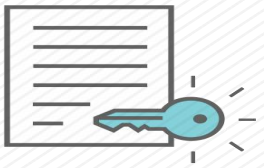
National Institute of Technology, Warangal
(*Department of Computer Science Engineering*)

Public Key Cryptography

Lecture By:-
Dr R Padmavathy
Dept of CSE, NIT Warangal

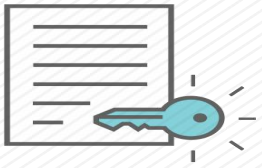
One-Way Functions



One-Way Functions

A **one-way function** is a function that is “easy” to compute and “difficult” to reverse.

How might we express this notion of a one way function informally in complexity theoretic terms?



OWF: Multiplying two primes (Integer Factorization)

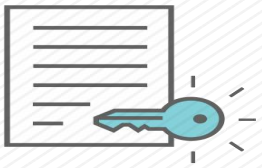
Integer Factorization Problem(IFP) is given two numbers u and v, finding their product n is easy but given n finding u and v is hard.

- Multiplication runs in polynomial time.
- Multiplication of two prime numbers is **believed** to be a one-wa function.

OWF: Modular exponentiation (*DLP*)

Let **G** be a finite group of size **n**, assume that **G** is a multiplicative cyclic group, and **g** is generator of **G**. Any element **a** in **G** can be uniquely expressed as $a = g^x$ for some integer **x** in the range $0 <= x <= n-1$. The integer **x** is called as the discrete logarithm or index of **a** with respect to **g**.

Computing **x** from **G, g** and **a** is called the discrete logarithm problem.



OWF: Modular exponentiation (*DLP*)

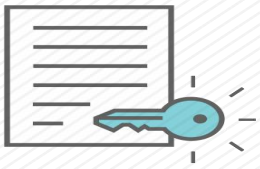
- **Modular exponentiation** means computing a^b modulo some other number **n**. We tend to write this as

  $a^b \bmod n$.

- Modular exponentiation is “easy”.
- In other words, given a number **a** and a prime number **n**, the function

  $f(b) = a^b \bmod n$

  is believed to be a one-way function.



OWF: Modular exponentiation (*DLP*)

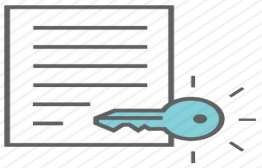
- However, given **a**, **n**, and $a^b \bmod n$ (when **n** is prime), calculating **b** is regarded by mathematicians as a hard problem.
- This difficult problem is often referred to as the **Discrete Logarithm Problem (DLP)**.
- In other words, given a number **a** and a prime number **n**, the function

$$f(b) = a^b \bmod n$$

- is believed to be a one-way function.

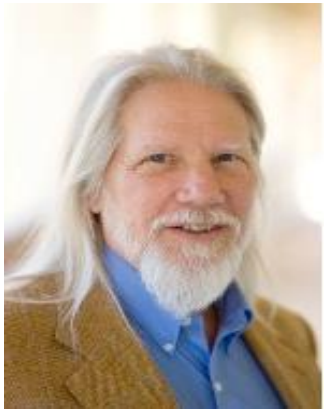
Introduction to

Public Key Cryptography



Public Key Cryptography

Martin Hellman



Whitfield Diffie

Whitfield Diffie & Martin Hellman introduced the idea of Public Key Cryptography at Stanford U.S. in 1976 .

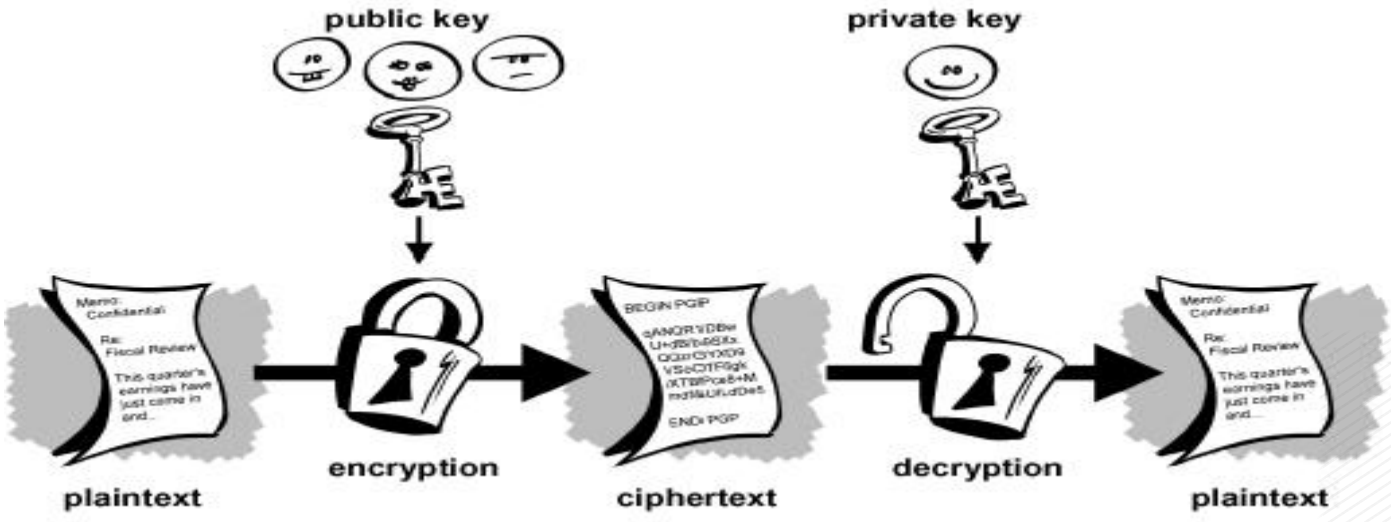


Fig-2: Public Key Cryptography Concept

Public Key Cryptography

Public key blueprint

- The keys used to encrypt and decrypt are different.
- Anyone who wants to be a receiver needs to "publish" an encryption key, which is known as the ***public key***.
- Anyone who wants to be a receiver needs a unique decryption key, which is known as the ***private key***.

Thus PKC has six ingredient

1. Plain text
2. Encryption algorithm
3. Public and private keys
4. Ciphertext
5. Decryption algorithm

Public Key Cryptography

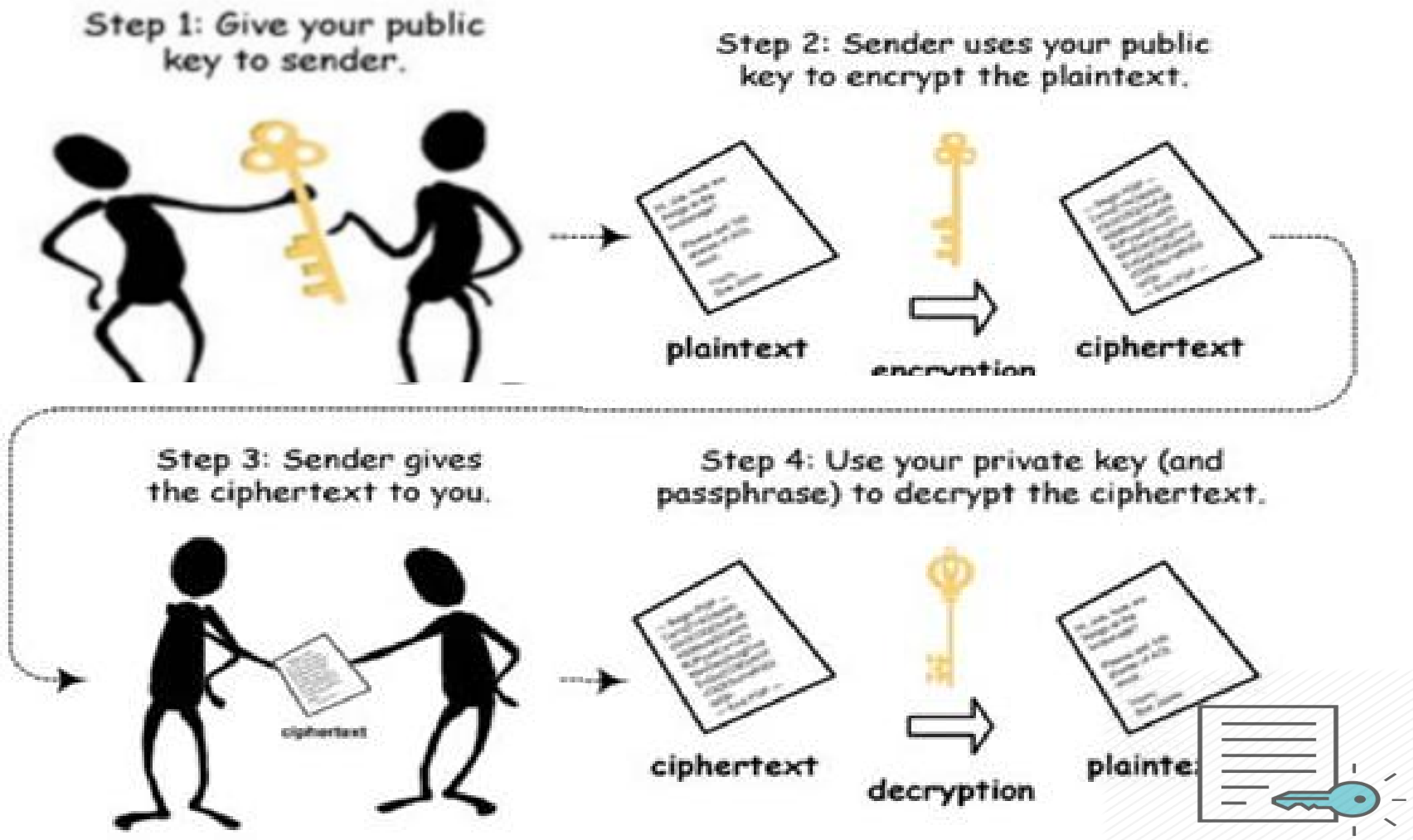
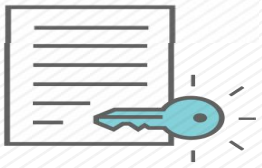


Fig-3: PKC procedure for Encryption and Decryption

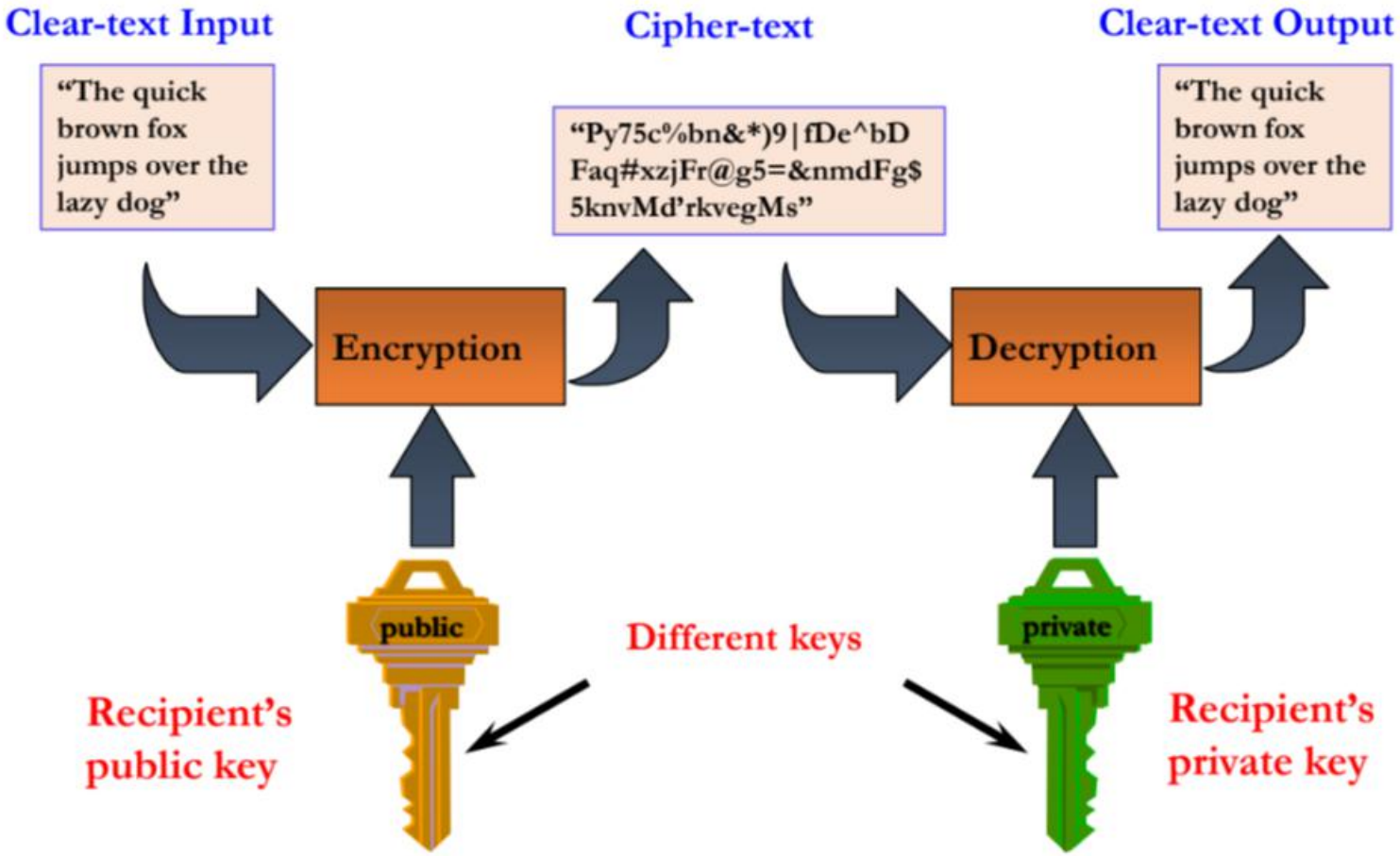
Public Key Characteristics

Public-Key algorithms rely on two keys where:

- it is computationally infeasible to find decryption key knowing only algorithm & encryption key
- it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
- either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)



Public Key Cryptosystem



Public Key Cryptography Requirements

1. It is easy for party **B** to generate a pair of keys (public key **PUb** , Private key **PRb).**
2. It is easy for a sender **A** , knowing the public key and message to be encrypt**. C=E(PUb, M)**
3. It is easy for receiver **B** to decrypt the resulting ciphertext using the private key . **M=D(PRb,C)=D[PRb,E(PUb,M)]**
4. It is infeasible for an any person , to know the public key **PUb** to determine the private key **PRb.**
5. It is infeasible for any person to know the public key **PUb** and a ciphertext **C** to recover the original message **M.**
6. Two keys can be applied in either order

M=DP[PUb, E(PRb,M)] = D[PRb,E(PUb, M)]

Public Key Cryptography Applications

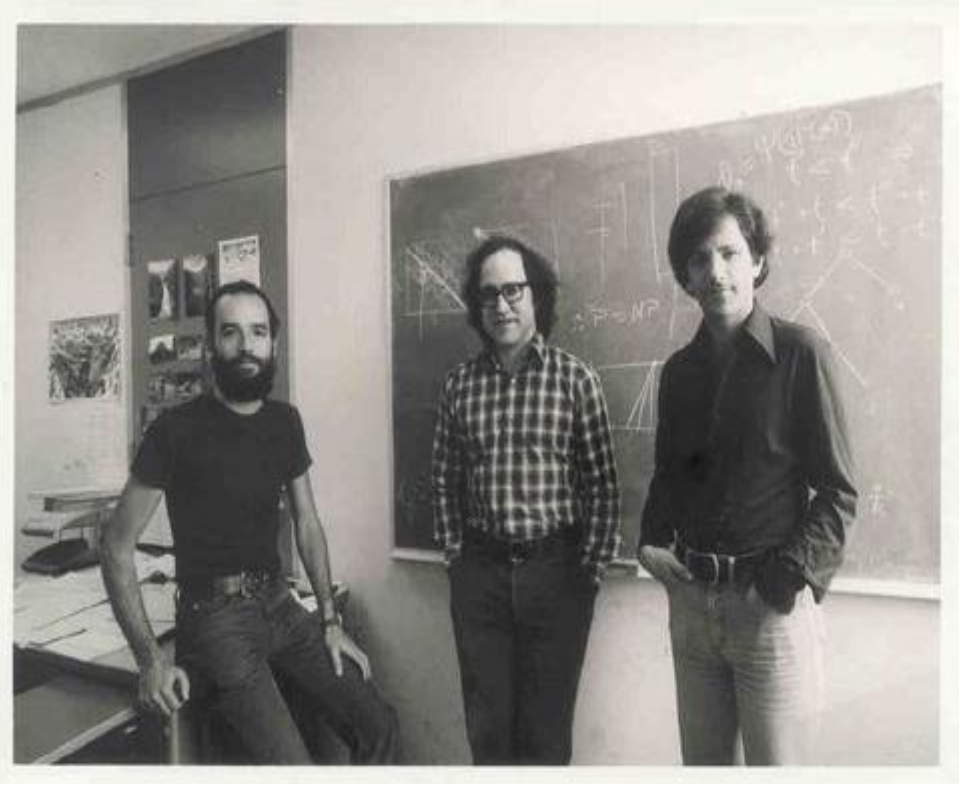
can classify uses into 3 categories:

1. *encryption/decryption* (provide secrecy)
2. *digital signatures* (provide authentication)
3. *key exchange* (of session keys)

some algorithms are suitable for all uses, others are specific to one

RSA Cryptosystem

RSA Algorithm



Ron Rivest , Adi Shamir and Len Adleman (left to right)

- Invented by **R**ivest, **S**hamir & **A**dleman of MIT in 1977.
- It is a best known & widely used public-key scheme.
- It is a *block cipher algorithm* in which palintext and ciphertext integers between 0 to n-1 for some ***n*.**
- A typical size for ***n*** is 1024 bits or 309 decimal digits.

- Based on idea that . . . *Factorization is difficult*.

RSA Algorithm (Mathmetics Involved)

- Let **n** be the product of two large primes **p** and **q**
  - By “large” we typically mean at least 512 bits.
- $\Phi(n) = (p-1) \times (q-1)$
- Select a special number **e** between 1 and $\Phi(n)$.
- Publish the pair of numbers (**n** , **e**)
- Compute the private key **d** from **p**, **q** and **e** .

RSA Algorithm (Computing Private key)

- The private key **d** is computed to be the unique inverse of **e** modulo (**p**-1)(**q**-1).
- In other words, **d** is the unique number less than (**p**-1)(**q**-1) that when multiplied by **e** gives you 1 modulo (**p**-1)(**q**-1).
- Written mathematically:

$$ed = 1 \bmod (p-1)(q-1)$$

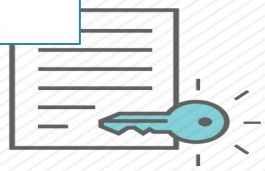
- The **Euclidean Algorithm** is the process that you need to follow in order to compute **d**.

RSA Algorithm

Key Generation	
Select p, q	p, q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1) \times (q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

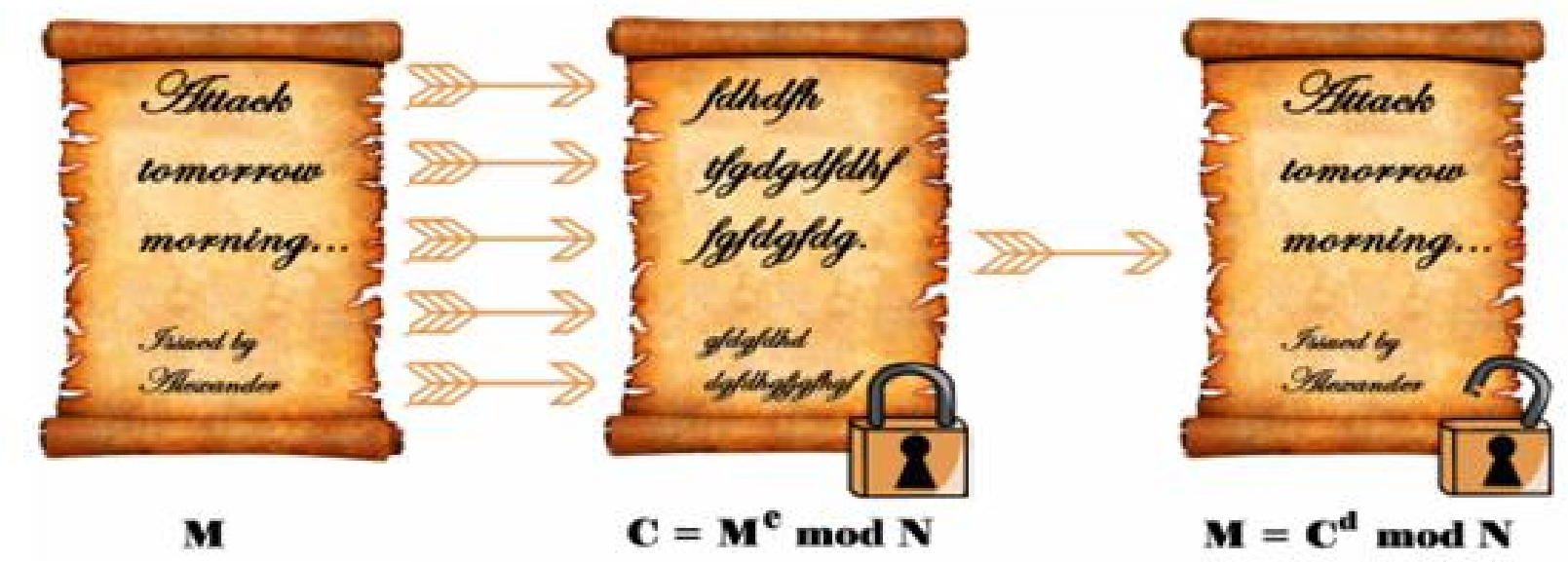
Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod n$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod n$



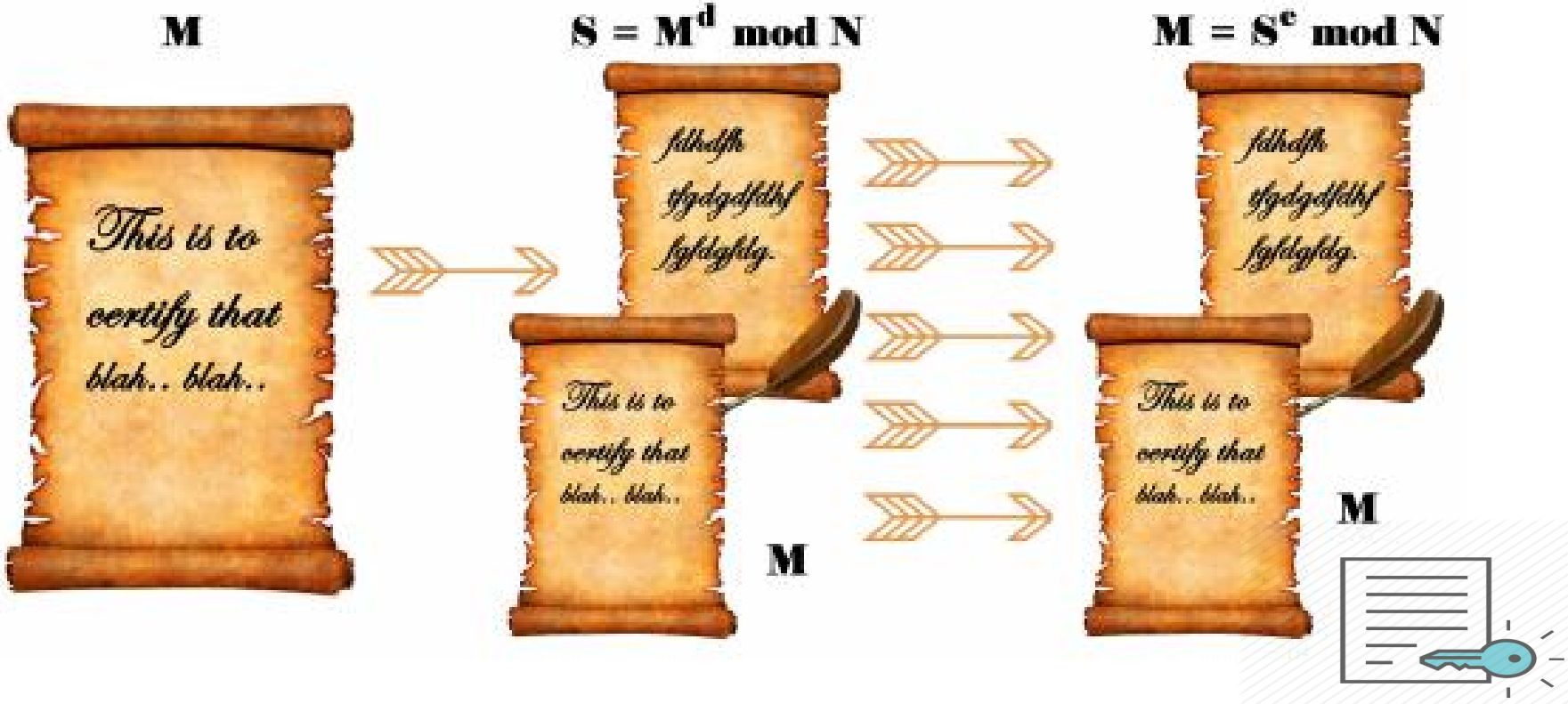
RSA Algorithm (Encryption)

Simplified model of RSA Cryptosystem



RSA Algorithm (Signature)

Simplified model of RSA Signature



RSA Algorithm Example

Step 1: Let p = **47** and q = **59**. Thus n = **47** x **59** = **2773**

Step 2: Select e = **17**

Step 3: Publish (n,e) = (**2773**, **17**)

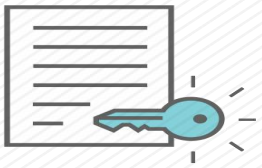
Step 4: $\Phi(n)$ = (p-1) x (q-1) = **46** x **58** = **2668**

Use the Euclidean Algorithm to compute the modular inverse of **17** modulo **2668**. The result is d = **157**

<< Check: **17** x **157** = 2669 = 1(mod **2668**) >>

Public key is (**2773**,**17**)

Private key is **157**



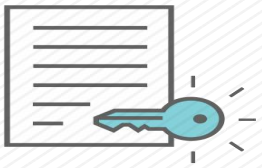
RSA Algorithm Example Cont.

- Public key is (**2773**,**17**)
- Private key is **157**

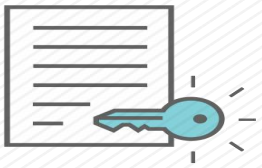
- Plaintext block represented as a number: M = 31
- Encryption using Public Key: $C = 31^{17} \pmod{2773}$

  $= 587$
- Decryption using Private Key: $M = 587^{157} \pmod{2773}$

  $= 31$



Attacks On RSA



RSA Security

We will look at two different strategies for trying to "break" RSA:

1. Trying to decrypt a ciphertext without knowledge of the private key
2. Trying to determine the private key

RSA Security

Decrypting ciphertext without the key

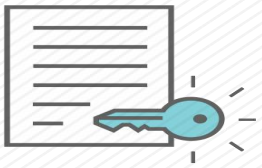
- The encryption process in RSA involves computing the function $C = M^e \bmod n$, which is regarded as being easy.
- An attacker who observes this ciphertext, and has knowledge of e and n, needs to try to work out what M is.
- Computing M from C, e and n is regarded as a hard problem.

Atack-1: Mathematical Attacks

Factoring Attacks :

There are many possible factoring attacks which are :

1. Fermat's Factorization Method
2. Euler's Factorizarion Method
3. Pollard's p-1 Method
4. Pollard's p Method
5. Miller Rabin Test



Attack-2 : Common Modulus Attack

- Suppose Alice dispatches the same message M to two different recievers whose encryption exponents respectively are e_1 and e_2.
- The encrypted messages are:

$$E = M^{e_1} \bmod n \quad \text{and} \quad F = M^{e_2} \bmod n$$

- An eavesdropper has access to the public keys n, e_1 and e_2, and the encrypted messages E and F.
- Since $gcd\,(e_1, e_2)=1$, the eavesdropper applies the extended Euclidean algorithm to compute integers x and y such that

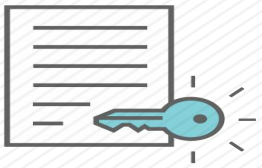
$$x \cdot e_1 + y \cdot e_2 = 1.$$

- Then with x and y in hand, the eavesdropper computes

$$E^x \cdot F^y \bmod n = M \text{ (original message)}$$

Attack-3 : Timing Attack

- developed in mid-1990's
- exploit timing variations in operations
  - ✓ eg. multiplying by small v/s large number
  - ✓ or IF's varying which instructions executed
- infer operand size based on time taken
- RSA exploits time taken in exponentiation
- countermeasures
  - ✓ use constant exponentiation time
  - ✓ add random delays
  - ✓ blind values used in calculations

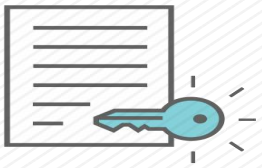


Attack-4 : Blinding Attack

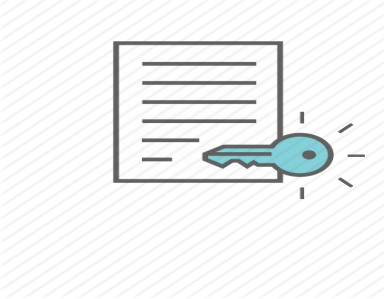
- Attack specific to RSA signatures .
- Suppose attacker A wants to get a document M signed by B .
- A needs $M^d \bmod N \cdots$ A sends $r^e M \bmod N$ for B to sign .
- Signing the sent message gives

$$r^{ed} M^d \bmod N = rM^{\mathrm{d}} \bmod N .$$

- Security measure: Random Padding, Signing Hash .



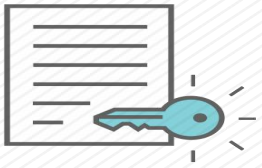
Diffie-Hellman Key Exchange



Diffie-Hellman Key Exchange

It is used by two users to *securely exchange a key* that can be used for subsequent encryption of messages.

- a public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants
- value of key depends on the participants (and their private and public key information)
- based on mathematical principles
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard



Diffie-Hellman Key Exchange Algorithm

Global Public Elements

q = prime number(300 decimal, i.e. 1024 bits)

α = generator of field F_q

User A key Generation

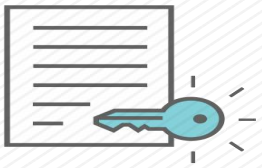
Select private X_a , $X_a < q$

Calculate public Y_a , $Y_a = \alpha^{X_a} \bmod q$

User B Key Generation

Select private X_b , $X_b < q$

Calculate public Y_b , $Y_b = \alpha^{X_b} \bmod q$



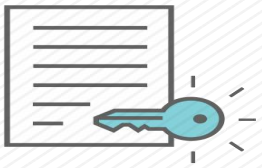
Diffie-Hellman Key Exchange Algorithm

Generation of secret key by user A

$$K=(Y_b)^{X_a} \bmod q$$

Generation of secret key by user B

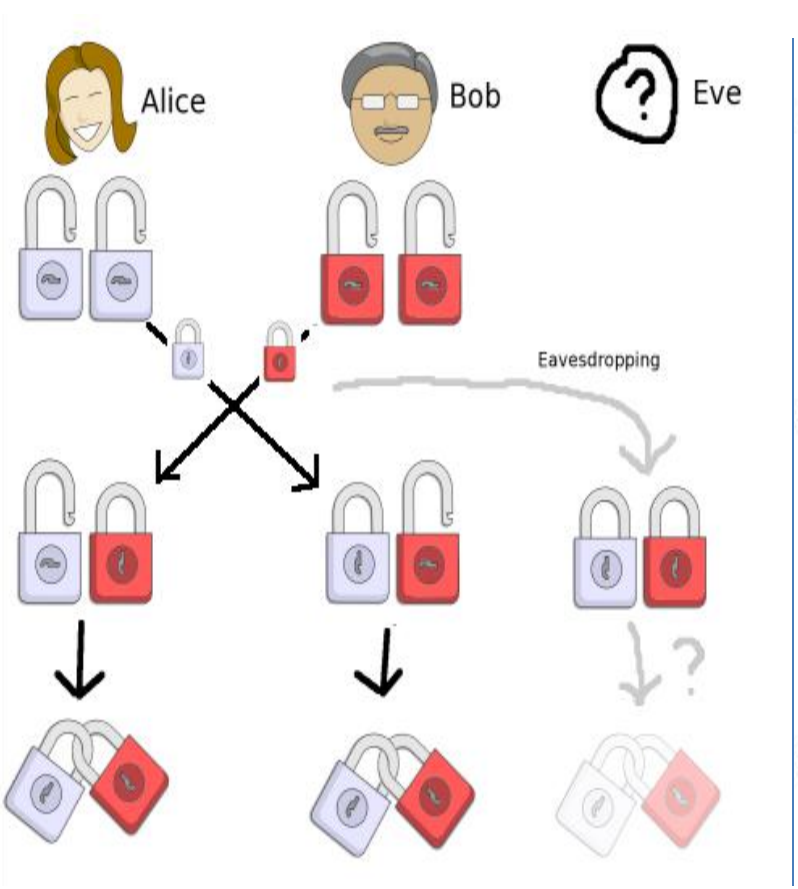
$$K=(Y_a)^{X_b} \bmod q$$



Key Exchange Example

- users Alice & Bob who wish to swap keys:
- agree on prime q=353 and α=3
- select random secret keys:
  - A chooses x_A=97, B chooses x_B=233
- compute respective public keys:
  - $y_A = 397 \mod 353 = 40$ (Alice)
  - $y_B = 3233 \mod 353 = 248$ (Bob)
- compute shared session key as:
  - $K_{AB} = y_B^{x_A} \mod 353 = 24897 = 160$ (Alice)
  - $K_{AB} = y_A^{x_B} \mod 353 = 40233 = 160$ (Bob)

Diffie-Hellman Key Exchange



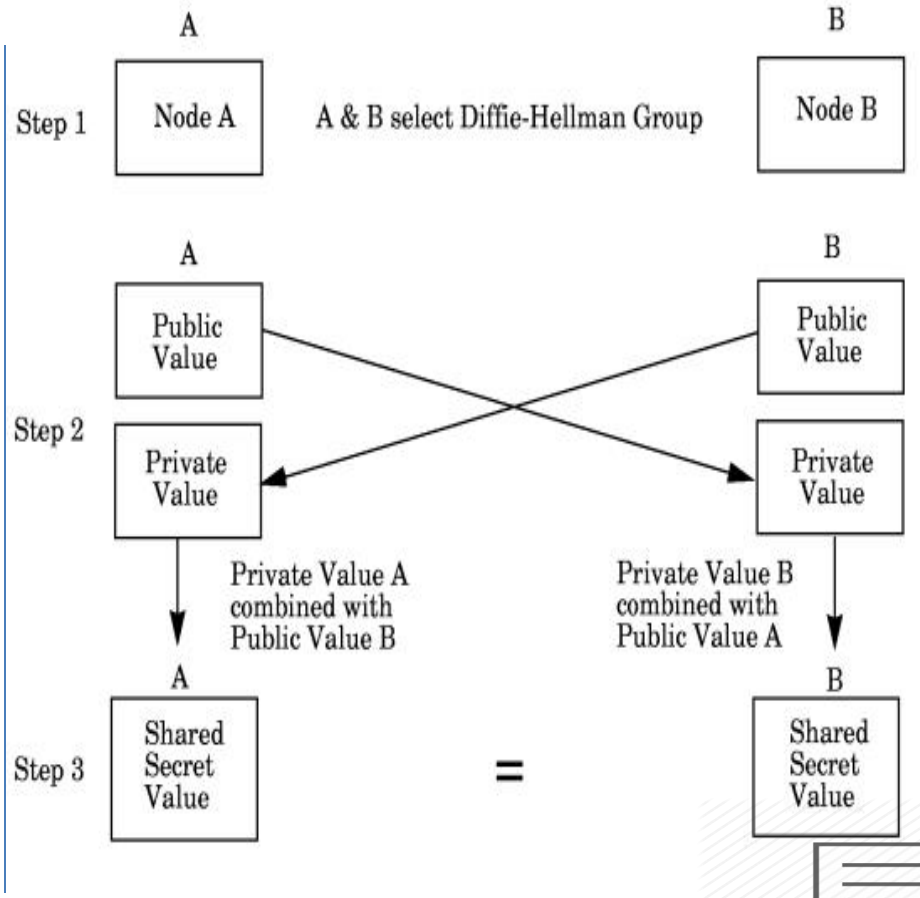
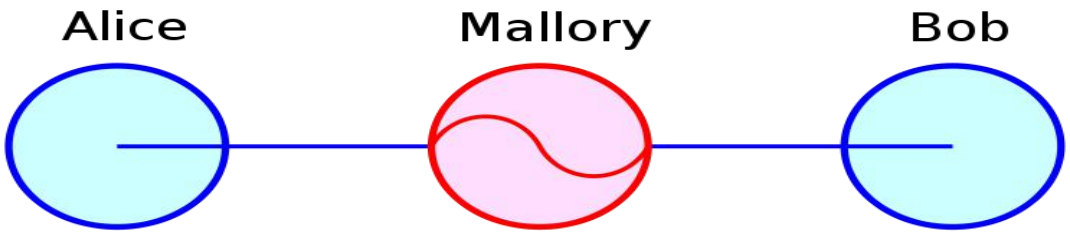


Fig-1 : Diffie-Hellman Key-Exchange Procedure

Man-in-the-Middle Attack

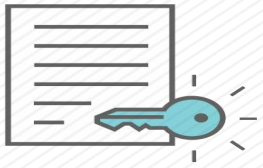
A man-in-the-middle (MITM) attack is a form of eavesdropping where communication between two users is monitored and modified by an unauthorized party. Generally, the attacker actively eavesdrops by intercepting a public key message exchange and retransmits the message while replacing the requested key with his own.



Man-in-the-Middle Attack

The man-in-the-middle attack, against the Diffie-Hellman key Exchange .

1. Mallory chooses an exponent z.
2. Mallory intercepts q^x and q^y.
3. Mallory sends q^z to both Alice and Bob. (After that Alice believes she has received q^y and Bob believes he has received q^x.)
4. Mallory computes $K_A = q^{xz} \pmod{p}$ and $K_B = q^{yz} \pmod{p}$.
   Alice, not realizing that Mallory is in the middle, also computes K_A and Bob, not realizing that Mallory is in the middle, also computes K_B.
5. When Alice sends a message to Bob, encrypted with K_A, Mallory intercepts it, decrypts it, then encrypts it with K_B and sends it to Bob.
6. Bob decrypts the message with K_B and obtains the message. At this point he has no reason to think that communication was insecure.
7. Meanwhile, Mallory enjoys reading Alice's message.



Man-in-the-Middle Attack

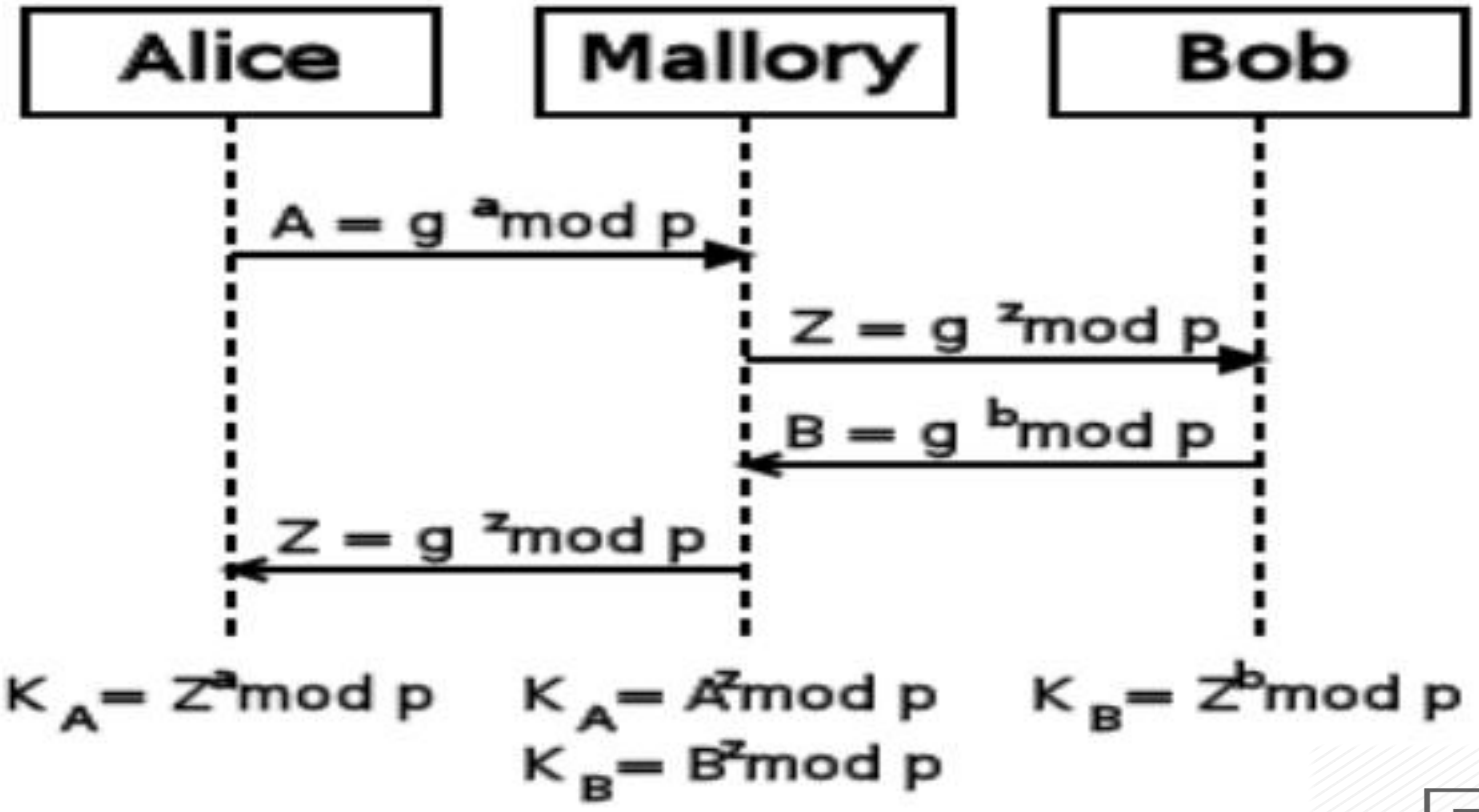


Fig-2 : Man-in-the-Middle Attack

Man-in-the-Middle Attack

Man in the middle is known most to others as "session hijacking" and to general public as "hijacking". These hackers are primarily targeting specific data about the transactions on computers.
MITM leads many other attacks such as:

- ARP spoofing , DNS spoofing ,
- DHCP spoofing ,IP address spoofin
- Port Stealing

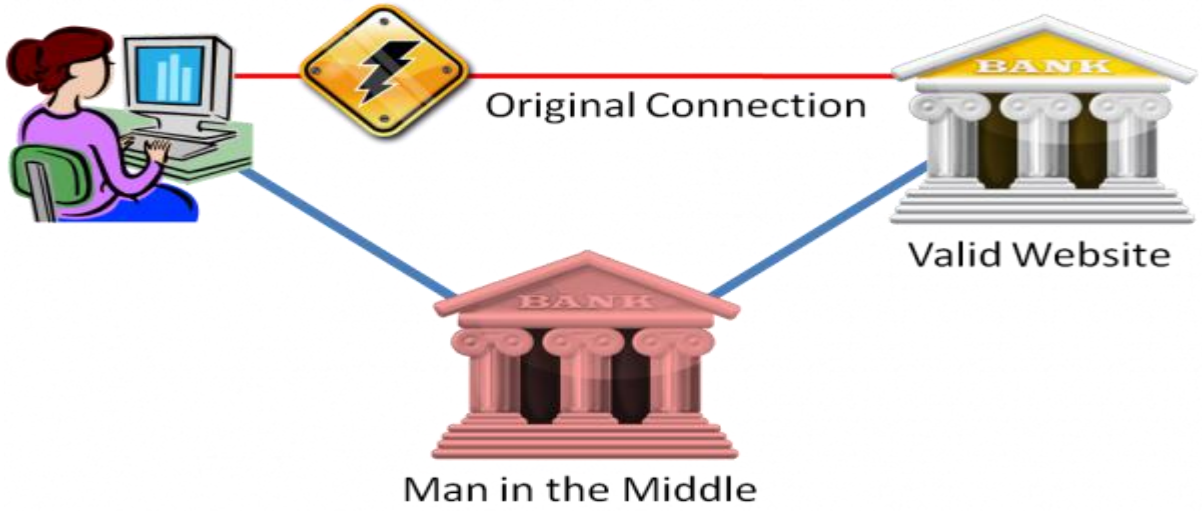
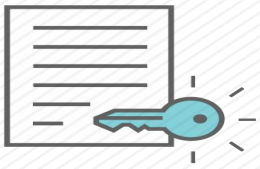
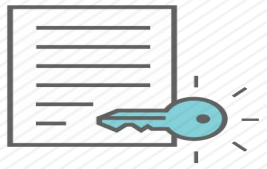


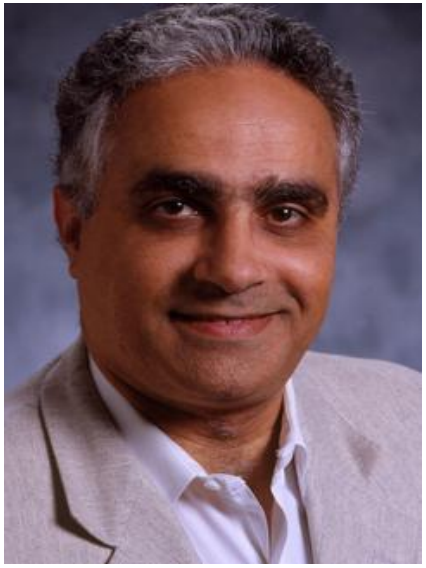
Fig-3 : MITM Attack Real World Scenario



ElGamal Cryptosystem



ElGamal Cryptosystem



Taher ElGamal

- El_Gamal is a public-key cryptosystem technique was designed by Dr. Taher Elgamal .
- El_Gamal depends on the one way function, means that the encryption and decryption are done in separate functions.
- **Key aspects:**
  - Based on Discrete Logarithm Problem,
  - Randomized Encryption
- **Applications :**
  - Establishing a secure channel fir key sharing
  - Encrypting messages.

ElGamal Algorithm

Let p be a large prime

By “large” we mean here a prime rather typical in length to that of an RSA modulus .

Select a special number **g**

The number g must be a **primitive element** modulo **p**.

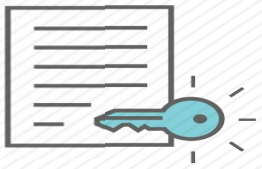
Choose a private key **x**

This can be any number bigger than 1 and smaller than **p**-1

Compute public key **y** from **x**, **p** and **g**

The public key **y** is **g** raised to the power of the private key **x** modulo **p**. In other words:

$$y = g^x \bmod p$$



ElGamal Algorithm : Example

Step 1: Let p = **23**

Step 2: Select a primitive element g = **11**

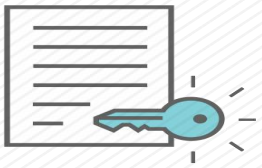
Step 3: Choose a private key x = **6**

Step 4: Compute $y = 11^6 \pmod{23}$

$= 9$

Public key is **9**

Private key is **6**



ElGamal Encryption

The first job is to represent the plaintext as a series of numbers modulo p. Then:

1. Generate a random number k
2. Compute two values C_1 and C_2, where

$$C_1 = g^k \bmod p \text{ and } C_2 = M \cdot y^k \bmod p$$

3. Send the ciphertext C, which consists of the two separate values C_1 and C_2.

ElGamal Encryption: Example

To encrypt M = 10 using Public key **9**

1 - Generate a random number k = **3**

2 - Compute $C_1 = 11^3 \bmod 23 = 20$

$C_2 = 10 \times 9^3 \bmod 23$

$= 10 \times 16 = 160 \bmod 23 = 22$

3 - Ciphertext C = (**20** , **22**)

ElGamal Encryption

$$C_1 = g^k \bmod p \qquad C_2 = M \cdot y^k \bmod p$$

1 - The receiver begins by using their private key **x** to transform

C_1 into something more useful:

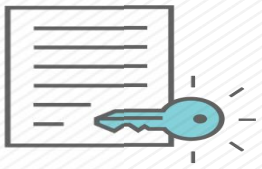
$$C_1^x = (g^k)^x \bmod p$$

NOTE: $C_1^x = (g^k)^x = (g^x)^k = (y)^k = y^k \bmod p$

2 - This is a very useful quantity because if you divide C_2 by it you

get **M**. In other words:

$$C_2 / y^k = (M \cdot y^k) / y^k = M \bmod p$$



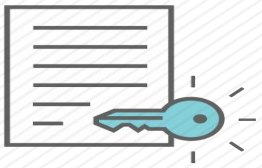
ElGamal Encryption : Example

To decrypt C = (**20** , **22**)

1 - Compute $20^{6} = 16 \bmod 23$

2 - Compute $22 / 16 = 10 \bmod 23$

3 - Plaintext = 10



Security of ElGamal

Recall the two different strategies for trying to “break” RSA:

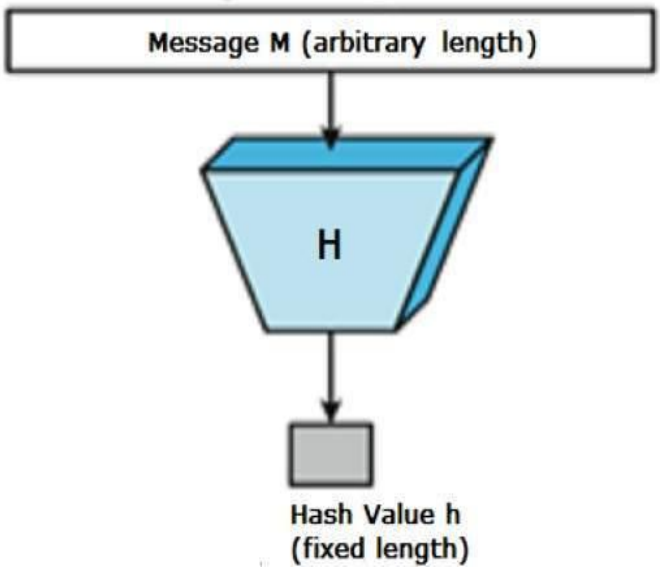
1. Trying to decrypt a ciphertext without knowledge of the private key .
2. Trying to determine the private key

What hard problems do you come across if you try to follow these two different strategies to break ElGamal?

Hash Functions

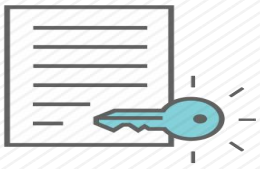
Cryptographic Hash Function

Cryptographic Hash functions is mainly used for Data Integrity And Authentication.

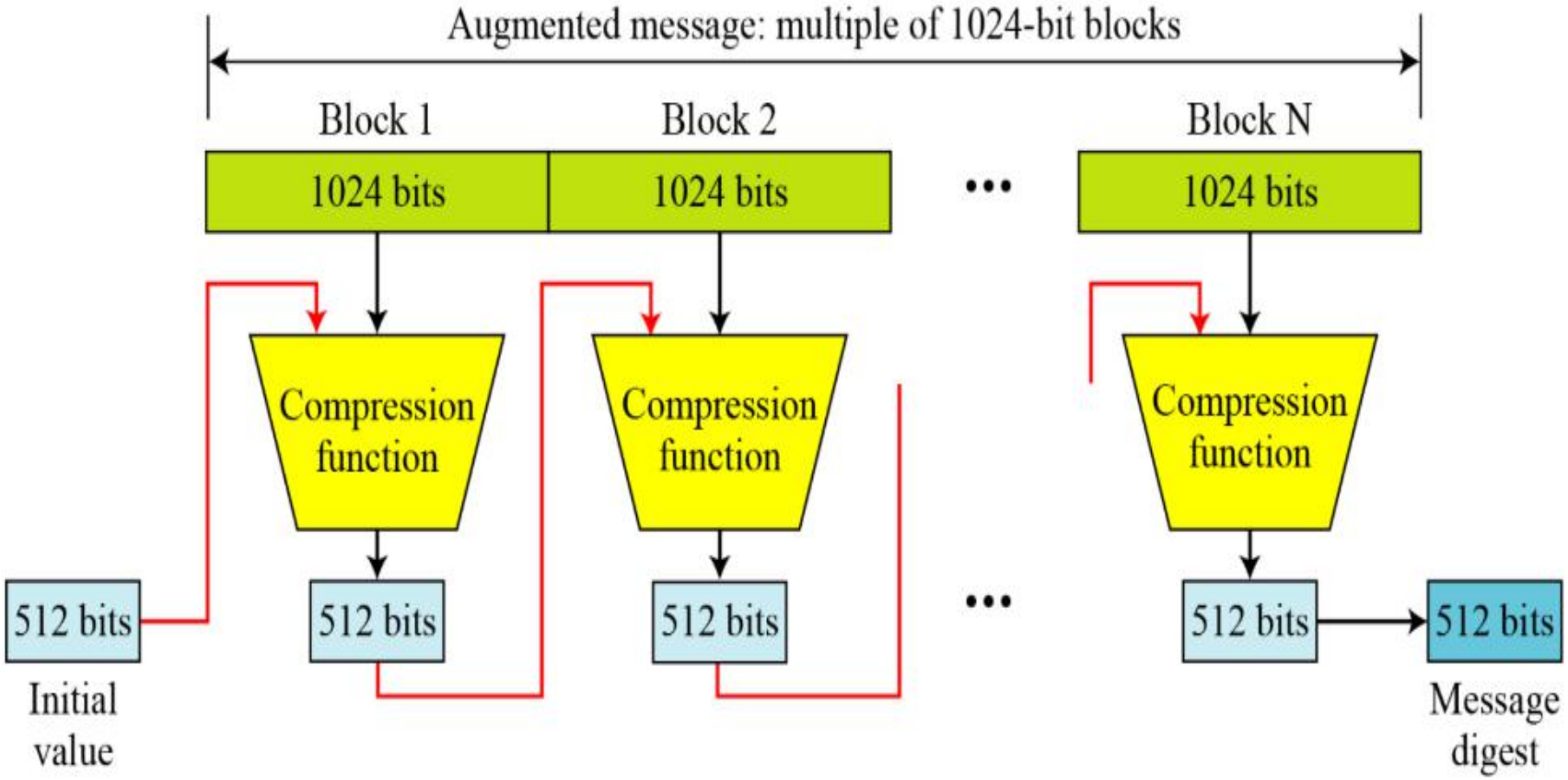


A hash function takes a variable sized input message and produces a fixed-sized output.

The output is usually referred to as the Hash Code or The Hash Value or The Message Digest.



Hash Function - SHA 512



Hash Function - Example

Input	Hash
sanya	834ac48d8e6d1d7f0b8d21a5b3e81446f5a4caa63765cc23836f61844b67fb83
SANYA	4247bff9d41c0f2da68ef43c5624531da9ca5bc31b39760a67e32265082e1ba8
Sanya	513a15ed036e62c14b41b2608a5bb18aa7af2a3502c90b892f9dddabaf136bc2

Input	Hash
	b48928ef0131d6fb61b5cee25163ae104a25f0edbd4230f2e7b3daa4a9b057d3
	043a718774c572bd8a25adbeb1bfcd5c0256ae11cecf9f9c3f925d0e52beaf89

Hash Function - Example

Sender^Receiver^Date^Nonce

input	d@blockchain.org.in^info@primechain.in^10092016^1
hash	288721860bec3a490811981c831702d4f41e54c3f8c183c5650ac73ff231659c

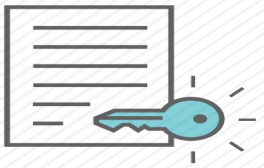
input	d@blockchain.org.in^info@primechain.in^10092016^2
hash	241e2b81192c0aa918c14f2896522428ccb77e937cade900d8f052ec3966c9cf

Hash begins with 4 zeros

... increase nonce till ....

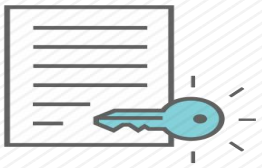
input	d@blockchain.org.in^info@primechain.in^10092016^66504
hash	00006bcc72f130eedbe9830c47e8d9f500d1e232540b03e095950aa798e2b97d

Digital Signature



Digital Signature

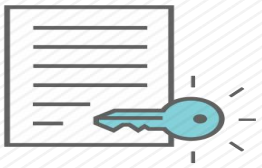
- *Encryption*, *message authentication* and *digital signatures* are all tools of modern cryptography.
- A signature is a technique for ***non-repudiation*** based on the public key cryptography.
- The creator of a message can attach a code, the signature, which guarantees the source and integrity of the message.



Properties Of Digital Signature

- Similar to handwritten signatures, digital signatures must fulfill the following:
  - ✓ Recipients must be able to verify them
  - ✓ Signers must not be able to repudiate them later

- In addition, digital signatures cannot be constant and must be a function of the entire document it signs.



Digital Signature

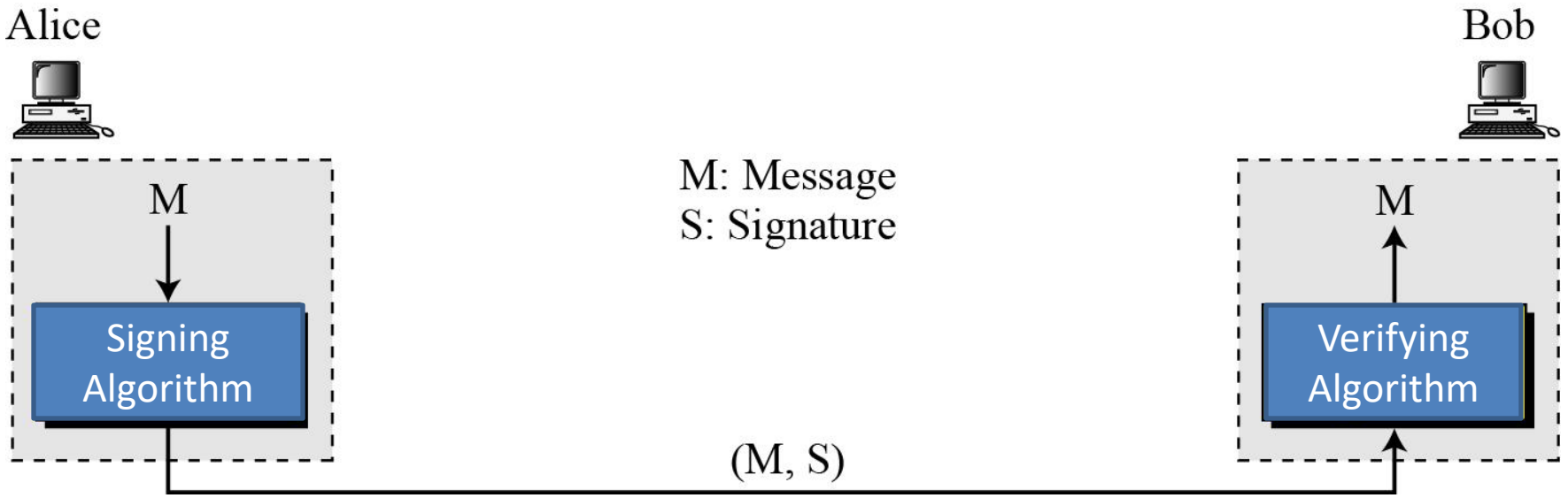
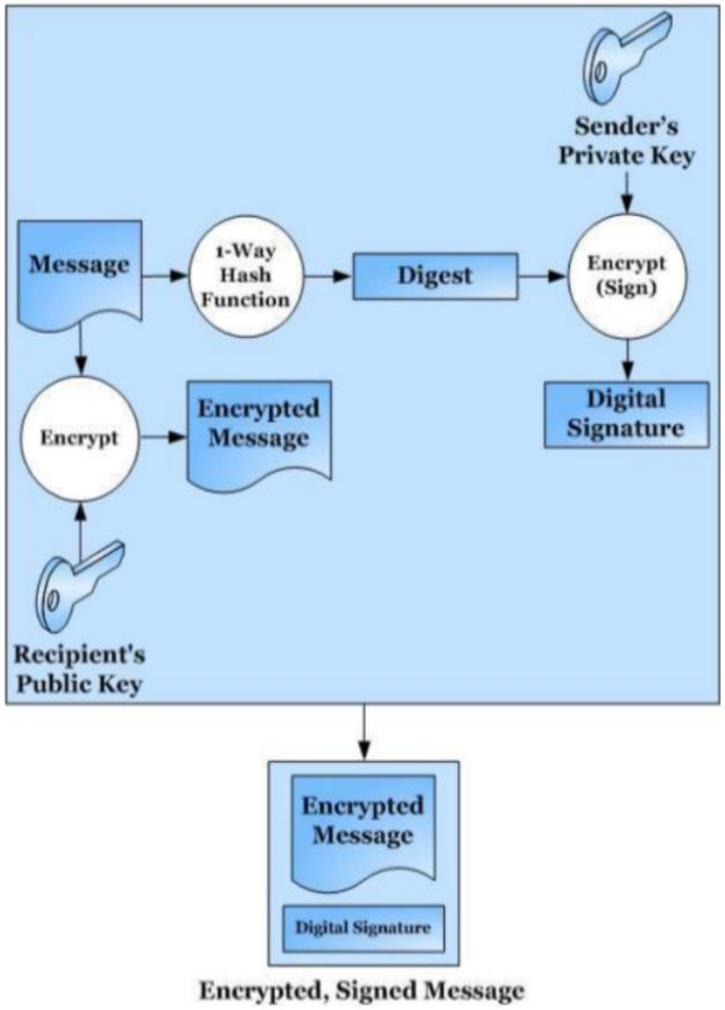
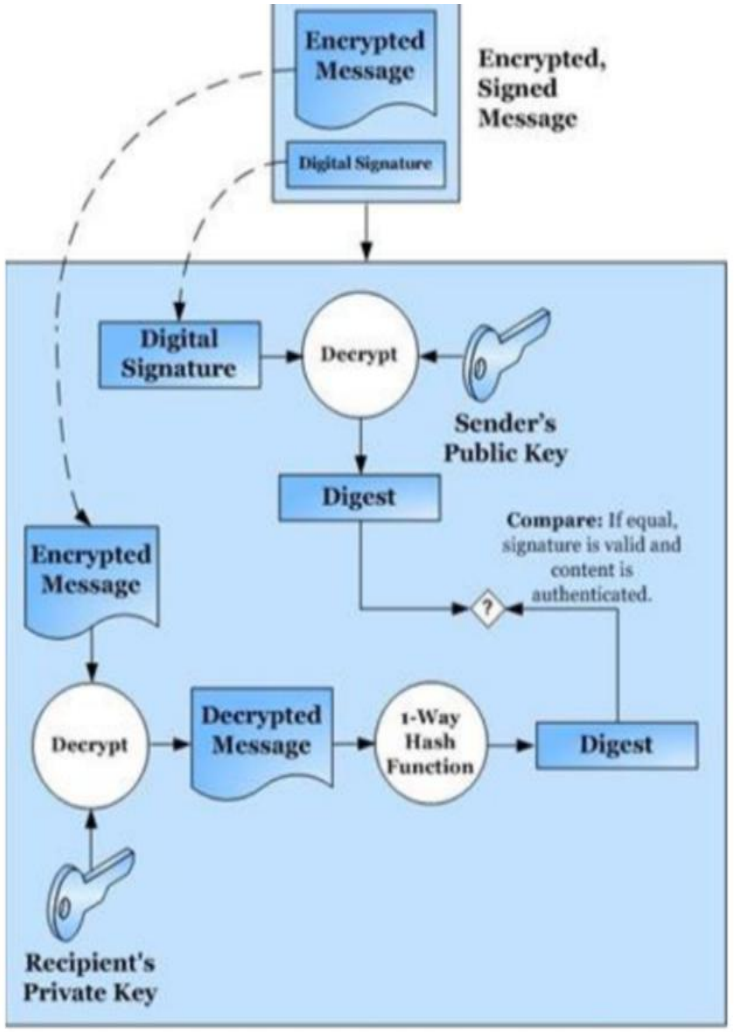


Fig-4 : Digital Signature Overview

Digital Signature





Digital Signature with Trusted Third Party

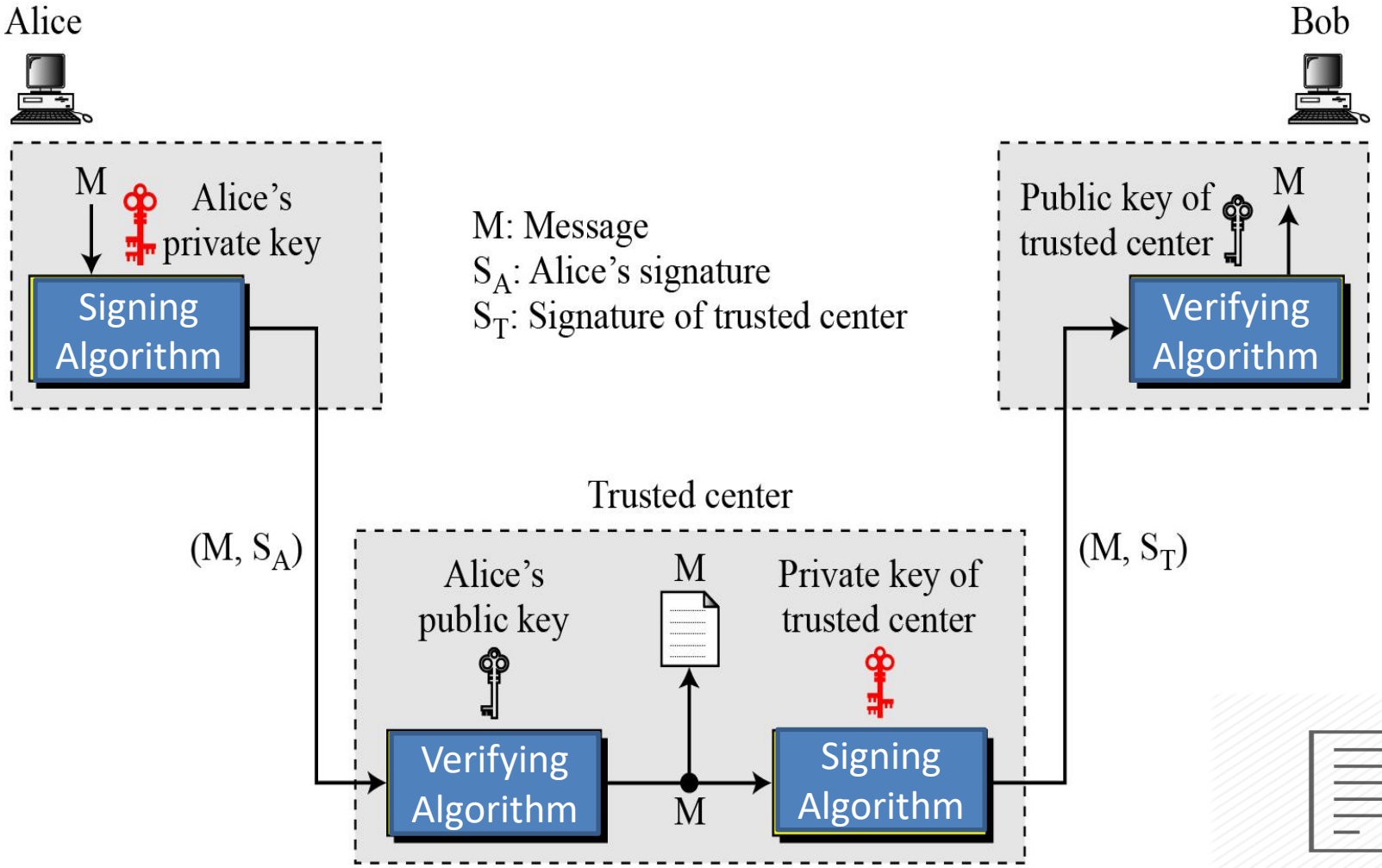


Fig-7 : Digital Signature by Trusted Third Party

Digital Signature Algorithm

Global Public-Key Components

p = A prime number of L bits where L is a multiple of 64 and $512 \leq L \leq 1024$

q = A prime divisor of(p-1)

$g = h^{(p-1)/q} \bmod p$, where h is any integer with $1<h< p-1$, such that $(h^{(p-1)/q} \bmod p)>1$

User's Private Key

x = Choose x s.t. $0<x<q$

User's Public Key

$y = g^x \bmod p$

User's Per-Message Secret Number

K = Choose K s.t. $0<k<q$

Signing

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

Signature = (r, s)

Verifying

$w = (s')^{-1} \bmod q$ $\quad$ $u_1 = [H(M')w] \bmod q$ $\quad$ $u_2 = (r')w \bmod q$

$v = [(g^{u1}y^{u2}) \bmod p] \bmod q$

Test: v = r'

Digital Signature Algorithm

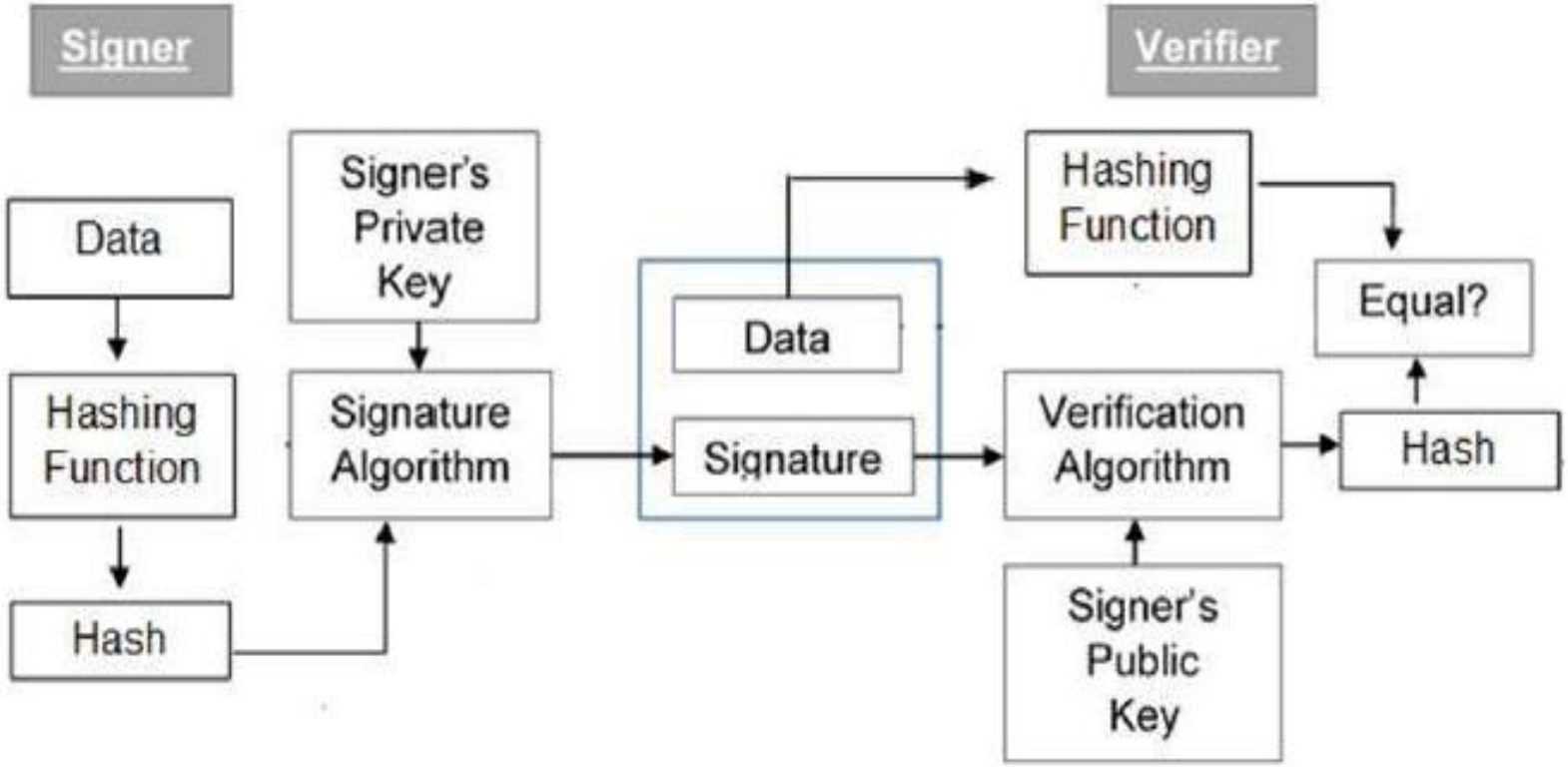


Fig-8 :DSS Approach

Alice Side : To sign a message, M:

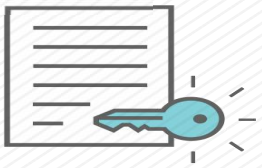
1. Alice chooses primes p and q.

p = A prime number of L bits where L is a multiple of 64 and $512 \leq L \leq 1024$

q = A prime divisor of(*p*-1)

Example : ✓ q = 5 ✓ P = 11

2. Alice uses $<Z_p^*, \times>$ and $<Z_q^*, \times>$.



Alice Side : To sign a message, M:

3. Alice genertaes the generator

   $g = h^{(p-1)/q} \bmod p$, where h is any integer with $1<h< p-1$, such that $(h^{(p-1)/q} \% p)>1$

   g = 3

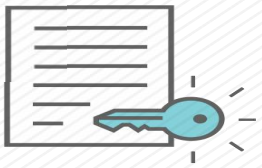
4. Alice chooses private key x *where* $0<x<q$.

   $x = 2$

5. *Alice Calculates the public key as*

   $y = g^x \bmod q$

   $y = 3^2 \bmod 11 = 9 \bmod 11 = 9$



Alice Side : To sign a message, M:

6. Alice's public key is (g, y, p, q); her private key is (x). Pub = (3,9,11,5) , Priv = (2)
7. Alice generates a random secret number , 'K', less than q. K = 2
8. Calculate message digest H(M) = 4
9. Alice generates ***Signature = (r, s) = (4,1)***

$r = (g^k \bmod p) \bmod q = (3^2 \bmod 11) \bmod 5 = 4$

$s = [\, k^{-1} (H(M) + xr) \,] \bmod q = 3\,(4 + 2*4) \bmod 5$

$= 36 \bmod 5 = 1$

$k^{-1} = 2^{-1} \bmod 5 = 3 \quad [2 * 3 \bmod 5 = 1]$

Bob Sides : Verification

(g, y, p, q) = (3,9,11,5) , ***(r, s) = (4,1)*** H(M) = 4

1. Bob verifies the signature by computing

$w = (s)^{-1} \bmod q = 1 \bmod 5 = 1$

$u_1 = [H(M)w] \bmod q = 4 * 1 \bmod 5 = 4$

$u_2 = (r)w \bmod q = 4 * 1 \bmod 5 = 4$

$v = [(g^{u1}y^{u2}) \bmod p] \bmod q$

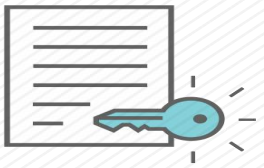
$= [(3^4.9^4) \% 11] \% 5 = ((81.6561) \%11) \%5$

$= 9 \bmod 5 = 4$

Test: v = r'

2. The signature is verifies if ***v = r'*** . ***4 = 4***

Any Queries ?



Thank You…