $|x|$ & pads the result with additional bits (e.g. zeros). So that resulting string y has length that is a multiple of t.

Example: SHA (Secure Hash Algorithm).

SHA-1-PAD(x). // $|x| \le 2^{64} - 1$.

input x with $|x| \le 2^{64} - 1$.

output y with $|y| = 512 = 2^9$.

$$y = x \| 1 \| 0^d \| l$$

(1 bit; x: $|x|$ bit; 0^d: ?; l: 64 bit; y: 512 bit)

where $d \leftarrow (447 - |x|) \bmod 512$.

$l \leftarrow$ binary representation of $|x|$, where $|l| = 64$.

$$\{512 - (|x| + 1 + 64)\} \bmod 512$$
$$= (447 - |x|) \bmod 512.$$

i.e. $d = 512k + (447 - |x|)$; k integer.

$a = b \bmod c$

$\Rightarrow c \mid a - b$

$a - b = ck$ for some integer k.

- if the mapping $x \to y$ in the preprocessing step is not injective then h would not be collision-resistant. i.e. (if $y = y'$ for $x \neq x'$, then $h(x) = h(x')$).

The Markle-Damgård Construction (iterated hash fun.)

- compress: $\{0,1\}^{m+t} \longrightarrow \{0,1\}^m$.
  a collision resistant compression fun, $t \geq 2$.
- To compute a collision resistant hash fun.
  $h: X \longrightarrow \{0,1\}^m$, when $X = \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i$

Preprocessing

(Padding)

Input $\rightarrow x \in X$, $|x| = n \geq m+t+1$

let $x = x_1 \| x_2 \| \cdots \| x_k$ (x_i: $(t-1)$ bit, x_k: $(t-1-d)$ bit)

$|x_1| = |x_2| = \cdots = |x_{k-1}| = t-1$

$|x_k| = (t-1) - d$, when $0 \leq d \leq t-2$, $k = \left\lceil \frac{n}{t-1} \right\rceil$

$d = (t-1)k - n$

Output $y = y_1 \| y_2 \| \cdots \| y_k \| y_{k+1}$

$(t-1)$ bit $\leftarrow y_i = x_i$, $1 \leq i \leq k-1$

$(t-1)$ bit $\leftarrow y_k = x_k \| 0^d$

pad on the left with zeros to make $|y_{k+1}| = t-1$. $\leftarrow y_{k+1}$ = the binary representation of d.

processing

$z_1 = 0^{m+1} \| y_1$ ($m+t$ bit, $t-1$ bit)

m-bit $\leftarrow g_1 \leftarrow$ compress (z_1)

for $1 \leq i \leq k$, $z_{i+1} \leftarrow g_i \| 1 \| y_{i+1}$ (m bit, 1 bit, $(t-1)$ bit)

$g_{i+1} \leftarrow$ compress (z_{i+1})

Set $h(x) = g_{k+1}$ (m-bit)

Design principal for collision resistant iterated hash fun^n.

- mapping $x \to y$ must be injective.
- Compress fun^n must be collision resistant.

Example of Compress fun^n.

Compress : $\{0,1\}^{m+t} \longrightarrow \{0,1\}^m$.

SHA-1 (x)

Input : x, $|x| \le 2^{64} - 1$.

Output : 160-bit message digest

$H_0 \| H_1 \| H_2 \| H_3 \| H_4$ → $32 \times 5 = 160$-bit.
(32 bits, 32 bits)

preprocessing

$y = SHA\text{-}1\text{-}PAD(x)$.

$= M_1 \| M_2 \| \cdots \| M_n$, each $M_i \to 512$ bits. block.

$= x \| 1 \| 0^d \| l$, l = binary representation of $|x|$

($|x|$ bit, 1 bit, d, 64 bit) $|l| = 64$, zeros are padded on left if $|x| < 64$.

$d = \{512 - (|x| + 1 + 64)\} \bmod 512$

$= \{447 - |x|\} \bmod 512$

i.e. $d = 512k + (447 - |x|)$.

Compress : $\{0,1\}^{\overset{t}{512+160}} \longrightarrow \{0,1\}^{\overset{m}{160}}$.

$IV = \underbrace{H_0 \| H_1 \| H_2 \| H_3 \| H_4}_{160 \text{ bits}}$, $y = \underbrace{M_1 \| M_2 \| \cdots \| M_n}_{512n \text{ bits}}$.

$z_1 = \text{Compress}(IV \| M_1)$ (160-bit, 512-bit)
$\downarrow$ 160 bit
$z_2 = \text{compress}(z_1 \| M_2)$
$z_3 = \text{compress}(z_2 \| M_3)$
$\vdots$
$z_n = \text{compress}(z_{n-1} \| M_n)$

Processing $H_0 = 67452301 \rightarrow 8 \times 4 = 32$ bits

$H_1 = EFCDAB89$

$H_2 = 98BADCFE$

$H_3 = 10325476$

$H_4 = C3D2E1F0$.

$y = M_1 \| M_2 \| M_3 \| \cdots \| M_n$ — 512 bits.

$M_i = W_0 \| W_1 \| \cdots \| W_{15}$ — 512 bits; each W_i is 32 bits.

for $i = 1$ to n do

  // $M_i = W_0 \| W_1 \| \cdots \| W_{15}$, each W_i is 32-bit

  for $j = 16$ to 79 do

    $W_j = ROTL^1 (W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16})$ // circular left shift by 1 position

  end do

  Compress ($H_0 \| H_1 \| H_2 \| H_3 \| H_4$, M_i 512 bit, 160-bit)

  $(A, B, C, D, E) \leftarrow (H_0, H_1, H_2, H_3, H_4)$

  for $j = 0$ to 79 do

    $temp := ROTL^5(A) + f_j(B, C, D) + E + W_j + K_j$ // Circular left shift by 5 positions.

    $E := D$

    $D := C$

    $C := ROTL^{30}(B)$ // Circular left shift by 30 positions

    $B := A$

    $A := temp$.

  end do.

  $(H_0, H_1, H_2, H_3, H_4) \leftarrow (H_0 + A, H_1 + B, H_2 + C, H_3 + D, H_4 + E)$.

end do.

return $(H_0 \| H_1 \| H_2 \| H_3 \| H_4)$

circular shift: A, B, C, D, E

$$f_j(B,C,D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D), & 0 \le t \le 19 \\ B \oplus C \oplus D, & 20 \le t \le 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D), & 40 \le t \le 59 \\ B \oplus C \oplus D, & 60 \le t \le 79 \end{cases}$$

$$K_j = \begin{cases} 5A827999, & 0 \le t \le 19 \\ 6ED9EBA1, & 20 \le t \le 39 \\ 8F1BBCDC, & 40 \le t \le 59 \\ CA62C1D6, & 60 \le t \le 79. \end{cases}$$

Other Examples

MD4, MD5, SHA, SHA-1

SHA-256, SHA-384, SHA-512.

SHA-3 Competition (2007–2012)

- 64 submissions by Oct, 2008. → 1st round 51 → 2nd round 14 → 3rd round 5.
- 5 finalists in Dec. 2010.
  - BLAKE
  - Grøstl
  - JH
  - Keccak
  - Skein

hosted by NIST (National Institute of Standards & Technology)

- Winner: **Keccak** in Oct, 2012.

SHA-3.

CBC-MAC (x, k)

$x = x_1 \| x_2 \| \dots \| x_n$

$IV \leftarrow 00\dots0$

$y_0 \leftarrow IV$

for $i = 1$ to n do

$y_i \leftarrow e_k(y_{i-1} \oplus x_i)$

end do

return y_n

- Secure if the underlying encryption satisfies appropriate security properties, with $x_i \neq x_j$ for $i \neq j$
- e_k random fun^n i.e. $e_k(y_{i-1} \oplus x_i) = e_k(y_{j-1} \oplus x_j)$ occurs with prob $\frac{1}{2}$
  - ↳ then CBC-MAC will be secure.