15/10/22

The AES Cipher - Rijndael.

Input - 128 bit

4×4 array

I_0	I_1	I_2	I_3
I_4	I_5	I_6	I_7
I_8	I_9	I_{10}	I_{11}
I_{12}	I_{13}	I_{14}	I_{15}

State array.

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

↳ output similar array

key also 128 bit — similar 4×4 array.

- 10 rounds
  each round - diff keys
  4 keys each round
  44 subkeys are generated from given master 128 bit key.

represented 4 words. W_0 W_1 W_2 W_3

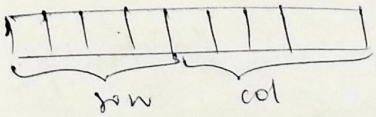
1 pre round computation - 4 keys.

→ Substitute keys:

Similar to S-boxes in DES.

But here it is not reducing.

each word ie., 8 bits



- Shift rows - circular left shift
- Mix columns - random matrix generated in GF(2⁸) · 4×4 multiply with each col of S-box gives corresponding col in op matrix

Byte substitution:

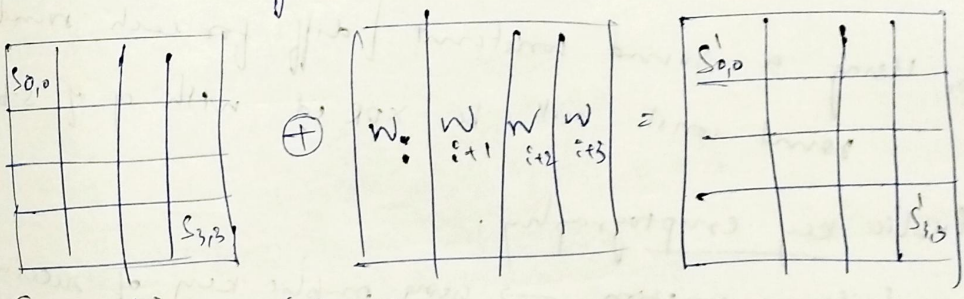
S-box - 16×16

Ex.: $S_{0,0}$ — 0000 | 1010 → 0 | A → row | col

- Shift rows:
  a circular byte shift in each
  - 1st row unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte " "
  - 4th " " 3 byte " "

- decrypt inverts using shifts to right

→ Add round key:



→ Generating sub keys:

AES key expansion

XOR operations with prev words.

→ Variations of AES key expansions

* No Mix columns in round 10.

3

AES Decryption:

Inverse

- decryption is not identical to encryption since steps done in reverse
- More complex than DES.

Implementation aspects.

→ Matrix multiplication in $GF(2^8)$.

17/10

key - 4 words

represented as a 4×4 matrix

each ele - one byte info

→ The fun g defined on w3 [while generating sub keys for each round]

is a three step process

1) 1-byte circular left shift i.e.,

k_{12} k_{13} k_{14} k_{15} become k_{13} k_{14} k_{15} k_{12}

2) S-box

16 × 16

	00	01	02	...	0f
00					
10					
20					
:					
f0					

3) Using a round constant [diff for each round].

round const will be XOR ed with op of S-box.

* Public key cryptography:

while encryption ⟶ using public key of receiver

decryption ⟶ using private key of receiver

Needed — a third party to share/store all public keys of communicating parties.

→ Encryption with public key ⇒ sender private key - encryption

" private key. ⇒ " public key - decryption

→ Diff conventional encryption vs Public-key encryption.

↓

- Sender & receiver agree upon pair of keys.
- One of the two keys have to be kept secret ie., private key.

→ RSA algorithm (Rivest, Shamir, Adleman)

- Public key cryptography.

Alice - sender Bob - receiver public key — e
private key of Bob (d)

- B chooses two large primes p, q & computes $n = pq$.
- B chooses e with $\gcd(e, (p-1)(q-1)) = \gcd(e, \psi(n)) = 1$
  i.e., public key e is relatively prime to n.
- B solves $de \equiv 1 \pmod{\psi(n)}$
- B makes (e, n) public & (p, q, d) secret/private.
- A encrypts M as $C \equiv M^e \pmod{n}$
- Bob decrypts by computing $M \equiv C^d \pmod{n}$

each plain text char is represented as integer.

- Perform encryption & decryption using RSA algo.

$p = 3 \quad q = 11 \quad M = 6$

$n = 3 \times 11 = 33$

$\psi(n) = 2 \times 10 = 20.$

$\gcd(e, 20) = 1 \Rightarrow e = 3$

$de \equiv 1 \bmod 20$

$3d \equiv 1 \bmod 20$

$3d \bmod 20 = 1 \Rightarrow d = 7.$

public = (3, 33) private = (3, 11, 7)

$C = 6^3 \pmod{33}$

$= 18$

$M = 18^7 \bmod 33$

$M = 18^2 \times 18^2 \times 18^2 \times 18 \bmod 33$

$= 6$

18/11/22

Fast exponentiation algorithm :

$a^b \bmod n$

Express b as $b_k b_{k-1} \cdots b_0$ ⇒ in bits

c ← 0 f ← 1

for i ← k down to 0 ⇒ c ← 2×c

f ← (f × f) mod n

if $b_i = 1$

then c ← c+1

f ← (f × a) mod n

return f

c ⇒ gives the power. i.e., 560.

Example:

$7^{560} \bmod 561$

b = 560 = 1000110000

b_k b_0

c = 0

f = 1

i	9	8	7	6	5	4	3	2	1	0
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
f	7	49	157	526	160	241	298	166	67	1

→ Given p = 11 q = 13 e = 11

encrypt M = 7. calculate 'c' using fast exponentiation algorithm

$C = 7^{11} \bmod (11 \times 13) = 7^{11} \bmod 143$.

11 ← [illegible] 1011

t = 1 n = 143 a = 7

i	3	2	1	0
bi	1	0	1	1
c	1	2	5	11
f	7	49	~~113~~ 98	~~56~~ 106

→ cipher.

$de = 1 \bmod \varphi(n)$

$d(11) \equiv 1 \bmod 10 \times 12$

$\equiv 1 \bmod 120.$

$11\,d \bmod 120 = 1$

$\boxed{d = 11}$

* Diffie-Hellman key exchange:
uses discrete logarithms

$x \equiv a^i \bmod p$

$i \equiv \log_a x$

1, Alice & Bob share a prime number q and an integer α, such that $\alpha < q$ and α is primitive root of q.

Alice

2, Alice generates a private key x_A such that $x_A < q$

3, Alice calculates a public key

$y_A = \alpha^{x_A} \bmod q$

Bob

Bob generates a private key x_B such that $x_B < q$.

Bob calculates a public key

$y_B = \alpha^{x_B} \bmod q$

y_B → Bob receives Alice public key y_A

y_A → Alice receives Bob public key y_B

Alice calculates shared secret key as

$$K = (Y_B)^{X_A} \bmod q$$

Bob calculates

$$K = Y_A^{X_B} \bmod q$$

X_A - private key of Alice

Y_A - public " "

Ex:- common prime $q = 353$, the primite root $\alpha = 3$

$X_A = 97$ find Y_A. $X_B = 233$ Find Y_B & find shared secret key K.

$$Y_A = \alpha^{X_A} \bmod q$$

$$Y_A = 3^{97} \bmod 353$$

$$Y_B = 3^{233} \bmod 353$$

$$Y_B = 3^{233}$$

97 in bits ↓

i	6	5	4	3	2	1	0
bi	1	1	0	0	0	0	1
f	X 3						

f = 1

a = 3

26/10

Elgamal public key encryption & decryption:

Global elements

q prime numbere

α $\alpha < q$, primitive root of q.

key generation (Alice)

Select private $x_A < q-1$

calculate $Y_A = \alpha^{x_A} \bmod q$.

Public key $\{q, \alpha, Y_A\}$

Private key $\{x_A\}$.

Encryption (Bob)

Plain text $m < q$ ⟶ message is represented as an integer less than q.

Select random $k < q$

calculate $K = Y_A^k \bmod q$

cal $C_1 = \alpha^k \bmod q$

" $C_2 = KM \bmod q$

cipher text (C_1, C_2).

* Encryption is done by Bob using public key of Alice.

Decryption (Alice)

(C_1, C_2)

$K = C_1^{x_A} \bmod q$

Plaintext $M = (C_2 K^{-1}) \bmod q$.

Ex:- $q = 19$

$\{2, 3, 10, 13, 14, 15\}$ ⇒ primitive roots of $q = 19$

Let choosen primitive root, $\alpha = 10$. {can take with any α}.

$x_A = 5$ {less than $19-1$}

$Y_A = 10^5 \bmod 19$

$Y_A = 3$

Public key : $\{19, 10, 3\}$ } - Alice.

Private " : $\{5\}$

Let $M = 17$ $\{< 19\}$

let $k = 6$ $\{< 19\}$ {Bob}.

$K = 3^6 \bmod 19 = 7$

(key)

ciphertext $C_1 = 10^6 \bmod 19 = 11$

$C_2 = (7 \times 17) \bmod 19 = 5$

17 encrypted are $(11, 5)$

Decryption:-

$K = 11^5 \bmod 19 = 7$

plaintext $M = 5 K^{-1} \bmod 19$

K^{-1} ⇒ multiplicative inverse of 7 mod 19.

$= 11$ $\quad 7 \times 11 \bmod 19 = 1$

$M = 5 \times 11 \bmod 19$

$M = 17.$

Ex:- Let $q = 71$ [common prime]

(Q1) primitive root $\alpha = 7$

If Bob as a public key $Y_B = 3$

& Alice chooses a random integer $k = 2$

what is cipher text of $M = 30$.

(Q2) If Alice chooses a diff value of k, so that encoding of $M = 30$ is $C = (59, C_2)$ what is C_2

P, Q, R = points on elliptic curve

$P + Q + R = 0$

1. O - identity ele
   $O = -O$
   $P + O = P$
2. $P = (x, y)$ $-P = (x, -y)$
   $P + (-P) = 0$ ↳ mirror image.
3. $P + Q = -R$ ⇒ R - third point
   i.e. P, Q points on elliptic curve
   R - 3rd point where PQ line intersects
   i.e. mirror image $-R$ is $P + Q$.
   if tangent, at P or Q then
   $P = R$ or $Q = R$
4. Point doubling: Q
   tangent at Q meet curve at S
   $Q + Q + S = 0$
   $2Q = -S$

(Q1)

Key generation :- (by Bob)

$Y_B = \alpha^{x_B} \bmod q$.

$3 = 7^{x_B} \bmod 71$

$x_B =$

⇒ $x_A = 6$

$Y_A = 7^6 \bmod 71$

$Y_A = 2$

Public key = $\{71, 7, 2\}$

Private key = $\{6\}$.

→ Encryption :-

$M = 30$.

$k = 2$.

$K =$

→ Elliptic curve — cubic polynomial.

* Elliptic curves :

↳ Weierstrass equation

represented by 7.

$y^2 = x^3 + ax + b$ defined F_7

→ Each curve symmetric about $y=0$

$y^2 = x^3 + 3x + 1$ (a, b)

↳ $E(F_7)$ $(3, 1)$

↓

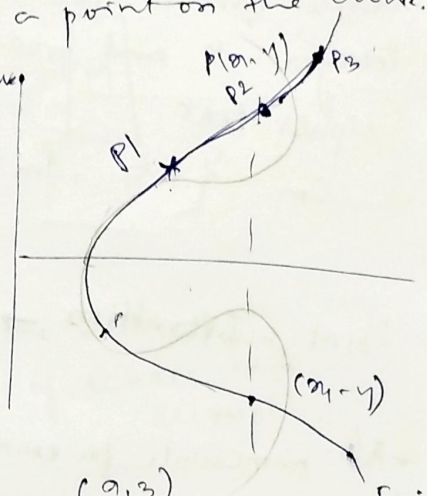
defined over finite field F_7.

The message is represented as a point on the curve.

- discrete log prob in elliptic curve is different

$Q = nP$ → Find n

$n = \log_P Q$



- Point addition :-

$P_3 = P_1 + P_2$

the intersection of line drawn by two points P1, P2 and the curve is the point addition P3.

(9, 3)

(9, -3)

$9 \equiv -3 + 7 \bmod 7$

is represented as (9, 9)

F_7

- Point doubling :-

Given one point P,

draw tangent at P, of term 2P.

if 2P ~~or other~~ find a such that nP tangent meets the curve again $Q = nP$

- Identity point in elliptic curves is infinity point

→ Equations of point addition & point doubling.

2/11/22

Elliptic curve encryption / decryption

M (x, y)

$P_m \in E_q(a, b)$

$y^2 = (x^3 + ax + b) \bmod q$

Plain text : P_m (point on elliptic curve)

$P_A = n_A \times G$

$P_B = n_B \times G$ → generator

Encryption

Choose random positive integer k and produce cipher text

$C_m = \{kG, P_m + kP_B\}$

Decryption

$P_m + kP_B - n_B(kG)$

$= P_m + k(n_B G) - n_B(kG)$

$= P_m$

Point subtraction → negation of y coordinate mod q and addition.

- No numericals for exam
- By encryption & decryption, we achieve confidentiality.

* Authentication Protocols:

1. Message encryption
2. Message Authentication Codes (MAC)
3. Hash Functions.

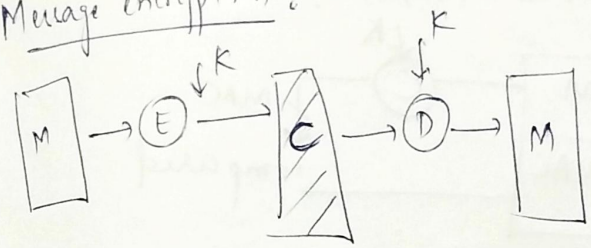
→ ciphertext produced itself is authenticator (of sender)

→ Uses some func on plain text msg itself to generate some fixed len code (MAC) using a key K.

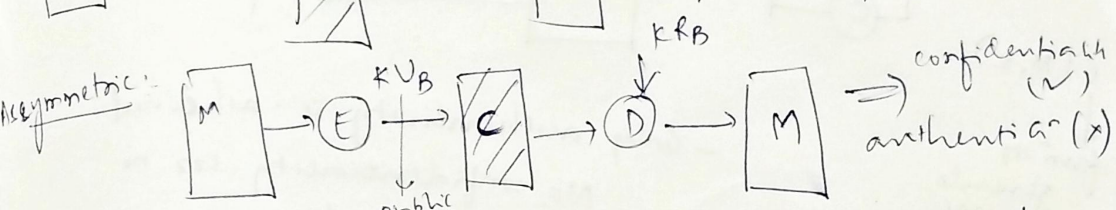
$C(M, K) = MAC$ → MAC is appended to M & is sent to receiver.

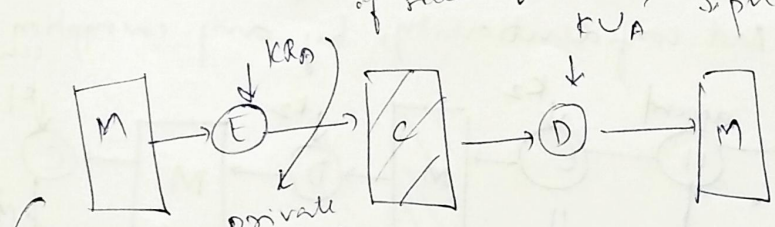
→ Also generates a fixed length code but doesn't use any key. $H(M) = h$, ⇒ fixed len code ↳ acts as authenticator.

Message encryption :



Symmetric

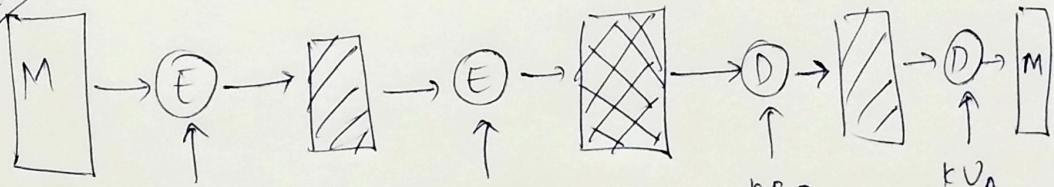




authentication (✓)
confidentiality (✗) → anyone with public key of sender can decrypt.

So, combine both these schemes to provide both authentication & confidentiality.

Dual encryption



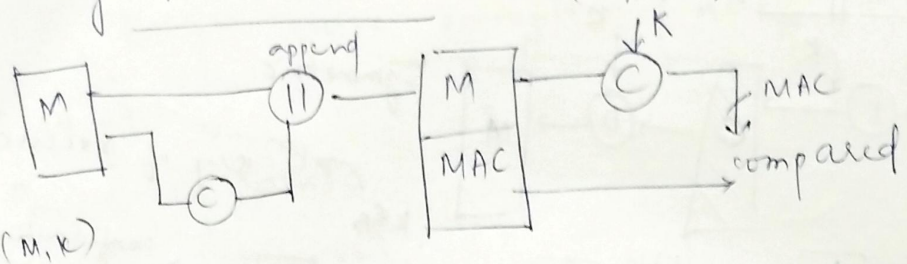
KRA Private key of sender → authentication.

KUB public key of receiver → conf

KRB private key of receiver

KUA public key of sender → confidentiality.

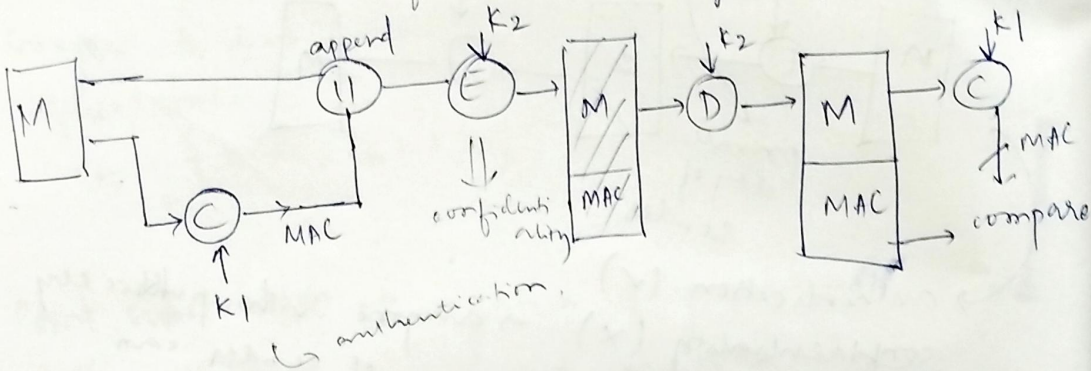
* Message Authentication Code (MAC) :



C(M, K)
↓
fun to generate MAC

- Only authenticity is achieved
No confidentiality coz no encryption.

- Add confidentiality, by any encryption scheme.



↳ authentication.

* Hash functions : Same like MAC, but hash function doesn't need any key to generate hash code.

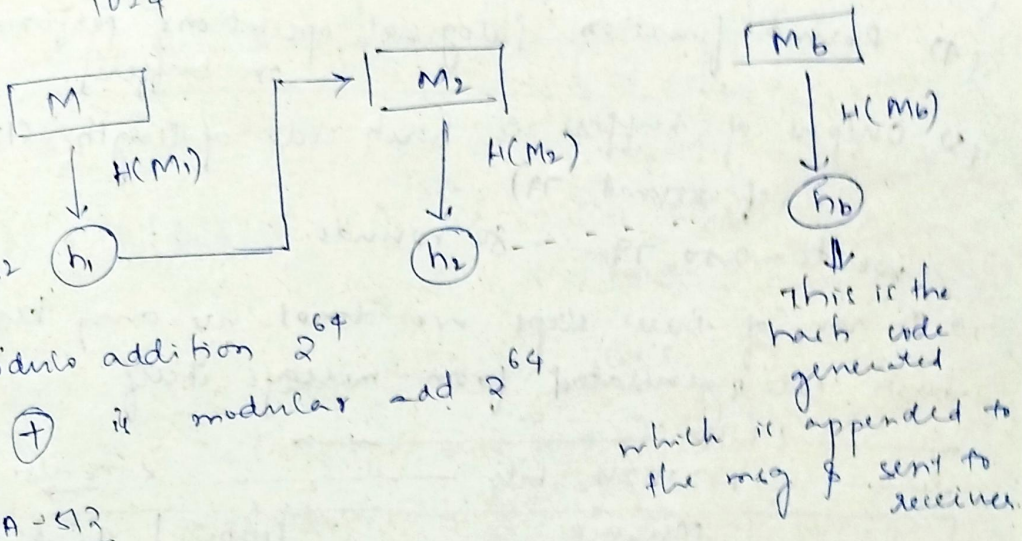
7/11/23

Authentication Protocole :- (Continuation)

Secure Hash Algorithm (SHA):

Hash code generated is
- 128 bits - SHA1
- 256 bits - SHA 256
- 512 - SHA 512

Block ciphers - chaining mode
- 1024 bits - block size



⇒ Modulo addition 2^{64}

⊕ is modular add 2^{64}

* SHA - 512
- 80 rounds of operation
- M - 1024 bits
- 64 bit word → round ← constant

ip

Hash code - 512

Buffers needed : (a, b, c, d, e, f, g, h) $\frac{512}{64} = 8$

↳ each will store a 64 bit length code

↳ also carries intermediate results from one round to other.

Steps in SHA - 512 :

(1) Pad the bits 1000.... so that M is 128 bits less than multiple of 1024 bits.

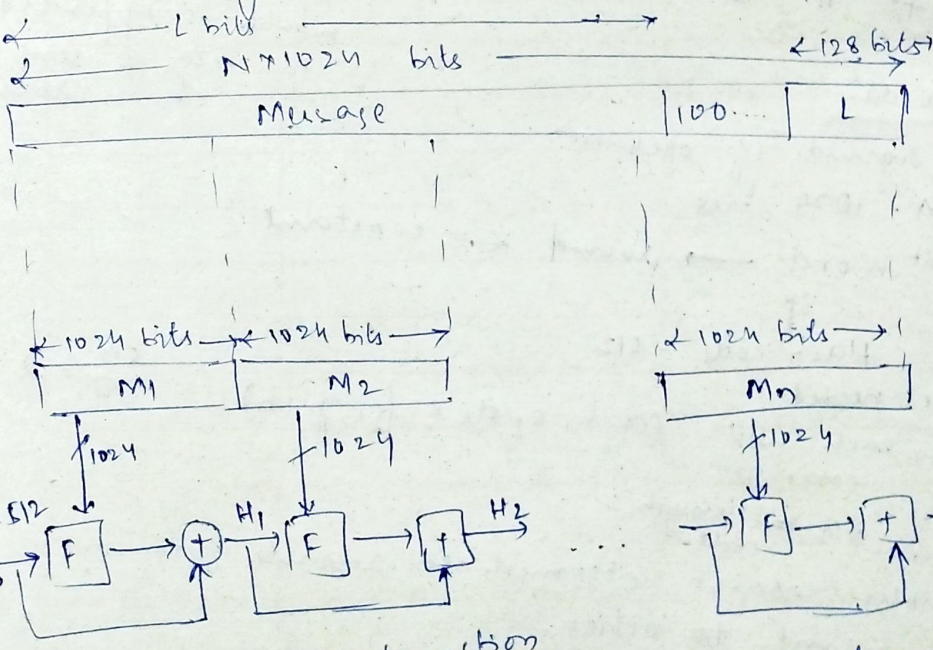
(2) [From original msg M, ~~a~~ ~~bits~~ ~~bit~~]
Append 128 bits representation of M such that the M in multiple of 1024 bits.

(3) Initialize the buffer a-h (64 bits)

(4) Round function [logical operations performed on buffers].

(5) Output of buffers is hash code of length 512.
[(of round 79)
rounds 0 to 79 - 80 rounds.

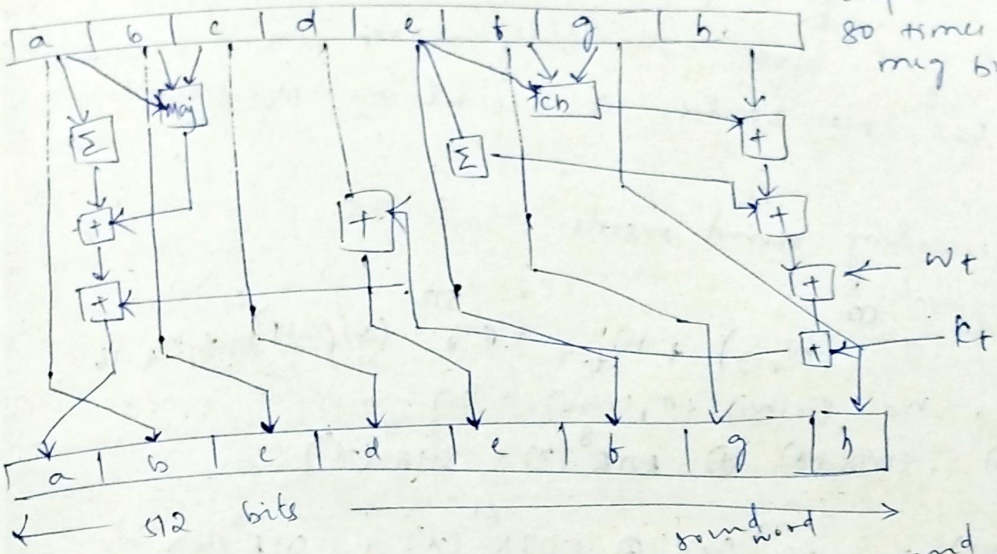
→ In any of these steps we donot use any key, hash code is generated from message itself.



→ F - compression function
For each block F performs 80 rounds of computation
each round has a constant, word (generated from M)

SHA-512 Round Function :

this is repeated for 80 times for each msg block



$$T_1 = h + Ch(e,f,g) + \sum_1^{512} e + W_t + K_t$$

W_t ← round word ; K_t ← round const

↳ generated from msg of 1024 bits

$h = g$ $\quad a = T_1 + T_2$

$g = f$ $\quad t$ ← step number $\quad 0 \le t \le 79$

$f = e$ $\quad Ch(e,f,g) = (e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$

$e = d + T_1$ $\quad$ ↳ conditional funⁿ

$d = c$ $\quad$ if e is TRUE then f else g.

$c = b$

$b = a$ $\quad Maj(a,b,c) = (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$

$$T_2 = \sum_0^{512} a + Maj(a,b,c)$$

$$\sum_0^{512} a = ROTR^{28}(a) \oplus ROTR^{34}(a) \oplus ROTR^{39}(a)$$

$$\sum_1^{512} e = ROTR^{14}(e) \oplus ROTR^{18}(e) \oplus ROTR^{41}(e)$$

$ROTR^n(x)$: circular right shift of 64 bit x by n shifts

K_t . round const [predefined in table] 80 constants.

Generating W_t :

separately for each block of msg

For first block - M itself is W_t

For next blocks - it depends on prev blocks.

9/11/22

Generating round words:

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

Ex:- $W_{16} = W_0 + \sigma_0(W_1) + \sigma_1(W_{14}) + W_9$

$\sigma_0^{512}(x) = ROTR^1(x) \oplus ROTR^8(x) + SHR^7(x)$

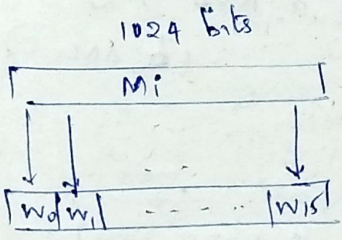
$\sigma_1^{512}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) + SHR^6(x)$

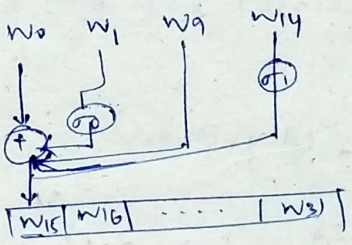
$ROTR^n(x)$ = circular right shift of x by n bits

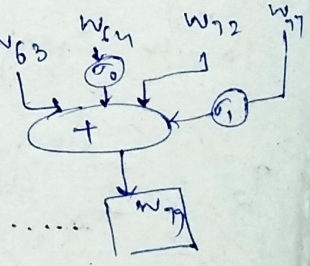
$SHR^n(x)$ = right shift of x by n bits padding 0's on left.

M_0 - msg itself

$W_0 - W_{15}$

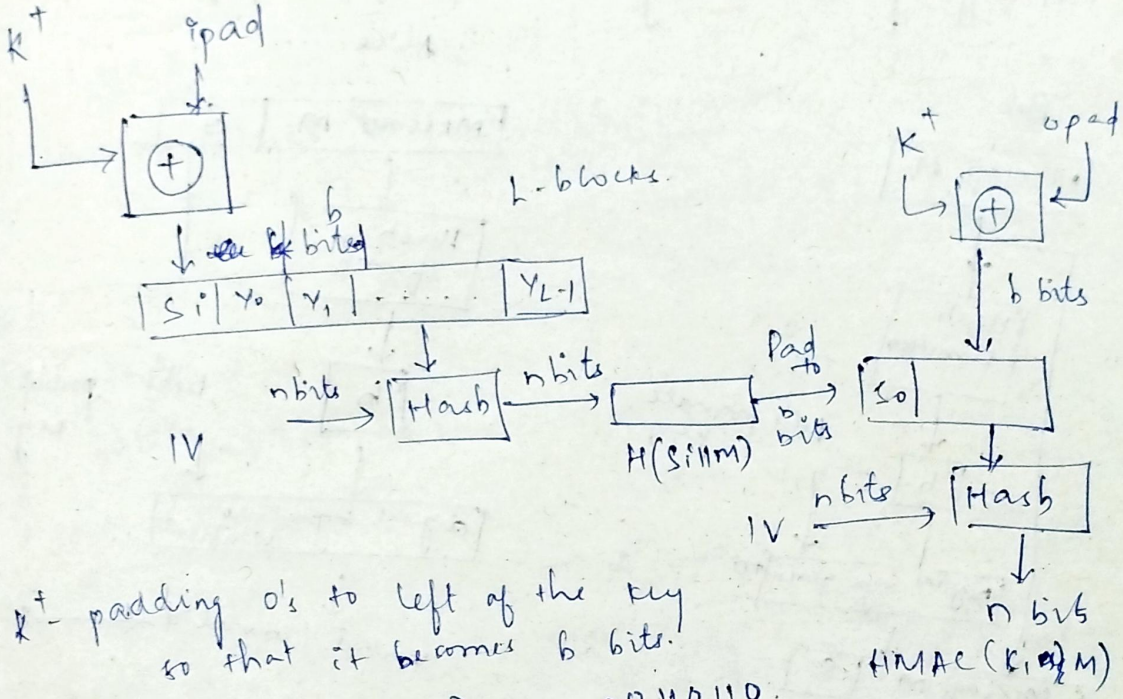






* Hash MAC (HMAC) :

combines properties of hashing & MAC
Inbetween/generating a MAC we use hashing algorithm.



K+ - padding 0's to left of the key so that it becomes b bits.

ipad - 36 (hexadecimal) ⇒ 00110110.

opad - 5C i.e., 01011100

Steps in HMAC :

(1) Append 0's to the left of key K to create a b bit string K+

(2) XOR K+ with ipad to produce a b bit block Si.
(ipad - 36 repeated b/8 times)

(3) Append M to Si

(4) Apply embedded hash function to the above stream generated

(5) XOR K+ with opad. (5C repeated b/8 times) to generate a block So.

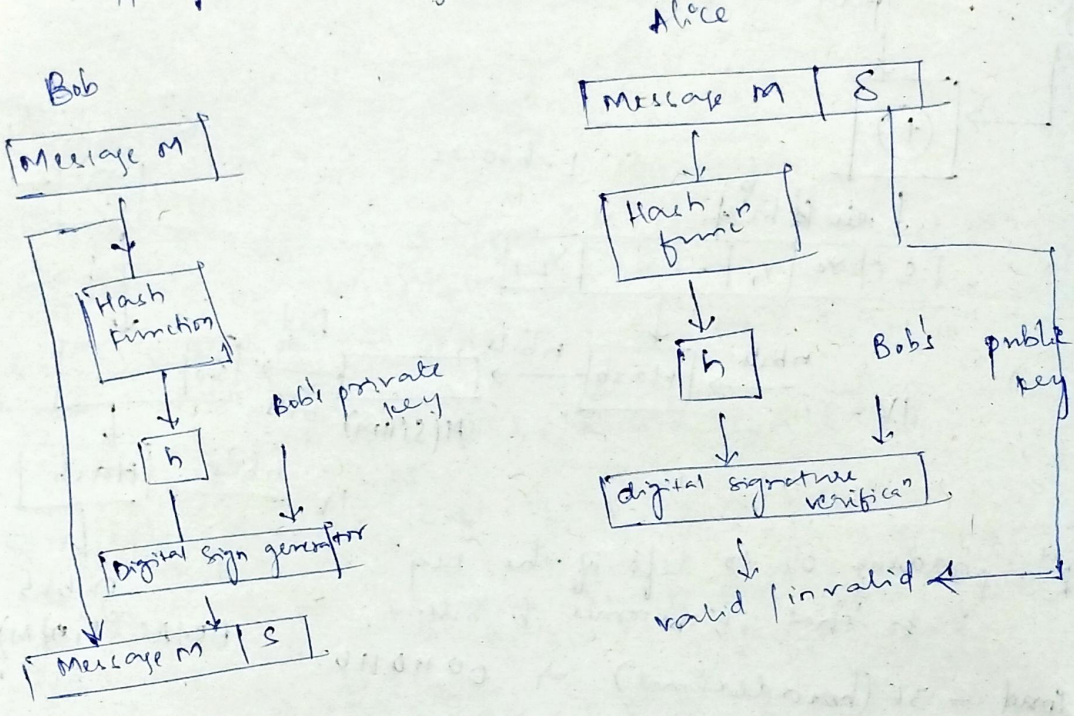
(6) Append the hash result from the prev step to So.

(7) Apply the embedded hash function to the stream generated in the prev step which gives a code represented as HMAC(K,M)

* Digital Signature :

DSS - Digital Signature Standard.

- Encrypting the message with private key of sender



q : prime number

α : primitive root

$\alpha, \alpha^2, \ldots, \alpha^{q-1} \bmod q$

1. For any integer m, $\alpha^m \equiv 1 \bmod q$ iff $m \equiv 0 \pmod{q-1}$
2. For any integer i, j, $\alpha^i \equiv \alpha^j \bmod q$ iff $i \equiv j \pmod{q-1}$

Elgamal Digital Signature algorithm :

B.Com

Global elements :

q, α

User A

1. random integer X_A

$1 < X_A < q-1$

2. compute $Y_A = \alpha^{X_A} \bmod q$.

private $\{X_A\}$

public $\{q, \alpha, Y_A\}$

Sign Generation :

1. message M
2. hash $m = H(M)$

such that

$0 \le m \le q-1$

3. choose random k

$1 \le k \le q-1$

$\gcd(k, q-1) = 1$

4. Compute $S_1 = \alpha^k \bmod q$
5. Compute $k^{-1} \pmod{q-1}$
6. Compute

$S_2 = k^{-1} m - (X_A S_1) \bmod q-1$

Signature pair (S_1, S_2)

Verification :

1. Compute $V_1 = \alpha^m \bmod q$.
2. Compute

$V_2 = (Y_A)^{S_1} (S_1)^{S_2} \bmod q$.

Check $V_1 = V_2$.

$\Rightarrow \alpha^m \bmod q = (Y_A)^{S_1} S_1^{S_2} \bmod q$.

$\alpha^m \bmod q = \alpha^{X_A S_1} (\alpha^k)^{S_2} \bmod q$.

$\alpha^{m - X_A S_1} \bmod q \equiv \alpha^{k S_2} \bmod q$

$m - X_A S_1 = k S_2 \pmod{q-1}$

$m - X_A S_1 \equiv k k^{-1} (m - X_A S_1) \bmod q-1$

1

Ex:- $q = 19$

$\alpha = \{2, 3, 10, 13, 14, 15\}$

$x = 10$

1. $X_A = 16$

2. $Y_A = 10^{16} \bmod 19$

$= 4$

A's private key $\{16\}$

A's public key $\{19, 10, 4\}$

Signing :-

1. A choose $k = 5$

$\gcd(5, 19-1) = 1$

2. $S_1 = \alpha^k \bmod q = 10^5 \bmod 19$

$= 3$

3. $S_2 = k^{-1}(m - X_A S_1) \bmod q - 1$

$= 11\,(14 - 16 * 3) \bmod 18$

$= 4$

Verification :

$V_1 = \alpha^m \bmod q$

$= 10^{14} \bmod 19$

$= 16$

$V_2 = (Y_A)^{S_1} (S_1)^{S_2} \bmod q$

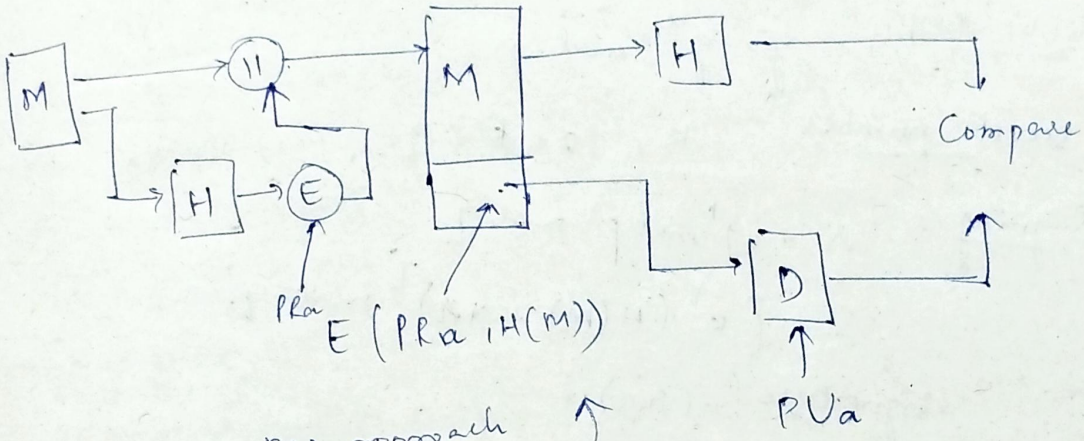
$= 4^3 \cdot 3^4 \bmod 19$

$= 16.$

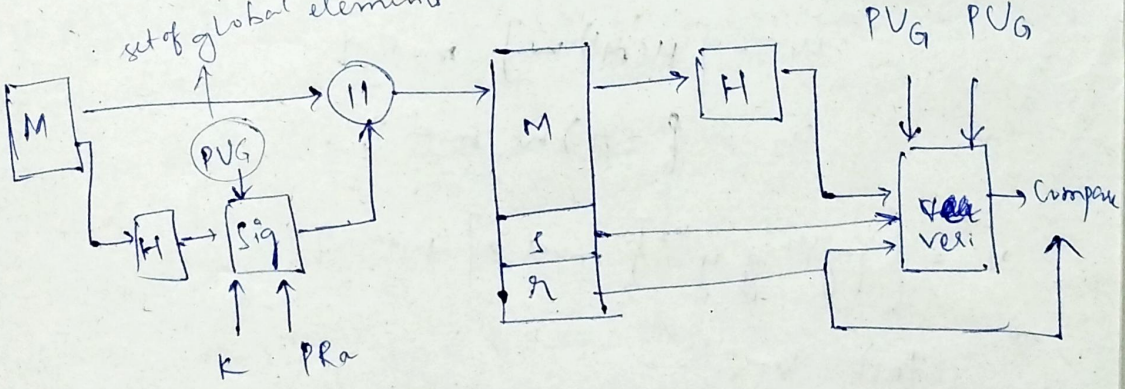
14/11/22

DSA (or DSS) : Digital Signature Standard

(NIST)



RSA approach of DSA



PUG (global public key) :

p prime num

$$2^{L-1} < p < 2^{L}$$

$512 \le L \le 1024$, L multiple of 64.

q prime divisor of p-1 $\boxed{p-1 \mid q.}$

$$2^{N-1} < q < 2^{N}$$

bit length N bits

$g = h^{(p-1)/q} \bmod p$ is exponent mod p

$1 < h < p-1$

such that $h^{(p-1)/q} \bmod p > 1$. (such that $g > 1$)

User's private key:

x, $0 < x < q$

$H(M)$ – hash code using SHA-1

Public key: $y = g^x \bmod p$

secret number: k, $0 < k < q$

Signing:

$$r = (g^k \bmod p) \bmod q$$

$$s = [k^{-1}(H(M) + xr)] \bmod q$$

signature $= (r, s)$

Verifying:

$$w = (s')^{-1} \bmod q$$

$$u_1 = [H(M')w] \bmod q$$

$$u_2 = (r')w \bmod q$$

$$v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$$

Test $v = r'$

(Q)

DSA specifies in the signature generation process if $s = 0$, a new value of k should be generated & new signature has to be calculated. State reasons.

Comm. → We need to compute $(s')^{-1}$ if $s = 0$.

while verification $w = (s')^{-1} \bmod q$

i.e., verification doesn't exist for $s = 0$.

ECDSA

Global domain parameters

q prime num

a, b $y^2 = x^3 + ax + b$

G base point $G = (x_g, y_g)$

n order of G

$nG = \infty$

Key generation :

1. Random d, $d \in [1, n-1]$.
2. $Q = dG$.
3. Public key Q, private key d.

Signing :

1. random k, $k \in [1, n-1]$
2. compute $P = kG$

   $r = x \bmod n$

   if $r = 0$ go to step 1
3. Compute $t = k^{-1} \bmod n$.
4. Compute $e = H(m)$

   SHA - 2 or SHA -
5. $s = k^{-1}(e + dr) \bmod n$

   if $s = \infty$ then go to step 1
6. r, s

Verifying :

1. r, s $[1, n-1]$
2. $e = H(m)$
3. $w = s^{-1} \bmod n$
4. $u_1 = ew$

   $u_2 = rw$
5. $X = u_1 G + u_2 Q$
6. if $X = \infty$ reject signature

   else

   $v = x_1 \bmod n$.
7. iff $v = r$

Problems - 13.5, 13.6 in textbook 4th edition.

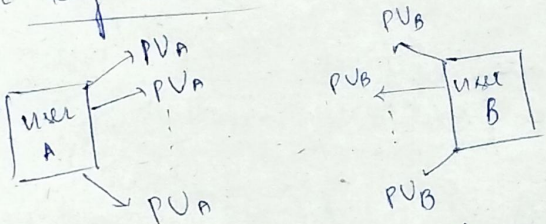
13.3

18/11/22.

Public key distribution :

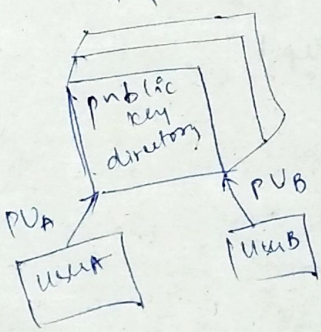
1. Public announcement
2. Public key directory
3. Public key authority
4. Certifying authority

Public key announcement :



- Broadcasting to all users in network
- No use of trusted third party.

Public key directory :



Public key authority :

PU_{Auth} PR_{Auth}

A → B

(1) A → PKA Request || T_1
Request : PU_B

(2) PKA → A

$E_{PR_{Auth}}\{PU_B \| Request \| T_1\}$

(3) A → B $E_{PU_B}[ID_A \| N_1]$

N_1 ↓ Nonce

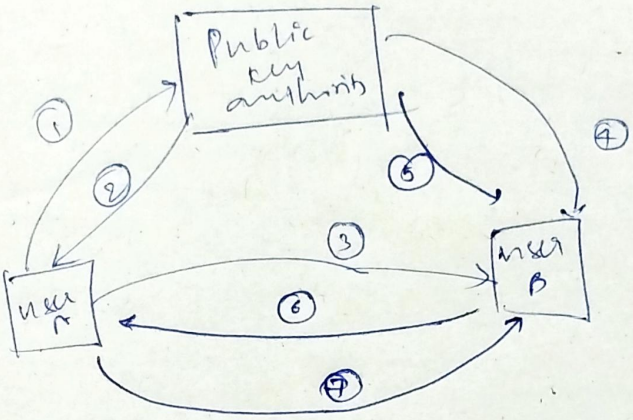
↓ Identity of A

(4) B → PKA Request || T_2 Request : PU_A

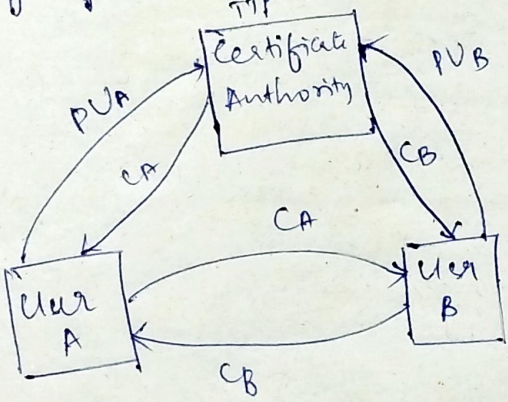
(5) PKA → B $E_{PR_{Auth}}\{PU_A \| Request \| T_2\}$

(6) B → A $E_{PU_A}[ID_B \| N_1 \| N_2]$

(7) A → B $E_{PU_B}(N_2)$



Certifying Authority :-



$C_A = E_{PR_{Auth}} \{ ID_A \| PU_A \| T_1 \}$

$C_B = E_{PR_{Auth}} \{ ID_B \| PU_B \| T_2 \}$.

Advantages :