Module 2

RSA Operations

Three RSA operations:

Key Generation

Encryption

Decryption

Key Generation

Step 1
Selected two distinct prime numbers a and b. Prime numbers are selected randomly with same memory size
Step 2
Computing n = a * b
Step 3
Calculating Euler’s totient function, Ø(n) = (a-1) * (b-1)
Step 4
‘e’ is an integer, 1 < e < Ø(n) and greatest common divisor of e , Ø(n) is 1. Now e is released as Public-Key exponent
Step 5
d = e-1(mod Ø(n)) i.e., d is multiplying the inverse of e mod Ø(n)
Step 6
d is retained as a component of the private key, so that d * e = 1 mod Ø(n)
Step 7
The public key has modulus n and the public exponent e (e, n)
Step 8
The private key contains modulus n and the private exponent d, that is to be maintained as a secret (d, n)

Encryption

Step 1
Transmitting the Public- Key (n, e), which one need to store
Step 2
Data of client are now mapped onto an integer by utilizing an accepted reversible protocol, labeled as a scheme of padding
Step 3
Encryption of data occurs and the resultant cipher text(data) C is $\mathbf{c} = \mathbf{m}^{\mathbf{e}} \pmod{n}$
Step 4
The stated text of cipher type rather data which got encrypted are then stored at the service provider of cloud

Decryption

Step 1
Cloud server response based on the request
Step 2
Cloud service verifies the authenticity of the decrypted data
Step 3
Computing the decryption process → $m = c^d \pmod{n}$
Step 4
'm' is original data

Example:

suppose RSA prime numbers are $p=3$, $q=11$, $e=3$, $m=00111011$

Solution:

Step 1: Compute modulus n

$$n = p*q$$
$$n = 3*11$$
$$\boxed{n = 33}$$

Step 2: Compute $\phi(n)$

$$\phi(n) = (p-1)*(q-1)$$
$$= (3-1)*(11-1)$$
$$= 2*10$$
$$\boxed{\phi(n) = 20}$$

Step 3: Compute encryption Key e.

$$\gcd(e, \phi(n)) = 1$$

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

$e = 3$ = public Key.

Step 4: Compute Decryption Key

$$d = e^{-1} \bmod \phi(n)$$

$$= 3^{-1} \bmod 20$$

$$\boxed{d = 7}$$

Step 5: Encryption

$$\boxed{C_i = m_i^e \bmod n}$$

Step 6: Decryption

$$\boxed{m_i = C_i^d \bmod n}$$

$$\boxed{m = \underbrace{001110}_{\text{Block 1}} \mid 11}$$

Block 2

$$\boxed{000011}$$

append zeros

NOTE: plain text M is divided into Block size of 6 bits as number of bits required to represent $m = 33$ requires 6 bits in binary.

Encryption	Decryption.
$C_i = m_i^e \bmod n$	$m_i = C_i^d \bmod n$
$m_1 = \underbrace{001110}$,	$d = 7$
$m_1 = 14$.	Replace C Value computed
$C_1 = 14^3 \bmod 33$	$m_1 = 5^7 \bmod 33$.
$\boxed{C_1 = 5}$	$\boxed{m_1 = 14}$
$C_2 = m_2^e \bmod n$	$m_2 = C_2^d \bmod n$.
$m = \underbrace{000011}_{}$.	C_2 Computed is 27 substitute C, d & n value
$m_2 = 3$	$m_2 = C_2^d \bmod n$
$C_2 = m_2^e \bmod n$	$= 27^7 \bmod n$.
$= 3^3 \bmod n$	$= (27^5 \bmod 33 \times 27^2 \bmod 33) \bmod 33$
$\boxed{C_2 = 27}$	$\boxed{m_2 = 3}$

Performance

Time Complexity

Both encryption and decryption involve repeated multiplications (modulo n) of b-bit numbers. Unoptimized multiplication of two b-bit numbers and reduction modulo n (division), both take O(b2) time.

Speeding Up RSA

Decryption of cipher text c can be speeded up by computing c, c2, c4, c8, etc., up to a maximum of b terms. Elements are multiplied in this series whose positions correspond to 1's in the binary representation of the decryption key d. Also referred to as "Square and Multiply."

Software Performance

The Java programming language has a number of APIs of relevance to cryptography. These include APIs for key generation and encryption/decryption, message digests, and digital signatures.

•Applications

Providing message confidentiality through encryption is an application of public key cryptography.

The principal drawback of public key cryptography is speed, while the principal drawback of secret key cryptography is key management.

Several other uses of public key cryptography- used to generate a digital signature that provides message integrity and authentication together with non-repudiation.

•Practical Issues

Generating Primes

Side Channel and Other Attacks

Several ways in which RSA may be attacked:

Modulus Factorization

Small Exponent Attack

Side Channel Attacks

•Public Key Cryptography Standard (PKCS)

The Public Key Cryptography Standard (PKCS # 1) specifies, among other things, the format of each block to be encrypted by RSA.

Cryptographic Hash

INTRODUCTION

- ***Definition:*** A hash function is a deterministic function that maps an input element from a larger (possibly infinite) set to an output element in a much smaller set.
- The input element is mapped to a ***hash value.***
- For example, in a district-level database of residents of that district, an individual's record may be mapped to one of 26 hash buckets.
- Each hash bucket is labelled by a distinct alphabet corresponding to the first alphabet of a person's name.
- Given a person's name (the input), the output or hash value is simply the first letter of that name (Fig. 7.1).
- Hashes are often used to speed up insertion, deletion, and que rying of databases.

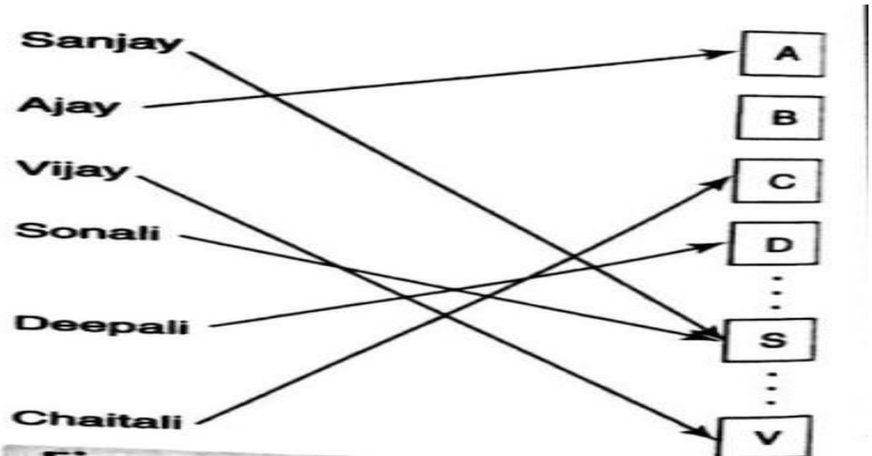


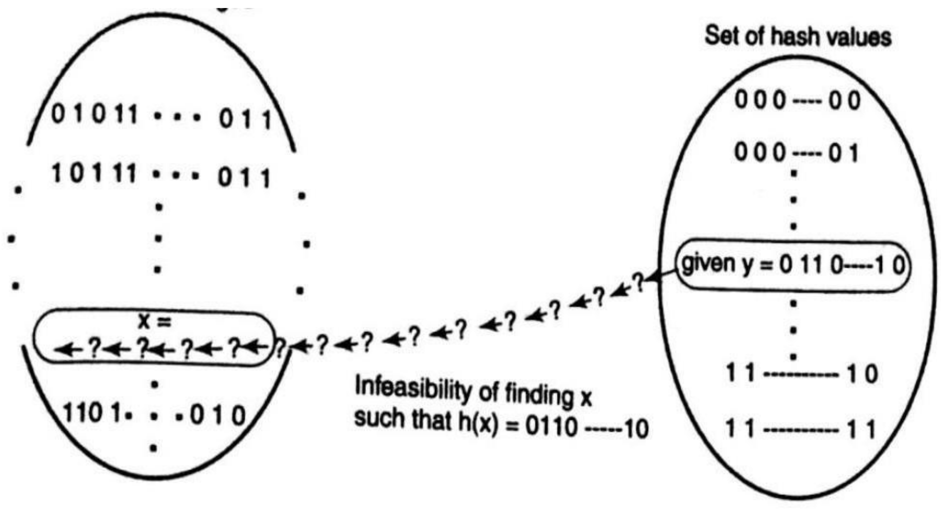
Figure 7.1 *Elementary hash function*

- ***In the example above, two names beginning with the same alphabet map to the same hash bucket and result in a collision.***

PROPERTIES

Basics

- A cryptographic hash function, ***h(x),*** maps a binary string of arbitrary length to a fixed length binary string.
- The properties of *h* are as follows:
    1. **One-way property.** Given a hash value, ***y*** (belonging to the range of the hash function), it is computationally infeasible to find an input x such that ***b(x) =*** **y**
    2. ***Weak collision resistance.*** Given an input value x1, it is computationally
    infeasible to find another input value x2 such that ***h(x1) = h(x2)***
    3. **Strong collision *resistance.*** It is computationally infeasible to find two input values x1 and no x2 such that h(x1)=h(x2)
    4. ***Confusion + diffusion.*** If a single bit in the input string is flipped, then each bit
    of the hash value is flipped with probability roughly equal to 0.5.



(a) Illustrating 1-way property

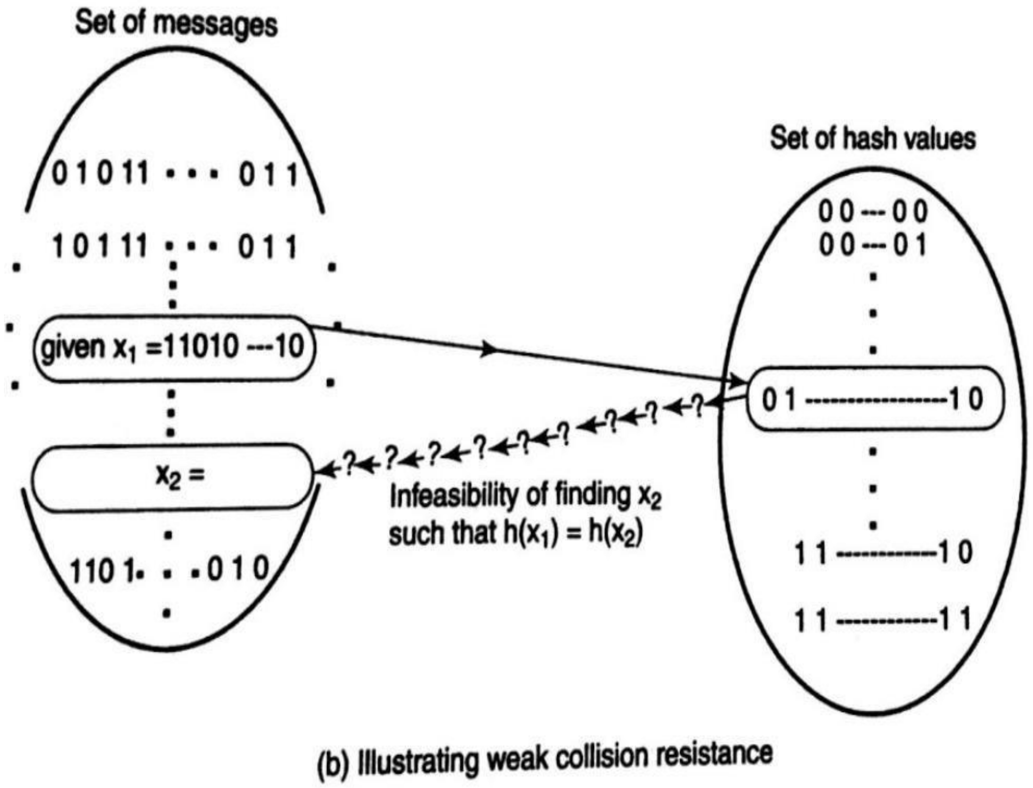


Figure Properties of the cryptographic hash

- There is a subtle difference between the two collision resistance properties.
- In the first, the hash designer chooses x1 and challenges anyone to find an x2, which

  maps to the same hash value as of x1. This is a more specific challenge

  compared to the one in which the attacker tries to find and x2 such that

  h(x1)= h(x2).
- In the second challenge, the attacker has the liberty to choose x1.

Attack Complexity Weak

Collision Resistance

- How low long would it take to find an input, x, that hashes to a given value y?
- Assume that the hash value is w bits long. So, the total numb er of

  possible hash values is 2^w
- brute force attempt to obtain x would be to loop through the following operations

```
do
{
    generate a random string, x'
    compute h(x')
}
while (h(x') != y)
return (x')
```

- assuming that any given string is equally likely to map to any one of the 2^{W} hash values, it follows that the above loop would have to run, on the average, 2^{w-1} times before finding an x' such that h(x') = y.
- A similar loop could be used to find a string, x2, that has the same hash value as a given string x1.

Strong Collision Resistance

- A Brute-force attack on strong collision-resistance of a hash function involves looping through the program in Figure.
- Unlike the program that attacks weak collision resistance, this program terminates

  when the hash of a newly chosen random string collides

  with any of the previously computed hash values.

```
// S is the set of (input string, hash value) pairs
// encountered so far

notFound = true
while ( notFound )
{
    generate a random string, x'
    search for a pair ( x, y) in S where x = x'
    if ( no such pair exists in S )
    {
        compute y' = h(x')
        search for a pair (x, y) in S where y = y'
        if ( no such pair exists in S )
            insert (x', y') into S
        else
            notFound = false
    }
}
return ( x and x' ) // these are two strings that have
                    // the same hash value
```

Figure program to attack strong collision resistance.

THE BIRTHDAY ANALOGY

- Attacking strong collision resistance is analogous to answering the following:
- "What is the minimum number of persons required so that the probability of two or more in the, group having the same birthday is greater than 1/2 ?"
- It is known that in a class of only 23 random individuals, there is a greater than 50%

chance that: the birthdays of at least two persons coincide (a "Birthday Collision").

- This statement is referred, to as the Birthday Paradox.
- The following idea, first proposed by Yuval illustrates the danger in choosing hash lengths less than 128 bits.
- A malicious individual, Malloc, wishes to forge the signature of his victim, Alka, on a

  fake document, F.

- F could, for example, assert that Alka owes Malloc several million rupees.

THE BIRTHDAY ATTACK

- Malloc does the following:
  1. He creates millions of documents, Fl, F2,………Fm, etc. that are, for all practical purposes, "clones" of F.
  2. This is accomplished by leaving an extra space between two words, etc.
  3. If there are 300 words in F, there are 2300 ways in which extra spaces may be left between words.
  4. He computes the hashes, h(F1), h(F2), . . . h(Fm) of each of these documents.
  5. He creates an innocuous document, D — one that most people would not hesitate to sign. (For example, it could espouse an environmental cause relating to conservation of forests.)
  6. He creates millions of "clones" of D in the same way he cloned F above.
  7. Let D1, D2, ... be the cloned documents of D.
  8. He computes the hashes, h(D1), h(D2), . . . h(Dm) of each of the cloned documents.
  9. Malloc asks Alka to sign the document D, and Alka obliges.
  10. Later Malloc accuses Alka of signing the fraudulent document
  11. the digital signature is obtained by encrypting the hash value of the document using the private key of the signer.
  12. Thus, Alka's signature on Dj, is the same as that on Fi,.
  13. Hence, at a later point in time, Malloc can use Alka's signature on Dj), to claim that she signed the fraudulent document, F.,.

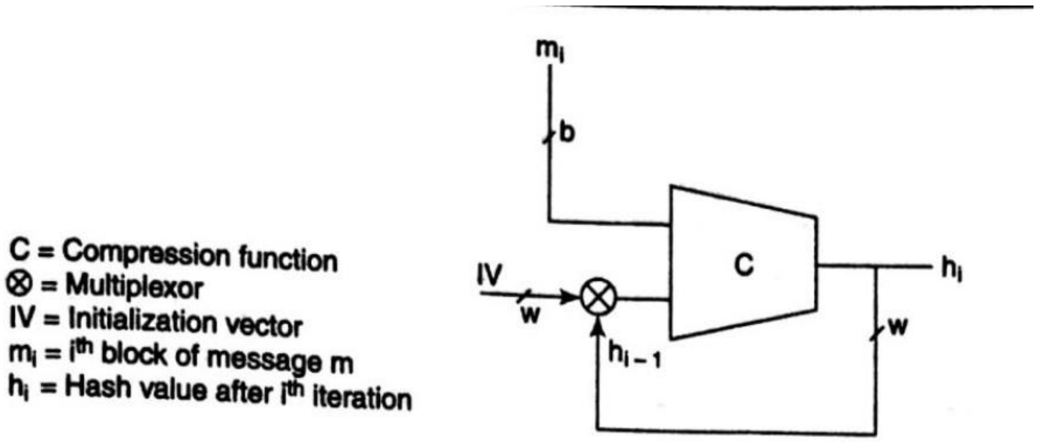
CONSTRUCTION

Generic Cryptographic Hash

- The input to a cryptographic hash function is often a message or document.
- To accommodate inputs of arbitrary length, most hash functions (including the commonly used MD-5 and SHA-1) use iterative construction as shown in Fig. 7.5.
- **C is a compression box.**
- It accepts two binary strings of lengths **b and w** and produces an output string of **length w.**
- Here, **b is the block size and w is the width of the digest**.
- During the first iteration, it accepts a pre-defined initialization vector (IV), while the top input is the first block of the message.
- In subsequent iterations, the ***"partial hash output" is fed back*** as the second input to

  the C-box.
- The top input is derived from successive blocks of the message.
- This is repeated until all the blocks of the message have been processed.
- The above operation is summarized below:
- **$h, = C(IV, m_1)$** for first block of message
- **$hi = C(h_{i-1}.m_i)$** for all subsequent blocks of the message

\



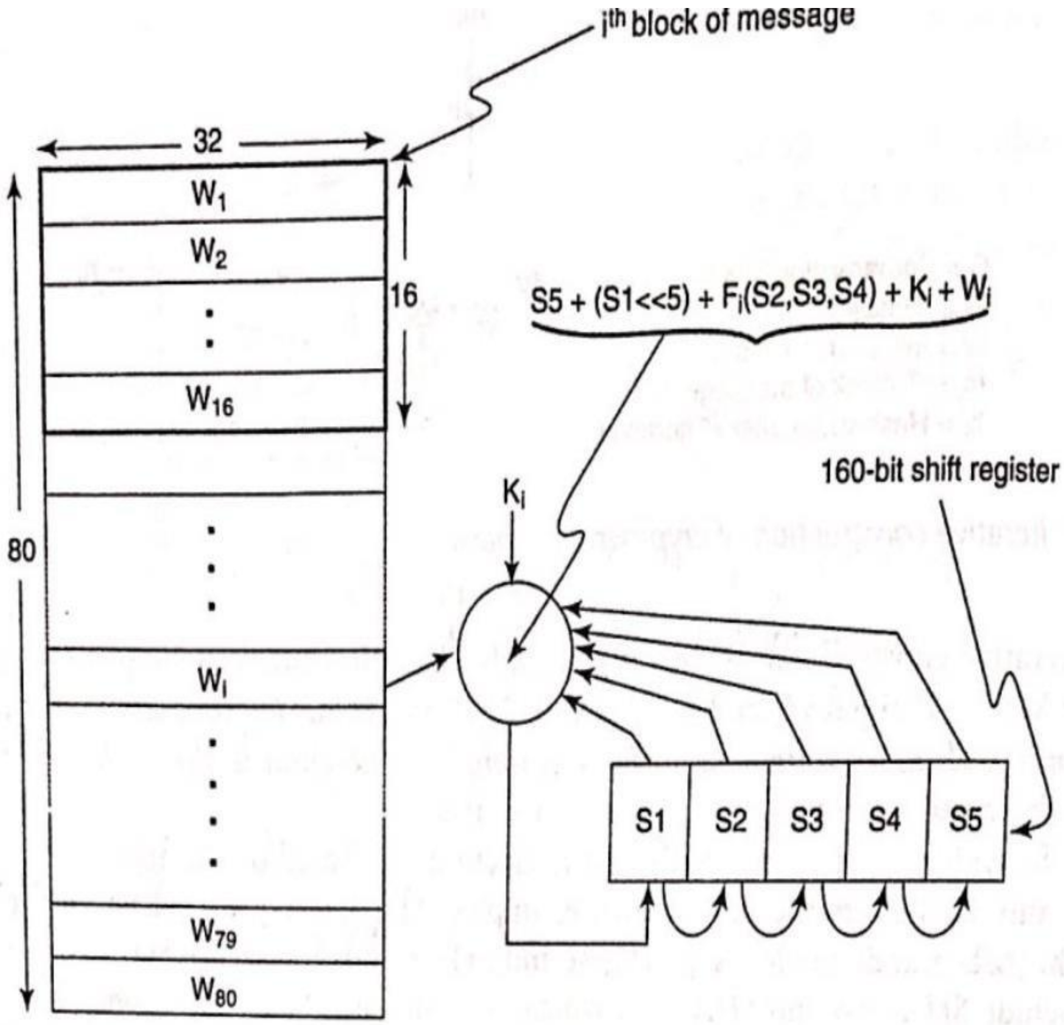
7.5 *Iterative construction of cryptographic hash*

Figure Iterative construction of cryptographic hash

- The above iterative construction of the cryptographic hash function is a simplified version of that proposed by **Merkle and Damgard.**
- It has the property that if the compression function is collision-resultant, then the resulting hash function is also collision-resultant.
- MD-5 and SHA-1 are the best known examples. MD-5 is a 128-bit hash, while SHA-1 is a 160-bit hash.

Case Study: SHA-1

- SHA-1 uses the iterative hash construction of Fig. 7.5.



igure 7.6 Computation of SHA-1

```
initialize the shift register, S1 S2 S3 S4 S5
for each block of the (message + pad + length field) {
    create the 80-word array [using Eq. (7.2)]
    for i = 1 to 80 {
        temp ← S5 + (S1 << 5) + F_i(S2, S3, S4) + K_i + W_i
        S5 ← S4
        S4 ← S3
        S3 ← S2 >> 2
        S2 ← S1
        S1 ← temp
    }
}
```

... function, F_i, is defined b

$$F_i\,(S2, S3, S4) = (S2 \wedge S3) \vee (\sim S2 \wedge S4), \qquad 1 \le i \le 20$$
$$F_i\,(S2, S3, S4) = S2 \oplus S3 \oplus S4, \qquad 21 \le i \le 40$$
$$F_i\,(S2, S3, S4) = (S2 \wedge S3) \vee (S2 \wedge S4) \vee (S3 \wedge S4), \qquad 41 \le i \le 60$$
$$F_i\,(S2, S3, S4) = S2 \oplus S3 \oplus S4 \qquad 61 \le i \le 80$$

- The message is split into blocks of ***size 512 bits***.
- The length of the message, expressed in binary as a 64 bit number, is appended to the message.
- Between the end of the message and the length field, a pad is inserted so that the length of the **(message + pad + 64)** is a ***multiple of 512,*** the block size.
- The pad has the form: 1 followed by the required number of 0's.

Array Initialization

- Each block is split into 16 words, each 32 bits wide.
- These **16 words** populate the first 16 positions, W1, W2 ……W16, of an array of **80 words.**

- The remaining **64 words** are obtained from :

$$W_i = W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16} \quad 16 < i \leq 80$$

- This array of words is shown in Figure.

Hash Computation in SHA 1

- A 160-bit shift register is used to compute the intermediate hash values (Fig. 7.6).
- It is initialized to a fixed pre-determined value at the start of the hash computation.
- We use the notation S1, S2, S3, S4, and S5 to denote the five 32 -bit words making up the shift register.
- The bits of the shift register are then mangled together with each of the words of the

  array in turn.
- The mangling is achieved using a combination of the following Boolean operations: **+, v, ~, ^, XOR ROTATE.**

APPLICATIONS AND PERFORMANCE

Hash-based MAC

- MAC is used as a message integrity check as well as to provide message

  authentication.
- It makes use of a common shared secret, k, between two communicating parties.
- The hash-based MAC that we now introduce is an alternative to the CBC -MAC.
- The cryptographic hash applied on a message creates a digest or

  digital fingerprint of that message.
- Suppose that a sender and receiver share a secret, k.
- If the message and secret are concatenated and a hash taken on this string, then the

  hash value becomes a fingerprint of the combination of the message,

  m and the secret, k.
- ***MAC = h (m|| k)***
- The MAC is much more than just a ***checksum*** on a message.
- It is computed by the sender, appended to the message, and sent

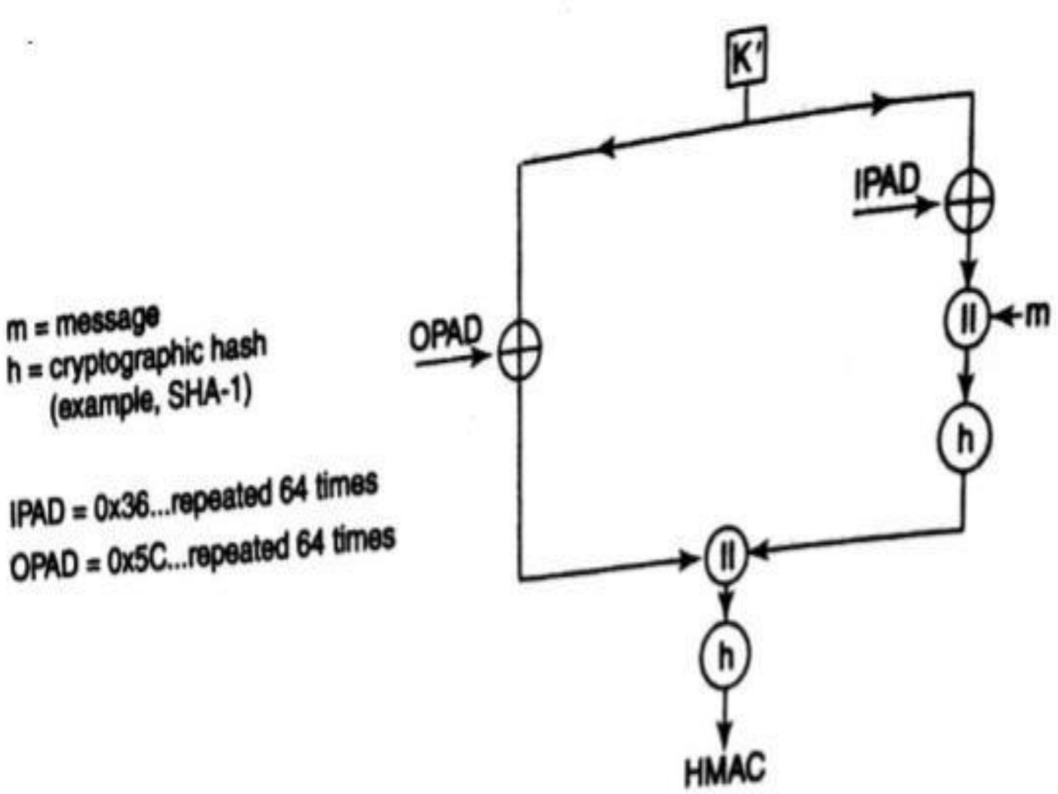
  across to the receiver.

- On receipt of the **message + MAC,** the receiver performs the computation using the common secret and the received message.
- It checks to see whether the MAC computed by it matches the received MAC.
- A change of even a single bit in the message or MAC will result in a mismatch between the computed MAC and the received MAC.
- In the event of a match, the receiver concludes the following:
- ***(a) The sender of the message is the same entity it shares the secret with — thus the MAC provides source authentication.***
- ***(b) The message has not been corrupted or tampered with in transit — thus the MAC provides verification of message integrity***.
- ***Drawbacks:***
- An attacker might obtain one or more message—MAC pairs in an attempt to determine the MAC secret.
- First, if the hash function is one-way, then it is not feasible for an attacker to deduce

  the input to the hash function that generated the MAC and thus recover the secret.
- If the hash function is collision-resistant, then it is virtually impossible for an attacker to suitably modify a message so that the modified message and the original both map to the same MAC value.

HMAC

- There are other ways of computing the hash MAC other than this method using HMAC .
- Another possibility is to use key itself as the Initialization Vector (IV) instead of concatenating it with the message.
- Bellare, Canetti, and Krawczyk proposed the HMAC and showed that their scheme is re against a number of subtle attacks on the simple hash-based MAC.
- Figure 7.7 shows how an HMAC is computed given a key and a message.



7.7 Computation of an HMAC

- The ***key is padded with O's*** (if necessary) to form a ***64-byte string*** denoted ***K'*** and

  XORed with a constant (denoted IPAD).
- It is then concatenated with the message and a hash is performed on the result.
- ***K' is also XORed*** with **another constant (denoted OPAD)** after which it is prepended to the output of the first hash.
- Once again hash is then computed to yield the HMAC.
- As shown in Fig. 7.7, HMAC performs an extra hash computation but

  provides greatly enhanced security.

Digital Signatures

The same secret that is used to generate a MAC on a message is the one that is used to verify the MAC.

Thus the MAC secret should be known by both parties - the party that generates the MAC and the party that verifies it.

A digital signature, on the other hand, uses a secret that only the signer is privy to. An example of such a secret is the signer's private key.

A crude example of an RSA signature by A on message, m, is $\mathbf{E_{A.pr}(m)}$

where A.pr is A's private key.

The use of the signer's private key is a fundamental aspect of signature generation. Hence, a message sent together with the sender's signature guarantees not just integrity and authentication but also non-repudiation, i.e., the signer of a document cannot later deny having signed it since she alone has knowledge or access to her private key used for signing.

The verifier needs to perform only a public key operation on the digital signature (using the signer's public key) and a hash on the message.

The verifier concludes that the signature is authentic if the results of these two operations tally,

$$E_{A.pu}\,(E_{A.pr}(h(m))) \stackrel{?}{=} h(m)$$

MODULE 2 - [Chapter 3]

DISCRETE LOGARITHM AND ITS APPLICATIONS.

INTRODUCTION.

- Consider the finite, multiplicative group $(Z_p^*, *_p)$ where p is prime.
- Let g be the generator of the group.
- $g^1 \bmod p, g^2 \bmod p, \ldots g^{p-1} \bmod p.$
- Let x be an element in $\{0, 1, \ldots p-1\}$.
- The function:

$$\boxed{y = g^x \pmod p}$$

→ Modular exponentiation with Base g and modulus p.

- The Inverse operation is:

$$\boxed{x = \log_g y \pmod p}$$

→ Discrete logarithm

Example.

→ Let $p = 131$
$g = 2$

* DIFFIE - HELLMAN KEY EXCHANGE.

PROTOCOL.

→ Consider two parties, A & B that need to agree upon a shared secret for the duration of their current session.

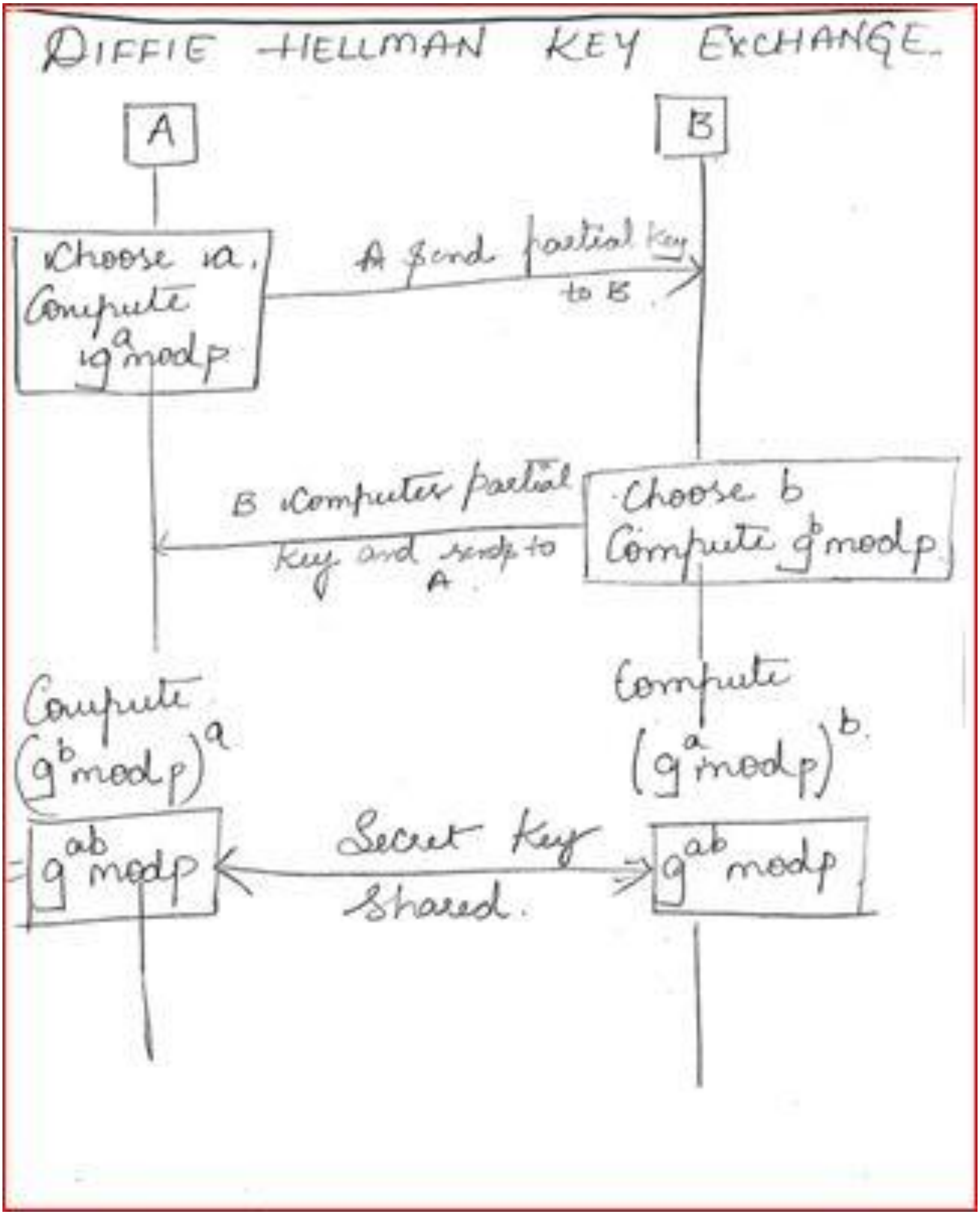
→ In 1976, Diffie and hellman proposed the idea of a private Key and corresponding public key,

1) A chooses a random integer a, $1 < a < p-1$, computes the partial key $g^a \bmod p$ and sends to B.

2) B chooses a random integer b, $1 < b < p-1$, computes the partial key $g^b \bmod p$ and sends to A.

3) On the receipt of A's msg, B computes
$(g^a \bmod p)^b \bmod p = g^{ab} \bmod p$

4) On the receipt of B's msg, A computes
$(g^b \bmod p)^b \bmod p = g^{ab} \bmod p.$



DIFFIE HELLMAN KEY EXCHANGE.
A
B
Choose a, Compute g^a mod p
A send partial key to B.
B computes partial key and sends to A.
Choose b Compute g^b mod p
Compute (g^b mod p)^a
Compute (g^a mod p)^b
g^ab mod p
Secret Key Shared.
g^ab mod p

Example:

Compute Diffie Hellman partial Keys and Secret Keys. where $a = 24$, $b = 17$, $g = 2$ and $p = 131$.

1) A computes partial Key:

$= g^a \bmod p$

$= 2^{24} \bmod 131$

$= 46$

2). B Computes partial key:

$= g^b \bmod p$

$= 2^{17} \bmod 131$

$= 72$

3) A Computes Secret key after receiving B's partial key.

$= (g^b \bmod p)^a$ → B's partial key.

$= (72)^{24} \bmod 131$

$= \boxed{13}$

4) B Computes Secret key: $(g^a \bmod p)^b$

$= 46^{17} \bmod 131$

$= \boxed{13}$

ATTACKS

- The partial Keys, $g^a \bmod p$ and $g^b \bmod p$ are sent in clear
- An Eavesdropper with the Knowledge of the partial Keys and public parameters (p and g) deduce the Common Secret $g^{ab} \bmod p$, derived by A & B.
- This problem is referred to as Computational Diffie Hellman problem.

MAN IN THE MIDDLE ATTACK ON DIFFIE - HELLMAN KEY EXCHANGE.

- An attacker, C chooses an integer c and computes $g^c \bmod p$.
- C then interrupts A's message to B, substitutes it with $g^c \bmod p$ and sends this instead to B.
- C also intercepts B's message to A sending $g^c \bmod p$ instead.
- After the message transfer B computes $\to (g^c \bmod p)^b \bmod p$

$$\to \boxed{g^{bc} \bmod p.}$$

- While A computes,

$$(g^{c} \bmod p)^{a} \bmod p = \boxed{g^{ac} \bmod p.}$$

- C also computes the two secrets
  → $g^{ac} \bmod p$ and
  → $g^{bc} \bmod p$.
- A and B might think that they have a secure channel for communication by encrypting all messages.
- But A shares the secret $g^{ac} \bmod p$ with C,
- B shares the secret $g^{bc} \bmod p$ with C.
- Every subsequent message encrypted by A and intended for B can be decrypted by C.
- Similarly every message from B to A can be decrypted by C.
- This is a classic example of an active "Man in the Middle Attack".

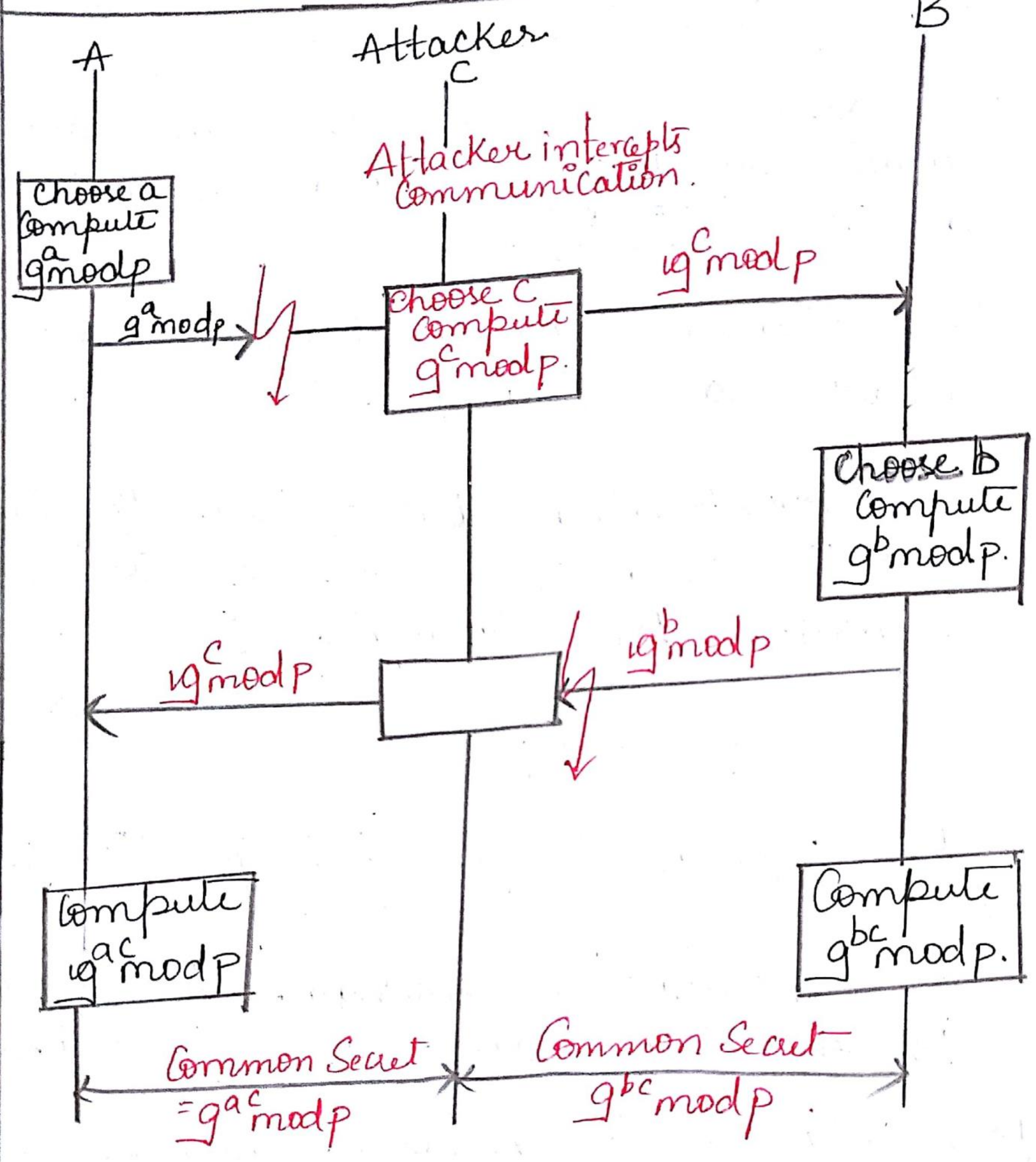


Fig: Man in the Middle Attack on Diffie Hellman Key Exchange.

EL GAMAL ENCRYPTION.

- El gamal encryption uses a large prime number p and generator g in $(Z_p^*, *_p)$.
- An Elgamal private Key, is an ~~integ~~ integer a, $1 < a < p-1$.
- The Corresponding public Key is the triplet (p, g, α) where α is the encryption Key calculated :

$$\alpha = g^a \bmod p.$$

- Let (p, g, α) be the public Key of A.
- To Encrypt a message to be sent to A, B does the following:

1) B chooses a random number r, $1 < r < p-1$ such that r is relatively prime to $p-1$

2) B Computes :

$$\boxed{C_1 = g^r \bmod p} \quad \boxed{C_2 = (m * \alpha^r) \bmod p}$$

3) B sends the Ciphertext
$C = [C_1, C_2]$ to A.

Decryption At A's side
* A uses its private key a to decrypt and obtain plaintext m.

$$\boxed{(C_1^{-a}) * C_2 \bmod p}$$

*ELGAMAL SIGNATURES.
→ Let a be the private key of A.
→ Let (p, g, α) be the public key of A.
→ To sign a message m, A does the following:
1) She computes the hash $h(m)$ of the message.
2) She chooses a random number r, $1 < r < p-1$, such that r is relatively prime to $p-1$.

3) She Computes

$$\boxed{x = g^r \bmod p}$$

4) She Computes

$$\boxed{y = (h(m) - ax)r^{-1} \bmod (p-1)}$$

5) The Signature is the pair (x, y).

* Signature verification user X,

To prove Elgamal Signature:

Consider step 4 Eqn

$$y = (h(m) - ax)r^{-1} \bmod p-1$$

$$y = (h(m) - ax)\frac{1}{r} \bmod p-1$$

$$ry = (h(m) - ax) + \underbrace{K(p-1)}$$ where K is an integer

→ Raising Both Sides to power of g and reducing Modulo P.

Is Equal to 1 [Fermat's theorem]

$$g^{ry} = g^{h(m)} g^{-ax} \bmod p.$$

$$g^{ry} = g^{h(m)} \frac{1}{g^{ax}} \bmod p.$$

$$g^{ax} \cdot g^{ry} = g^{h(m)} \bmod p.$$

So. $\alpha^{x} * x^{y} = g^{h(m)} \bmod p.$ [Since $\alpha = g^{a} \bmod p$. $x = g^{r} \bmod p$]

* SCHNORR SIGNATURE

→ Schnorr signature is the pair (x, y) where

$$x = h(m \,||\, g^{r} \bmod p) \text{ and}$$

$$y = (r + ax) \bmod q.$$

where $\alpha = g^{a} \bmod p$.

r be random number

$$1 \leq r \leq q-1.$$

PROBLEMS ON ELGAMAL ENCRYPTION.

Q. A Block of plaintext message $m=3$, has to be encrypted,
Assume $P=11$, $g=2$, recipients private Key $=5$,
Sender chooses random integer $r=7$.
Perform Encryption & Decryption.

Step 1: $p=11$, $g=2$
Recipients private key, $a=5$.
Compute public key of receiver:

$$\alpha = g^{a} \bmod p.$$

$$\alpha = 2^{5} \bmod 11$$

$$\alpha = 32 \bmod 11$$

$$\boxed{\alpha = 10}$$

Step 2: Compute C_1 and C_2 [Sender has to compute]

$$C_1 = g^{r} \bmod p$$

$C_1 = g^r \bmod p$ $\quad [r=7]$

$= 2^7 \bmod 11$

$= 128 \bmod 11$

$\boxed{C_1 = 7}$

$C_2 = m * \alpha^r \bmod p$ $\quad [m=3]$

$= 3 * 10^7 \bmod 11$

$\boxed{C_2 = 8}$

$C = [7, 8]$

$7 \times 3 = 21 \bmod 11 \times$

$7 \times 5 = 35 \bmod 11 \times$

not Equal to 1 hence Continue.

Step 3: Decrypt

$m = C_1^{-a} * C_2 \bmod p$

$= 7^{-5} * 8 \bmod 11$

$= (7^{-1})^5 * 8 \bmod 11$

Substitute $7^{-1} = 8$

Inverse

$\therefore 7 \times 8 = 56$

Take $56 \bmod 11$

$= 1$

[Equivalent to 1]

$= 8^5 * 8 \bmod 11$

$\boxed{m = 3}$

Q. $p = 23,\ g = 11,\ a = 6,\ r = 3,\ m = 10.$

Step1: $\alpha = g^a \bmod p$

$= 11^6 \bmod p$

$\boxed{\alpha = 9}$

Step2: Compute C_1, C_2

$C_1 = g^r \bmod p$

$= 11^3 \bmod 23$

$\boxed{C_1 = 20}$

$C_2 = (m * \alpha^r \bmod p)$

$= (10 * 9^3 \bmod 23)$

$\boxed{C_2 = 22}$

Step3: Decrypt:

$m = C_1^{-a} * C_2 \bmod p.$

$= 20^{-6} * 22 \bmod 23$

$= (20^{-1})^6 * 22 \bmod 23.$

$= (15)^6 * 22 \bmod 23$

$\boxed{m = 10}$

[Not Equal to 1]

$20*1 = 20 \bmod 23$ ✗

$20*2 = 40 \bmod 23$ ✗

⋮

$20*15 = 300 \bmod 23$

$= 1$ ✓