7. Hash Functions

Map large domains into smaller codomains: |X| >> |Y|

One application is Data integrity: we want compact value to represent large amount of data

7.1 Properties

1. Arbitrary input length: process even large files.

2. Fixed and short outpt length.

3. Efficiency: fast even with large inputs.

7.2 Security requirements

1. Preimage resistance (one wainess): for a given output y = h(x), it is computationally infeasible to find any input x such that h(x) = y.

2. Second preimage resistance (weak collision resistance): given x1, and thus h(x1), it is computationally infeasible to find any x2 such that h(x1) = h(x2).

3. Collision resistance (strong collision resistance): It is computationally infeasible to find any pairs (x1, x2) where x1 != x2 such that h(x1) = h(x2).

There is no possible way to avoid collision; however a HF requires to be difficult to find collisions.

7.3 Attacks

1. Second preimage brute-force

Find (x’, h(x’)) s.t. x!=x’ and h(x) = h(x’)

Complexity: 2^n (secure n = 80)

2. Collision resistance attack

the attacker has two degrees of freedom to find a collision. This makes even a “simple” brute force attack much easier than what we may expect. (Birthday paradox)

P(no collision) = Π (1 – $i/2^n$)

Attack complexity is roughly the square root of the output space.

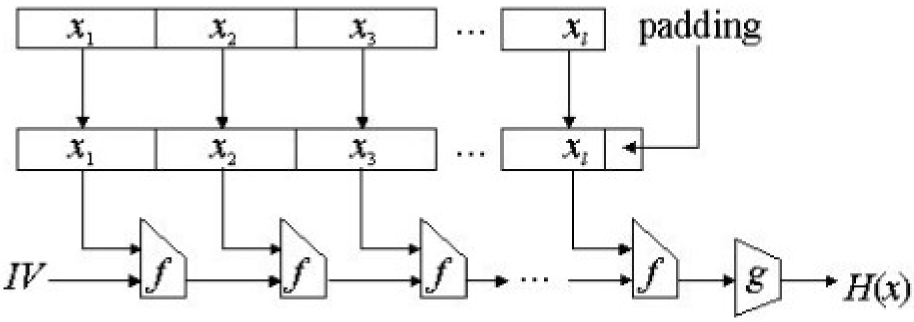
Strong collision implies weak collision:

- suppose there is polynomial algorithm A_h: $A_h(x) = x'$ such that $h(x) = h(x')$

- construct a polynomial algorithm $B_h() = (x, x')$: randomly picks x and returns $(x, A_h(x))$

7.4 Merkle-Damgard construction

Set of HF to fullfil requirement of arbitrary input length

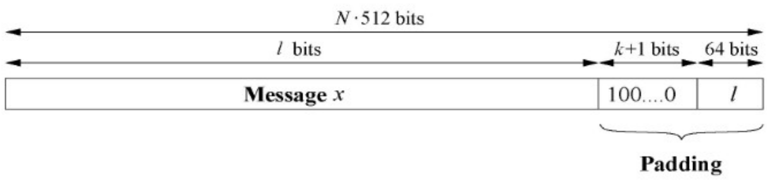


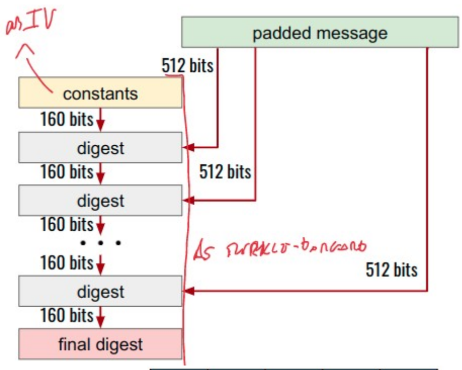
7.5 Random and Prime Numbers

MD (Message Digest) family
SHA (Secure Hash Algorithm) family

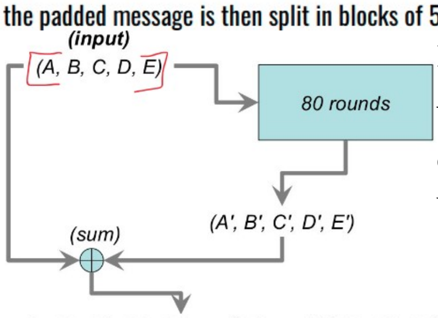
7.5.1 SHA-1

input length: up to 2^{64}
block size: 512 bits
output length: 160 bits
original message is padded with (10...00) and input length (l) to reach a multiple of block size:



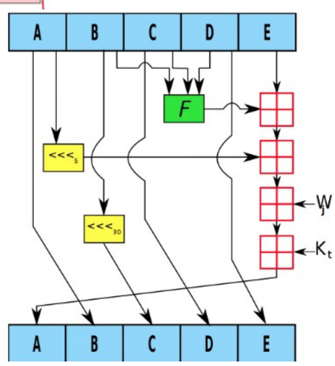


the padded message is then split in blocks of 512 bits



It is based on Merkle-Damgard, 80 rounds.

Wj: 32 bit word obtained by expanding original 16 words (512 bit block) into 80 words (using shift and xor).



$$A, B, C, D, E = (E + f_t(B, C, D) + (A)_{\lll 5} + W_j + K_t), A, (B)_{\lll 30}, C, D$$

Stage t	Round j	Constant K_t	Function f_t
1	$0 \ldots 19$	$K_1 = $ 5A827999	$f_1(B,C,D) = (B \wedge C) \vee (\bar{B} \wedge D)$
2	$20 \ldots 39$	$K_2 = $ 6ED9EBA1	$f_2(B,C,D) = B \oplus C \oplus D$
3	$40 \ldots 59$	$K_3 = $ 8F1BBCDC	$f_3(B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
4	$60 \ldots 79$	$K_4 = $ CA62C1D6	$f_4(B,C,D) = B \oplus C \oplus D$

$$W_j = \begin{cases} x_i^{(j)} & 0 \le j \le 15 \\ (W_{j-16} \oplus W_{j-14} \oplus W_{j-8} \oplus W_{j-3})_{\lll 1} & 16 \le j \le 79, \end{cases}$$

7.5.2 SHA-3

●Output length:

- ○224 (same resistance as 3DES when using birthday attack),
- ○256
- ○384 (same resistance as 192 AES when using birthday attack)
- ○512

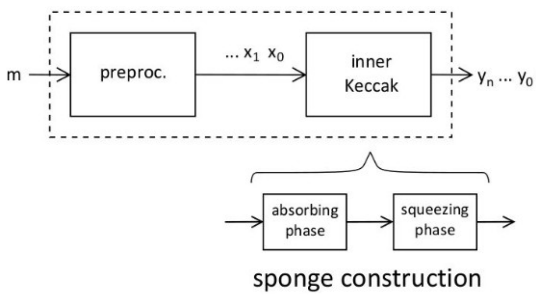
●Keccak state size b, also called bus, is: $b = 25 \cdot 2^l$ where $l \in \{0, 1, 2, \ldots, 6\}$

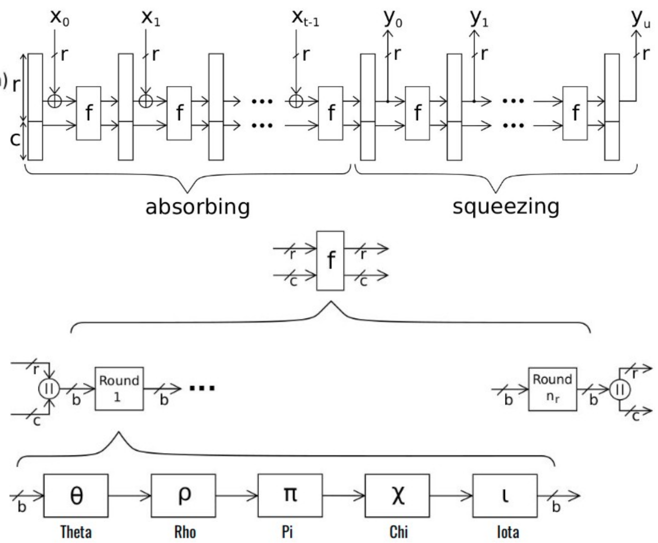
●SHA-3: only use b = 1600 $b = \{25, 50, 100, 200, 400, 800, 1600\}$

Number of rounds = 12 + 2l

block size r, capacity c = b - r

b (state) [bits]	r [bits]	c [bits]	security level [bits]	hash output [bits]
1600	1152	448	112	224
1600	1088	512	128	256
1600	832	768	192	384
1600	576	1024	256	512





7.5.2 Theta step

Each of the 1600 bits is replaced by the xor sum of 10 bits “in its neighborhood” and the original bit itself.

Original bit XOR 5-bit column to left of the bit XOR 5-bit column to the right and oneposition to the front of the bit.

7.5.3 Rho and Pi steps

We consider words A(x,y) of length 64: we take all 64 bits on the z-axis given a (x, y) position

Rho step: temp[x, y] = rotate(A[x,y], r[x,y])

Pi step: permutations of the words based on B[y, 2x+3y] = temp[x, y] (B is the new state)

E.g., temp[3,1] = rot(A[3,1], 55) then B[1, 9] = temp[3,1]

7.5.4 Chi step

$$A[x,y] = B[x,y] \oplus ((\bar{B}[x+1,y]) \wedge B[x+2,y])$$

where $\bar{B}$ is the bitwise complement

7.5.5 Iota step

$$A[0,0] = A[0,0] \oplus RC[i]$$

where RC[i] is given by:

RC[0] = 0x0000000000000001	RC[12] = 0x000000008000808B
RC[1] = 0x0000000000008082	RC[13] = 0x800000000000008B
RC[2] = 0x800000000000808A	RC[14] = 0x8000000000008089
RC[3] = 0x8000000080008000	RC[15] = 0x8000000000008003
RC[4] = 0x000000000000808B	RC[16] = 0x8000000000008002
RC[5] = 0x0000000080000001	RC[17] = 0x8000000000000080
RC[6] = 0x8000000080008081	RC[18] = 0x000000000000800A
RC[7] = 0x8000000000008009	RC[19] = 0x800000008000000A
RC[8] = 0x000000000000008A	RC[20] = 0x8000000080008081
RC[9] = 0x0000000000000088	RC[21] = 0x8000000000008080
RC[10] = 0x0000000080008009	RC[22] = 0x0000000080000001
RC[11] = 0x000000008000000A	RC[23] = 0x8000000080008008