Hash Functions

28 November 2019 20:49

Hash functions are auxilliary functions in cryptography. They are used, eg, for signatures, MACs, key derivation, RNGs etc

Motivation:

Alice → Bob : K_{pub} ; $s = sig_{K_{priv}}(x)$; (x, s)

RSA : $S = x^d \bmod n$

problem → x is restricted in length. eg $|x| < 256B$

Soln → somehow "compress" the message x prior to signing.

basic protocol for digital signatures using HASH FUNCTIONS

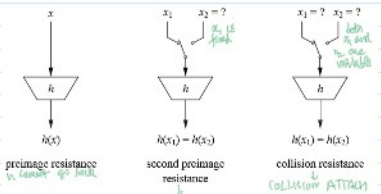
$z = h(x)$ → VDT (z, s) K_{pub} → verify Digital ...

$z = h(x)$; x, s ; $s = sig_{K_{priv}}(z)$

z → is a "FINGERPRINT" of x, or a "MESSAGE DIGEST".

REQUIREMENTS : (for a hash Function)

1. should work with Arbitrary lengths
2. should give fixed, short output lengths
3. Efficient.
4. "PRE IMAGE RESISTANCE" for "ONE-WAYNESS"
5. "2nd PRE IMAGE RESISTANCE"



2nd Preimage attack :

assume x_1 = "Transfer €10 in oscars account"

x_2 = " " €10,000 in " "

and $h(x_1) = h(x_2) = z$

DAMN OSCAR IS GONNA TO BE VERY RICH

COLLISION ATTACK :

x_1, x_2 as in 2nd Preimage Attack

Alice → Oscar → Bob

Alice → Oscar : x_1, x_2 ; Oscar → Bob : K_{pub} ; Bob → Oscar : x_2 ; Oscar → Alice : $z = h(x_2)$; Oscar → Bob : (x_2, s) ; $s = sig_{K_{priv}}(z)$

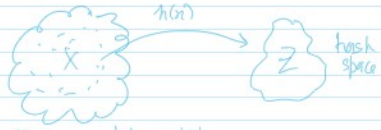
$z = h(x_2)$; (x_2, s) ; ver K_{pub} & s = TRUE

COLLISION ATTACKS and THE BIRTHDAY PARADOX

collision attacks are much harder to prevent than 2nd pre-image attack

Q→ Can we have hash function w/o COLLISIONS ?

Soln : NO



since $|x| >> |z|$, COLLISIONS MUST EXIST by DIRICHLET'S DRAWER PRINCIPLE or PIGEON HOLE PRINCIPLE.

There are only soln is to make COLLISIONS HARD TO FIND.

2nd preimage attack w/ Brute Force :

SHA

SECURE HASH ALHORITHM

→ most popular hash functions in practice

"MD4 FAMILY"

MD4 → 1980s

↓

MD5 → 1991

↓

SHA-1 → 1995

theoritically broken in 2004.

actually broken in 2017.

↓

SHA-2 → 2001 [just SHA-1 on steroids]

↓

SHA-3 competition by NIST

November 2007 → Call for algorithms

October 2008 → 64 submissions.

December 2010 → 5 left → "Round 3" Algorithms

October 2nd, 2012 → KECCAK was selected as SHA3

SHA-3 Requirements by NIST :

arbitrary → SHA-3 → 224/256/384/512

Resistance required for 224 → 112 (birthday paradox)

Same as 3DES

256 → 128

Same as AES.

192, 256 → also AES.

→ HIGH LEVEL VIEW :

sponge construction

Absorption Phase : Input x_i is read-in + processed

Squeezing Phase : Output is produced.

KECCAK PARAMETERS :

State → $b = 25 \cdot 2^l$, $l = 0, 1, \ldots, 6$

$b \in \{25, 50, 100, 200, 400, 800, 1600\}$ → SHA-3

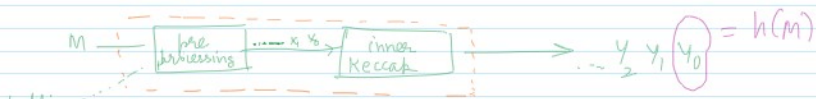
#Rounds → $n_r = 12 + 2l$

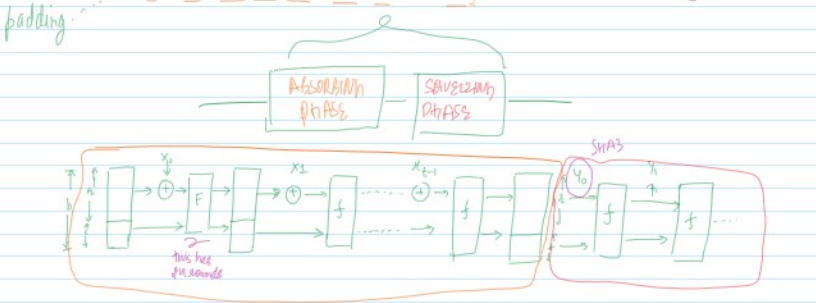
for SHA-3, $l = 6$

$\therefore n_r = 12 + 2(6) = 24$

Output	b	Block size r	Capacity c
224	1600	1152	448
256	"	1088	512
384	"	832	768
512	"	576	1024

note: $r + c = b$

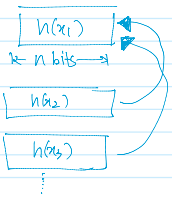




THE f FUNCTION :

b → ROUND 1 → b → ROUND 24 → b

2nd preimage attack w/ Brute Force:



if $|Z| = 2^{80}$, $n = 80$
then attack requires $\approx 2^{80}$ steps.

Collission Attack:
"how many people do i need to invite for a BIRTHDAY PARTY so that atleast 2 of them have same birthday".

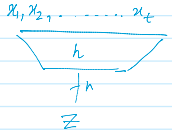
$P(\text{no collision among 2 people}) = 1 - \frac{1}{365}$

$P(\text{" " " 3 people}) = \left(1 - \frac{1}{365}\right)\left(1 - \frac{2}{365}\right)$

$P(\text{" " " t people}) = \prod_{i=1}^{t-1} \left(1 - \frac{i}{365}\right)$

for $t = 23$,

$$p = \prod_{i=1}^{23} \left(1 - \frac{i}{365}\right) \approx 0.507 \approx 50\%$$



$P(\text{no collision}) = \prod_{i=1}^{t-1} \left(1 - \frac{i}{2^n}\right)$

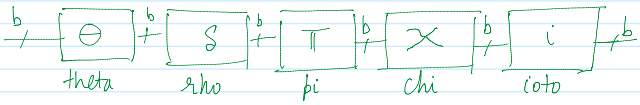
$$t = 2^{\frac{n+1}{2}} \sqrt{\ln\left(\frac{1}{1-\lambda}\right)}$$

$\lambda \rightarrow$ probability for atleast one collision

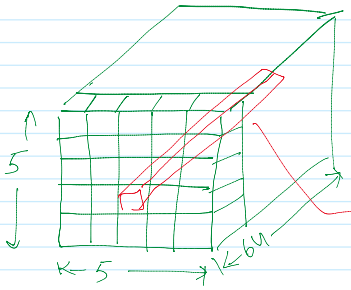
Eg: $n = 80$, $\lambda = 0.50$

then $t = 2^{\frac{81}{2}} \sqrt{\ln 2} \approx 2^{40.5}$

Each Round has 5 steps:



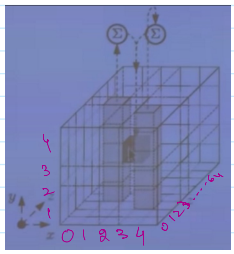
the b = 1600 state can be viewed as a 3-dimensional array.



[] → 1 bit

note: 5 X 5 X 64 = 1600

64 bit (A WORD SIZE REGISTER) with a coordinate (x,y) s.t $0 \le x, y \le 4$



θ (theta):
Each of the 1600 state bits are replaced by the XOR sum of 11 bits
the original bits
⊕ the 5-bit column "to the left"
⊕ the 5-bit column "to the right" and "1 position to the prev" of the bit

	x=3	x=4	x=0	x=1	x=2
y=2	25	39	3	10	43
y=1	55	20	36	44	6
y=0	28	27	0	1	62
y=4	56	14	18	2	61
y=3	21	8	41	45	15

table

ρ(Rho) and π(Pi) step:
operates on words of length 64

Input A [x,y] , x,y → 0,1,2,3,4
Output B [x,y] , "

ρ-step: ROTATE each word is rotated by a fix # of positions.
In pseudo-code,
TEMP [x,y] = rot (A [x,y], r [x,y]) → table entry

π step: Permute the rotate'd words.

B[y, 2x+3y] = TEMP [x,y] , x,y = 0,...4

Eg: input word A[3,1]
ρ → TEMP [3,1] = rot [A[3,1], 55]
π → B[1, 2·3 + 1·3]
⇒ B[1, 9]
⇒ B[1, 4]) mod 5 = TEMP [3,1]

χ (CHI) STEP:
Input B[x,y]
Output A[x,y]

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

i (iota) STEP:
Add constants from constant table to word A[0,0]:

A[0,0] = A[0,0] + RC[i]
i → Round #

Table 1.5 The round constants RC[i], where each constant is 64 bits long and given in hexadecimal notation

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