UNIT-III

➢ Public-Key Cryptography

- The development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography.
- Earlier all cryptographic systems have been based on the elementary tools of substitution and permutation.
- Public key algorithms are based on **mathematical functions** rather than on substitution and permutation.
- More important, **public-key cryptography is asymmetric**, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key.
- The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication, as we shall see.

Before proceeding, we should mention several common misconceptions concerning public-key encryption.

1. Public-key encryption is more secure from cryptanalysis than is symmetric encryption. In fact, the security of any encryption scheme depends on the length of the key and the computational work involved in breaking a cipher.
2. A second misconception is that public-key encryption is a general-purpose technique that has made symmetric encryption obsolete.
3. Finally, there is a feeling that key distribution is important when using public-key encryption, compared to the rather handshaking involved with key distribution centers for symmetric encryption.

Will now discuss the radically different **public key** systems, in which **two keys** are used. Anyone knowing the public key can encrypt messages or verify signatures, but **cannot** decrypt messages or create signatures. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

Terminology Related to Asymmetric EncryptionAsymmetric Keys

Two related keys, a public key and a private key, that are used to perform complementary operations, such as encryption and decryption or signature generation andsignature verification.

Public Key Certificate

A digital document issued and digitally signed by the private key of a Certification Authority that binds the name of a subscriber to a public key. The certificate indicates that the subscriber identified in the certificate has sole control and access to the corresponding private key.

Public Key (Asymmetric) Cryptographic Algorithm

A cryptographic algorithm that uses two related keys, a public key and a private key. The two keys have the property that deriving the private key from the public key is computationally infeasible.

Public Key Infrastructure (PKI)

A set of policies, processes, server platforms, software and workstations used for the purpose of administering certificates and public-private key pairs, including the ability to issue, maintain, and revoke public key certificates.

Principles of Public-Key Cryptosystems

The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption. The first problem is that of **key distribution.**

As we have seen, key distribution under symmetric encryption requires either

(1) Two communicants already share a key, which somehow has been distributed to them

or (2) The use of a key distribution center.

Whitfield Diffie, one of the discoverers of public-key encryption reasoned that this second requirement negated the very essence of cryptography: the ability to maintain to**tal secrecy** over your own communication.

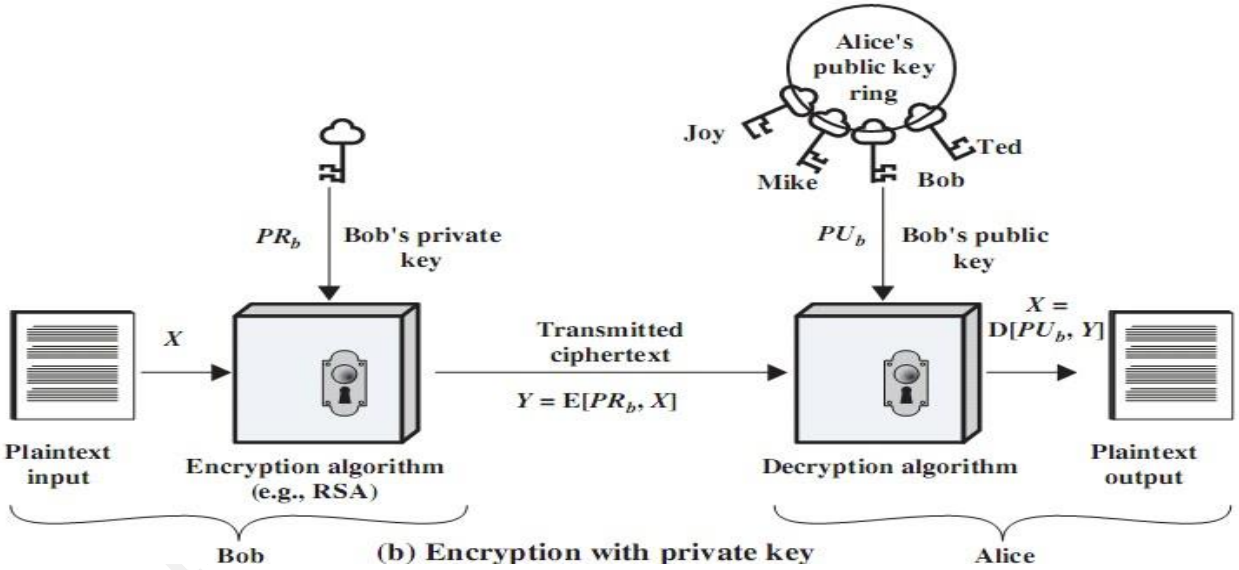
The second problem is **digital signatures.** The electronic messages and documents would need the equivalent of signatures used in paper documents. That is digital message had been sent by a particular person? This is a somewhat broader requirement than that of authentication,

Diffie and Hellman achieved breakthrough by coming up with a method that addressed both problems and was radically different from all previous approaches tocryptography.

Public-Key Cryptosystems

Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristic:

- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.



- Either of the two related keys can be used for encryption, with the other used fordecryption.

Anyone knowing the public key can encrypt messages or verify signatures, but **cannot** decrypt messages or create signatures, thanks to some clever use of number theory.

A public-key encryption scheme has six ingredients

- **Plaintext:** This is the readable message or data that is fed into the algorithm asinput.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
- **Public and private keys:** This is a pair of keys that have been selected so that ifone is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.

Ciphertext: This is the scrambled message produced as output. It depends on theplaintext and the key. For a given message, two different keys will produce two different ciphertexts.

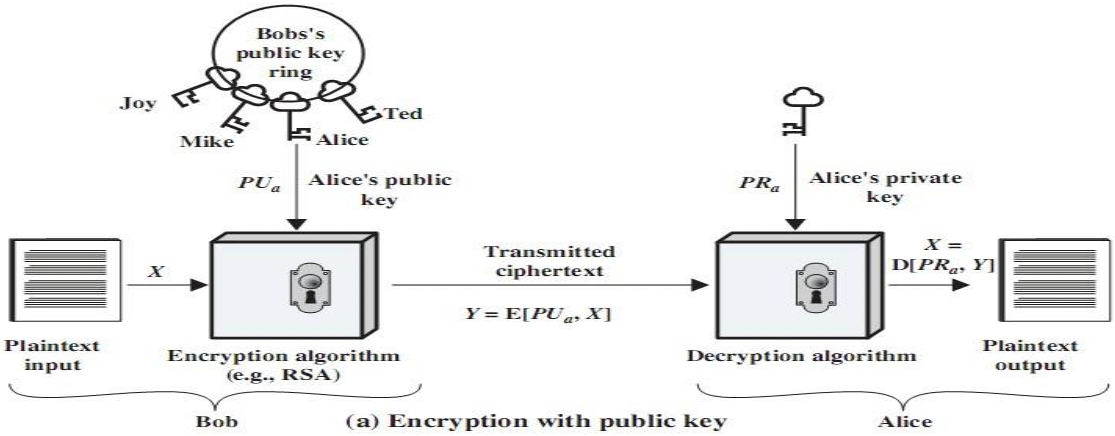
- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption ofmessages.

2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As Figure 9.1a suggests, each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

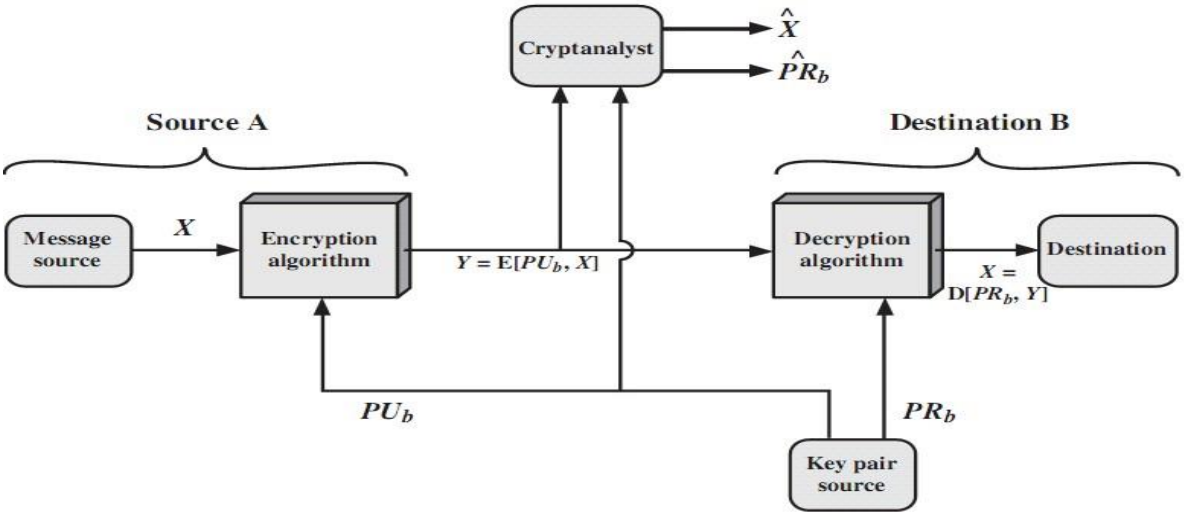
To discriminate between symmetric and public-key encryption, we refer to the key used in symmetric encryption as a **secret key**. The two keys used for asymmetric encryption are referred to as the **public key** and the **private key**. Invariably, the private key is kept secret, but it is referred to as a private key rather than a secret key to avoid confusion with symmetric encryption.



Conventional Encryption	**Public-Key Encryption**
Needed to Work:	Needed to Work:
1. The same algorithm with the same key is used for encryption and decryption. 2.The sender and receiver must share thealgorithm and the key.	1. One algorithm is used for encryption anddecryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one ofthe matched pair of keys (not the same one).
Needed for Security:	Needed for Security:
1. The key must be kept secret. 2. It must be impossible or at least 3. impractical to decipher a message 4. if noother information is available. 5. Knowledge of the algorithm plus 6. samples of ciphertext must be 7. insufficient to determine the key.	One of the two keys must be kept secret. It must be impossible or at least impractical todecipher a message if no other information is available. Knowledge of the algorithm plus one of thekeys plus samples of ciphertext must be insufficient to determine the other key.

Public-Key Cryptosystems: Secrecy and Authentication

 Public-key schemes can be used for either secrecy or authentication, or both (as shown here). There is some source A that produces a message in plaintext X. The message is intended for



destination B.

- B generates a related pair of keys: a public key, PUb, and a private key, PRb. PRb is known only to B, whereas PUb is publicly available and therefore accessible by A.
- With the message X and the encryption key PUb as input, A forms the ciphertext

  Y = E(PUb, X)
- The intended receiver, in possession of the matching private key, is able to invert the transformation: X = D(PRb, Y).
- An adversary, observing Y and having access to PUb, but not having access to PRb or X, must attempt to recover X and/or PRb. This provides confidentiality.
- Public-key encryption can also provide authentication:

  Y = E (PRa, X); X = D(PUa, Y)
- To provide both the authentication function and confidentiality have a double use of the public-key scheme (as shown here):

  Z = E (PUb, E (PRa, X))

  X = D (PUa, D (PRb, Z))

Applications for Public-Key Cryptosystems

We can classify the use of public-key cryptosystems into three categories:

- **Encryption/decryption:** The sender encrypts a message with the recipient'spublic key.
- **Digital signature:** The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.
- **Key exchange:** Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

Requirements for Public-Key Cryptography

1. It is computationally easy for a party B to generate a pair (public key PU_b, privatekey PR_b).
2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, M, to generate the corresponding ciphertext:

   $C = E(PU_b, M)$
3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

$$M = D(PR_b, C) = D[PR_b, E(PU_b, M)]$$

4. It is computationally infeasible for an adversary, knowing the public key, PU_b, to determine the private key, PR_b.
5. It is computationally infeasible for an adversary, knowing the public key, PU_b, and a ciphertext, C, to recover the original message, M.

   We can add a sixth requirement that, although useful, is not necessary for all public-key applications:

6. The two keys can be applied in either order:

$$M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$$

A **one-way function** is one that maps a domain into a range such that every function value has a unique inverse, with the condition that the calculation of the function is easy whereas the calculation of the inverse is infeasible:

$$Y = f(X) \text{ easy}$$
$$X = f^{-1}(Y) \text{ infeasible}$$

Trap-door one-way function

It is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known.

$$Y = f_k(X) \text{ easy, if k and X are known } X = f^{-1}(Y) \text{ easy, if k and Y are known}$$
$$X = f_k^{-1}(Y) \text{ infeasible, if Y known but k not known}$$

Public-Key Cryptanalysis

- As with symmetric encryption, a public-key encryption scheme is vulnerable to a **brute-force attack.** The countermeasure is the same: **Use large keys.**
- The key size must be large enough to make **brute-force attack impractical** but result in encryption/decryption speeds that are too slow for general-purpose use.
- Another form of attack is to find some way to **compute the private key** given the public key. To date, it has not been mathematically proven that this form of attack is infeasible for a particular public-key algorithm.
- Finally, there is a form of attack that is peculiar to public-key systems. This is, in essence, a **probable-message attack**.
- Suppose, for example, that a message were to be sent that consisted solely of a 56-bit DES key. An adversary could encrypt all possible 56-bit DES keys using the public key and could discover the encrypted key by matching the transmitted ciphertext.
- Thus, no matter how large the key size of the public-key scheme, the attack is reduced to a brute-force attack on a 56-bit key. This attack can be thwarted by **appending some random bits** to such simple messages.

➢ Introduction to Number Theory:

Number Theory plays an important role in encryption algorithm. Cryptography is the practice of hiding information, converting some secret information to not readable texts.

INTRODUCTION

For thousands of years people have searched for the way to send a message secretly. There is a story that, in ancient time, a king needed to send a secret message to his general in battle. The king took a servant, shaved his head and wrote the message on his head. He waited for the servant's hair to grow back and then sent the servant to the general. The general then shaved the servant's head and read the message. If the enemy had captured the servant, they presumably would not have known to shave his head and message would have been safe.

Cryptography is the study of methods to send and receive the secret messages. In general we have a

sender who is trying to send a message to receiver. There is also an adversary, who wants to steal the message. We are successful if sender is able to communicate a message to the receiver without adversary learning what the message was.

We will use some important concepts of Number Theory and Cryptography which are given below:

Important concepts in Number Theory

Prime Numbers- A positive integer *p* is said to be a prime if it has only two factors namely 1 and *p* itself. For Example: Primes are 2, 3, 5, 7, 11, 13, 17 …

Divisors: A positive integer a is said to divide an integer b if there exist an integer c such that b = a.c and written as $a \mid b$.

for example $2 \mid 10$ as 10 = 2.5 but 3 do not divide 10 as there does not exist any integer *c* such that 10 = 3. *c*

Greatest Common Divisor: Let a and b be two positive integers then an integer d is called greatest common divisor of a and b if if d/a and d is common divisor of a and b.

And if any integer c is such that c | a and c/b then c/d i.e any other common divisor of a and b will divide it denoted by d= (a, b). For Example: 6 = (24,30) Two numbers a and b are said to relatively prime or co prime if their greatest common divisor is 1 i.e. $(a, b) = 1$

For Example: 10 and 11 are co prime

Congruence: Let a and b be two integers and m is any positive integer then a is said to congruent to b modulo m if m divide difference of a and b i.e for m/a-b.it is denoted by $a \equiv b \pmod{m}$

➢ Prime Numbers

A central concern of number theory is the study of prime numbers. An integer p > 1 is a prime number if and only if its only divisors are ± 1 and $\pm p$ Prime numbers play a critical role in number theory.

Any integer a >1 can be factored in a unique way as $a = p1^{a1} * p2^{a2} * \ldots\ldots\ldots * pt^{at.}$

where p1 < p2 <……. < pt are prime numbers and where each ai is a positive integer. This is known as the fundamental theorem of arithmetic; a proof can be found in any text on number theory.

$91 = 13*7$

$3600 = 2^4 * 3^2 * 5^2$

$11011 = 7^8 * 11^2 * 13$

It is useful for what follows to express another way. If P is the set of all prime numbers, then any positive integer a can be written uniquely in the following form:

$$a = a = \pi_{p \in P} p^{a_p} \text{ where each } a_{p>} \geq 0$$

The right-hand side is the product over all possible prime numbers p; for any particular value of a, most of the exponents ap will be 0. The value of any given positive integer can be specified by simply listing all the nonzero exponents in the foregoing formulation.

➢ Modular Arithmetic (Clock Arithmetic)

Modular arithmetic is a system of arithmetic for integers, where values reset to zero and begin to increase again, after reaching a certain predefined value, called the modulus (modulo). Modular arithmetic is widely used in computer science and cryptography.

Definition Let Z_N be a set of all non-negative integers that are smaller than N:

$Z_N = \{0, 1, 2, ..., N-1\}$

where:

- **N** is a positive integer,
- if **N** is a prime, it will be denoted **p** (and the whole set as $\mathbf{Z_p}$).

To determine the value of an integer for a modulus N, one should divide this number by N. Its value in Z_N is equal to the remainder of the division. In modular arithmetic, it is possible to define all typical operations, as in normal arithmetic. They work as one may expect. It is possible to use the same commutative, associative, and distributive laws.

Modular inversion

Integers in modular arithmetic may (but not must) have inverse numbers.

Definition The inverse of x in Z_N is a number y in Z_N, that satisfies the equation:

$x \cdot y = 1$ (in Z_N)

where:

- **y** is denoted $\mathbf{x^{-1}}$

For example, if N is an odd number, then the inverse of 2 in Z_N is (N+1)/2:

$x \cdot (N+1)/2 = N + 1 = 1$ (in Z_N)

Theorem A number x is invertible in Z_N if and only if the numbers x and N are relatively prime.

- The theorem can be proved using the fact that it is possible to present the greatest common divisor of two integers as a sum of two products each of the numbers and another properly selected integer:

  $a \cdot x + b \cdot y = \gcd(x,y)$

Determining inverse numbers in Z_N allows solving linear equations in modular arithmetic: the equation: $a \cdot x + b = 0$ (in Z_N)

has the solution: $x = -b \cdot a^{-1}$ (in Z_N)

Definition The symbol Z^*_N denotes a set of all elements of Z_N that are invertible in Z_N; that means the set of numbers x that belong to Z_N, and x and N are relatively prime.

for example for a prime number p:

$Z^*p = Zp \setminus \{0\} = \{1, 2, \ldots, p - 1\}$

➢ Euler's Theorem

Euler's theorem states that for every a and n that are relatively prime:

$$a^{\phi(n)} \equiv 1 (\mathrm{mod}\, n)$$

Proof: Equation (8.4) is true if n is prime, because in that case, $\phi(n) = (n - 1)$ and Fermat's theorem holds. However, it also holds for any integer n. Recall that f(n) is the number of positive integers less than n that are relatively prime to n. Consider the set of such integers, labeled as

$R = \{x1, x2, \ldots, x\phi(n)\}$

That is, each element xi is a unique positive integer less than n with $\gcd(xi, n) = 1$. Now multiply each element by a, modulo n:

$S = \{(ax1 \bmod n), (ax2 \bmod n), \ldots, (ax\phi(n) \bmod n)\}$

The set S is a permutation6 of R, by the following line of reasoning:

1. Because a is relatively prime to n and xi is relatively prime to n, axi must also be relatively prime to n. Thus, all the members of S are integers that are less than n and that are relatively prime to n.

1. There are no duplicates in S. Refer to Equation (4.5). If $axi \bmod n = axj \bmod n$, then xi = xj.

Therefore

$$\prod_{i=1}^{\phi(n)} (ax_i \bmod n) = \prod_{i=1}^{\phi(n)} x_i$$

$$\prod_{i=1}^{\phi(n)} ax_i \equiv \prod_{i=1}^{\phi(n)} x_i \pmod{n}$$

$$a^{\phi(n)} \times \left[\prod_{i=1}^{\phi(n)} x_i\right] \equiv \prod_{i=1}^{\phi(n)} x_i \pmod{n}$$

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

which completes the proof. This is the same line of reasoning applied to the proof of Fermat's theorem.

$a = 3; n = 10; \phi(10) = 4 \quad a^{\phi(n)} = 3^4 = 81 = 1 \pmod{10} = 1 \pmod{n}$
$a = 2; n = 11; \phi(11) = 10 \quad a^{\phi(n)} = 2^{10} = 1024 = 1 \pmod{11} = 1 \pmod{n}$

As is the case for Fermat's theorem, an alternative form of the theorem is also useful:

$$a^{\phi(n)+1} \equiv a \pmod{n}$$

➢ Primary and Factorization

Applications of Prime Factorization

Prime factorization is used extensively in the real world. The two most important applications of prime factorization are given below.

- Cryptography and Prime Factorization
- HCF and LCM Using Prime Factorization

Cryptography and Prime Factorization

Cryptography is a method of protecting information using codes. Prime factorization plays an important role for the coders who create a unique code using numbers which is not too heavy for computers to store or process quickly.

In information security, large semi-primes have applications in encryption algorithms. They are used for generating public keys and private keys, such as in the Rivest–Shamir– Adleman (RSA) cryptosystems . The property that the prime factorization of large numbers is a challengingly difficult task is well utilized in RSA-based encryption algorithms. Due to the dominant application of the RSA public-key primitive in cryptography, the security of RSA has been extensively analyzed for various attack scenarios.

Special-purpose

A special-purpose factoring algorithm's running time depends on the properties of the number to be factored or on one of its unknown factors: size, special form, etc. The parameters which determine the running time vary among algorithms.

An important subclass of special-purpose factoring algorithms is the Category 1 or First Category algorithms, whose running time depends on the size of smallest prime factor. Given an integer of unknown form, these methods are usually applied before general-purpose methods to remove small factors.[9] For example, naive trial division is a Category 1 algorithm.

- Trial division
- Wheel factorization
- Pollard's rho algorithm
- Algebraic-group factorisation algorithms, among which are Pollard's p − 1 algorithm, Williams' p + 1 algorithm, and Lenstra elliptic curve factorization
- Fermat's factorization method
- Euler's factorization method
- Special number field sieve

➢ DISCRETE LOGARITHMS

Discrete logarithms are fundamental to a number of public-key algorithms, including Diffie-Hellman key exchange and the digital signature algorithm (DSA).

The Powers of an Integer, Modulo n

Recall from Euler's theorem that, for every a and n that are relatively prime,

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

where ϕ(n), Euler's totient function, is the number of positive integers less than n and relatively prime to n. Now consider the more general expression:

$$a^{m} \equiv 1 \pmod{n}$$

If a and n are relatively prime, then there is at least one integer m that satisfies Equation namely, M = ϕ(n). The least positive exponent m for which Equation (8.10) holds is referred to in several ways:

- The order of a (mod n)
- The exponent to which a belongs (mod n)
- The length of the period generated by a

To see this last point, consider the powers of 7, modulo 19:

$$\begin{aligned}
7^1 &\equiv & &\equiv 7 \pmod{19}\\
7^2 &= 49 = 2 \times 19 + 11 & &\equiv 11 \pmod{19}\\
7^3 &= 343 = 18 \times 19 + 1 & &\equiv 1 \pmod{19}\\
7^4 &= 2401 = 126 \times 19 + 7 & &\equiv 7 \pmod{19}\\
7^5 &= 16807 = 884 \times 19 + 11 & &\equiv 11 \pmod{19}
\end{aligned}$$

There is no point in continuing because the sequence is repeating. This can be proven by noting that $7^3 \equiv 1 \pmod{19}$, and therefore, $7^{3+j} \equiv 7^3 7^j \equiv 7^j \pmod{19}$, and hence, any two powers of 7 whose exponents differ by 3 (or a multiple of 3) are congruent to each other (mod 19). In other words, the sequence is periodic, and the length of the period is the smallest positive exponent m such that $7^m \equiv 1 \pmod{19}$.

Table 8.3 shows all the powers of a, modulo 19 for all positive a 6 19. The length of the sequence for each base value is indicated by shading. Note the following:

1. All sequences end in 1. This is consistent with the reasoning of the preceding few paragraphs.

2. The length of a sequence divides $\phi(19) = 18$. That is, an integral number of sequences occur i n each row of the table.

3. Some of the sequences are of length 18. In this case, it is said that the base integer a generates (via powers) the set of nonzero integers modulo 19. Each such integer is called a primitive root of the modulus 19.

Table 8.3 Powers of Integers, Modulo 19

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

More generally, we can say that the highest possible exponent to which a number can belong (mod n) is $\phi(n)$. If a number is of this order, it is referred to as a **primitive root** of n. The importance of this not ion is that if a is a primitive root of n, then its powers

$$a, a^2, \ldots, a^{\phi(n)}$$

are distinct (mod n) and are all relatively prime to n. In particular, for a prime number p, if a is a primitiv e root of p, then

$$a, a^2, \ldots, a^{p-1}$$

are distinct (mod p). For the prime number 19, its primitive roots are 2, 3, 10, 13, 14, and 15.
Not all integers have primitive roots. In fact, the only integers with primitive roots are those of the form 2, 4, pa, and 2pa, where p is any odd prime and a is a positive integer.

Logarithms for Modular Arithmetic

With ordinary positive real numbers, the logarithm function is the inverse of exponentiation. An analogous function exists for modular arithmetic.
Let us briefly review the properties of ordinary logarithms. The logarithm of a number is defined to be the power to which some positive base (except 1) must be raised in order to equal the number. That is, for base x and for a value y,

$$y = x^{\log_x(y)}$$

The properties of logarithms include

$$\log_x(1) = 0$$
$$\log_x(x) = 1$$
$$\log_x(yz) = log_x(y) + \log_x(z)$$
$$\log_x(y^r) = r \times \log_x(y)$$

Consider a primitive root a for some prime number p (the argument can be developed for non primes as well). Then we know that the powers of a from 1 through (p - 1) produce each integer from 1 through (p - 1) exactly once. We also know that any integer b satisfies

$$b \equiv a^i \pmod{p} \quad \text{where } 0 \le i \le (p-1)$$

by the definition of modular arithmetic. It follows that for any integer b and a primitive root a of prime number p, we can find a unique exponent i such that

$$b \equiv r \pmod{p} \quad \text{for some } r, \text{ where } 0 \le r \le (p-1)$$

This exponent i is referred to as the **discrete logarithm** of the number b for the base a (mod p). We denote this value as dloga,p(b).10

Note the following:

dloga,p(1) = 0 because a0 mod p = 1 mod p = 1

dloga,p(a) = 1 because a1 mod p = a

Here is an example using a nonprime modulus, $n = 9$. Here $\phi(n) = 6$ and $a = 2$ is a primitive root. We compute the various powers of a and find

$2^0 = 1 \quad 2^4 = 7 \pmod 9$

$2^1 = 2 \quad 2^5 = 5 \pmod 9$

$2^2 = 4 \quad 2^6 = 1 \pmod 9$

$2^3 = 8$

This gives us the following table of the numbers with given discrete logarithms (mod 9) for the root $a = 2$:

Logarithm	0	1	2	3	4	5
Number	1	2	4	8	7	5

To make it easy to obtain the discrete logarithms of a given number, we rearrange the table:

Number	1	2	4	5	7	8
Logarithm	0	1	2	5	4	3

Now consider

$$x = a^{\text{dlog}_{a,p}(x)} \bmod p \qquad y = a^{\text{dlog}_{a,p}(y)} \bmod p$$

$$xy = a^{\text{dlog}_{a,p}(xy)} \bmod p$$

Using the rules of modular multiplication,

$$xy \bmod p = [(x \bmod p)(y \bmod p)] \bmod p$$

$$a^{\text{dlog}_{a,p}(xy)} \bmod p = \left[\left(a^{\text{dlog}_{a,p}(x)} \bmod p\right)\left(a^{\text{dlog}_{a,p}(y)} \bmod p\right)\right] \bmod p$$

$$= \left(a^{\text{dlog}_{a,p}(x)+\text{dlog}_{a,p}(y)}\right) \bmod p$$

But now consider Euler's theorem, which states that, for every a and n that are relatively prime,

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

Any positive integer z can be expressed in the form $z = q + k\phi(n)$, with $0 \leq q < \phi(n)$. Therefore, by Euler's theorem,

$$a^z \equiv a^q (\text{mod } n) \qquad \text{if } z \equiv q \bmod \phi(n)$$

Applying this to the foregoing equality, we have

$$\text{dlog}_{a,p}(xy) \equiv [\text{dlog}_{a,p}(x) + \text{dlog}_{a,p}(y)](\text{mod}\phi(p))$$

and generalizing,

$$\text{dlog}_{a,p}(y^r) \equiv [r \times \text{dlog}_{a,p}(y)](\text{mod } \phi(p))$$

This demonstrates the analogy between true logarithms and discrete logarithms.
Keep in mind that unique discrete logarithms mod m to some base a exist only if a is a primitive root of m.
Table 8.4, which is directly derived from Table 8.3, shows the sets of discrete logarithms that can be defined for modulus 19.

Table 8.4 Tables of Discrete Logarithms, Modulo 19

Table 8.4 Tables of Discrete Logarithms, Modulo 19

(a) Discrete logarithms to the base 2, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{2,19}(a)$	18	1	13	2	16	14	6	3	8	17	12	15	5	7	11	4	10	9

(b) Discrete logarithms to the base 3, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{3,19}(a)$	18	7	1	14	4	8	6	3	2	11	12	15	17	13	5	10	16	9

(c) Discrete logarithms to the base 10, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{10,19}(a)$	18	17	5	16	2	4	12	15	10	1	6	3	13	11	7	14	8	9

(d) Discrete logarithms to the base 13, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{13,19}(a)$	18	11	17	4	14	10	12	15	16	7	6	3	1	5	13	8	2	9

(e) Discrete logarithms to the base 14, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{14,19}(a)$	18	13	7	8	10	2	6	3	14	5	12	15	11	1	17	16	4	9

(f) Discrete logarithms to the base 15, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$log_{15,19}(a)$	18	5	11	10	8	16	12	15	4	13	6	3	7	17	1	2	14	9

➢ Diffie Hellman Algorithm

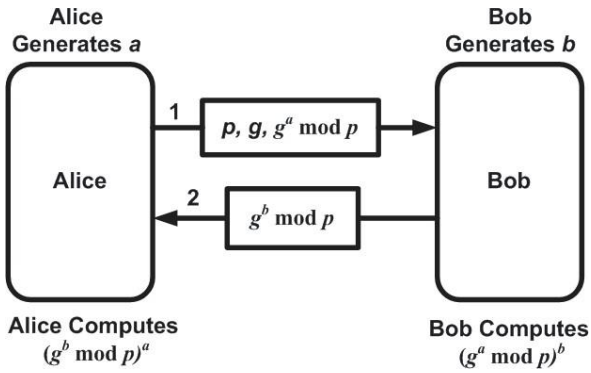
Diffie Hellman (DH) key exchange algorithm is a method for securely exchanging cryptographic keys over a public communications channel. Keys are not actually exchanged – they are jointly derived. It is named after their inventors Whitfield Diffie and Martin Hellman.
If Alice and Bob wish to communicate with each other, they first agree between them a large prime number p, and a generator (or base) g (where $0 < g < p$).

Alice chooses a secret integer a (her private key) and then calculates g^a mod p (which is her public key). Bob chooses his private key b, and calculates his public key in the same way.

Bob knows b and g^a, so he can calculate (g^a)^b mod p = g^ab mod p. Therefore both Alice and Bob know a shared secret g^ab mod p. An eavesdropper Eve who was listening in on the communication knows p, g, Alice's public key (g^a mod p) and Bob's public key (g^b mod p). She is unable to calculate the shared secret from these values.

In static-static mode, both Alice and Bob retain their private/public keys over multiple communications. Therefore the resulting shared secret will be the same every time. In ephemeral-static mode one party will generate a new private/public key every time, thus a new shared secret will be generated.

Diffie–Hellman Concept The Diffie–Hellman protocol is based on the discrete logarithm problem (DLP). The protocol requires two large numbers p and g, where, p and g are both publicly available numbers. Parameter p is a prime number with at least 512 bits and parameter g (called a generator) is an integer less than p, with the property: for every number n between 1 and p-1 inclusive, there is a power k of g such that $n = g^k \bmod p$. Suppose Alice and Bob want to agree on a shared secret key using the Diffie–Hellman key agreement protocol, they will generate shared key as follows: first, Alice generates a random private value a and Bob generates a random private value b. Both a and b are drawn from the same set of integers. Then they derive their public values using parameters p and g and their private values. Alice's public value is $x = g^a \bmod p$ and Bob's public value is $y = g^b \bmod p$. They then exchange public values x and y. Finally, Alice computes $ka = y^a \bmod p$, and Bob computes $kb = x^b \bmod p$. Since ka = kb = k, Alice and Bob now have a shared secret key k . Figure 1 depicts a simplified schematic of the Diffie–Hellman key exchange.



A. Diffie–Hellman Key Exchange Algorithm

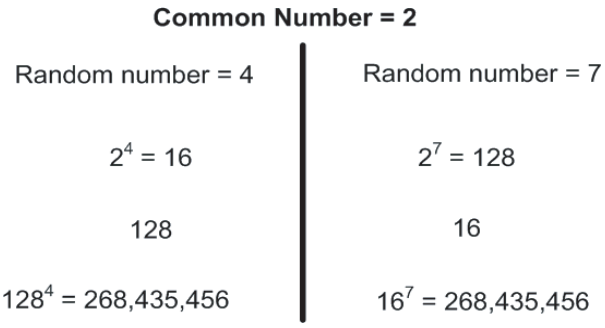
Let p be a large prime and assume that α is a primitive element of Z_p, p and α are publicly known [4].

1. Alice choose a $(0 \le a \le -2)$at random.
2. Alice computes $x = g^a \bmod p$ and sends it to Bob.
3. Bob chooses b $(0 \le b \le -2)$at random.
4. Bob computes $y = g^b \bmod p$ and sends it to Alice.
5. Alice computes $(g^b)^a \bmod p$ whereas Bob computes $(g^a)^b \bmod p$. In other words, both Alice and Bob compute the same key $g^{ab} \bmod p$. Figure 2 depicts an example of the Diffie–Hellman key exchange. If p = 170141183460469231731687303715884105727, then it would take roughly 1.14824×10^{21} steps to solve, and each step requires many calculations. Even using Google′ s computers which are estimated to perform 300 trillion calculations per second, it would take roughly 5 years to solve

B. Security of Diffie–Hellman If eavesdropper,

C. Mallory, learns integers p, g, x, y but not the discrete logarithm a of x and b of y to the base g. She wants to determine the secret key $k = g^{ab} \bmod p$ from p, g, x, y. This is called DiffieHellman problem. She can compute discrete logarithms mod p, and try to solve the Diffie-Hellman problem. She determines the discrete logarithm b of y to the base g and computes the key $k = x^b$.This is the only known method for breaking the DiffieHellman protocol. Until now, no one has succeeded in breaking Diffie-Hellman problem. It is an

important open problem of public-key cryptography to find such a proof. As long as the Diffie-Hellman problem is difficult to solve, no eavesdropper can determine the secret key from publicly known information



C. Applications of Diffie–Hellman in Network Protocols Diffie-Hellman is currently used in many network protocols, such as: • Secure Sockets Layer (SSL)/Transport Layer Security (TLS). • Secure Shell (SSH). • Internet Protocol Security (IPSec). • Public Key Infrastructure (PKI). • In all major VPN gateway's today (PKI).

INTRODUCTION

RSA is a public-key algorithm that based on mathematical functions rather than on substitution and permutation operations. RSA was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman and first published in 1978. The Rivest-Shamir-Adleman (RSA) scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption.

The security of RSA was based on the number of bits in private-key .The encryption and decryption in RSA requires taking heavy exponential multiplications modulus of a large integer N which is the product of two large primes' p and q .

The encryption and decryption time in RSA are roughly proportional to the number of bits in public and private exponents respectively. To reduce the encryption time, one may wish to use a small public exponent e, and to reduce the decryption time, a short secret exponent d can be used. In this context many RSA variants are designed.

The wide range of applications today raises the issue of what is the adequate security option that satisfies the application's security requirements. For applications applying encryption the question may be tailored to what RSA variant is more appropriate.

➢ OVERVIEW OF RSA VARIANTS

This section describes the Standard RSA and the RSA variants that had been designed to decrease the decryption time.

A. RSA standard

The original RSA cryptosystem was proposed in 1978 by Rivest, Shamir and Adelman and consists of three parts , described in the following subsections.

1. RSA Key generation

1- Generate two different primes p and q of $(n/2)$-bit each. 2- Compute $N = pq$ and $\emptyset(N) = (p-1)(q-1)$.Choose a random integer $1 < e < \emptyset(N)$ such that $\gcd(e,\emptyset(N)) = 1$.

3- Next, compute the uniquely defined integer $1 < d < \emptyset(N)$ satisfying $ed \equiv 1 \pmod{\emptyset(N)}$.The public key is $<N, e>$ and the private key $<N, d>$.

2. RSA Encryption

To encrypt a message X with the public key $<N, e>$, transform the message X to an integer M in $\{0,\ldots,N-1\}$ and compute the ciphertext $C = M^e \bmod N$.

3. RSA Decryption

To decrypt the ciphertext C with the private key $<N, d>$, compute $M = C^d \bmod N$ and employ

the reversetransformation to obtain the message *X* from *M* [3][1].

4. Complexity of RSA algorithm

The calculate the complexity of RSA algorithm, the exponentiations of the form $C^d \ mod\ N$ take time $O(dM(n))$,where $M(n)$ is the cost of multiplying two n-bit integers and can be taken as $O(n^2)$. And $|d| \approx |n|$, recall that the number of binary operations to compute $C^d \ mod\ N$ is $1.5n.n^2 = 1.5n^3$, thus these algorithms take time $O(n^3)$.

On another hand $Me\ (mod\ N)$ take time $O\ (eM\ (n))$, |e| is small, thus the encryption algorithms take time

$O(n^2)$. This means that encryption is much faster than decryption in RSA Standard.

B. Standard RSA with CRT

Based on the Chinese Remainder Theorem (CRTproposed an RSA variant, RSA-CRT, to speed up RSA decryption.

Here we review the basic steps constituting the RSA, together with the frequently used technique CRT. Suppose the message is M, the two large primes are p and q, the modulus is $N = pq$, and public key is $<N, e>$, private key is $<N, d>$ and to decrease decryption time private key is $<dp, dq, p, q>$ such that $dp \equiv d\ mod\ (p-1)$ and $dq \equiv d\ mod\ (q-1)$.

The encryption/decryption steps are as follows:

1. RSA-CRT Encryption

E (M): $C = M^e \ mod\ N$

2. RSA-CRT Decryption

Instead of directly computing $M = C^d \ mod\ N$, the decryption algorithm evaluates $Mp = C^{dp} \ mod\ p$ and $Mq = C^{dq} \ mod\ q$ where $dp = d\ mod\ p-1$ and $dq = d\ mod\ q-1$. It is then possible to recover the plaintext M using the Chinese remainder theorem. This method is faster because it computes two exponentiations of n/2- bit integers instead of one exponentiation of n-bit integers [1].

Algorithm: RSA decryption using CRTInput: C, p, q, N;

Output: M;

1. dp ←d mod(p - 1);
2. dq ←d mod(q - 1);
3. Mp ← C^{dp} mod p;
4. Mq← C^{dq} mod q;
5. pinv ← p^{-1} mod q;
6. M ←CRT(Mp ,Mq, p, q, (pinv), N);
7. return(M); [1]

RSA-CRT decryption Algorithm shows an optimization of RSA using the Chinese remainder theorem and the fast exponentiation algorithm. The two inverses that are normally needed for the Chinese remainder theorem are reduced in one inversion, which is pre-computed.

3. Complexity of RSA-CRT algorithm

The private key in RSA-CRT is <dp, dq>, the key length of dp and dq are (n/2). So RSA-CRT require twotimes $O((n)/2)^3)$ compared to the $O(n^3)$ decryption in RSA standard.
Thus, the theoretical speedup of RSA-CRT is

$$\text{RSA-CRT} = n^3/2(n/2)^3 = 4$$

Then, the decryption procedure of RSA-CRT is about 4 times faster than that of RSA standard

Rebalanced RSA-CRT

Based on the Chinese Remainder Theorem (CRT), Wiener suggested another RSA variant, Rebalanced RSA-CRT, to make the decryption time is much faster by carefully choosing *d* in the key generation phase such that $dp \equiv d \pmod{p-1}$ and $dq \equiv d \pmod{q-1}$ are small.
One first selects two small CRT-exponents *dp* and *dq*, and then these two CRT-exponents are combined to get the secret exponent. At last, computes the corresponding public exponent ***e*** satisfying $ed \equiv 1 \pmod{\varphi(N)}$ [5]. This variant of RSA enables rebalances the costs of encryption and decryption. In other words, it speeds up the RSA decryption by shifting the decryption cost to the encryption cost.

In Rebalanced RSA-CRT, both d and e will be of the same order of magnitude as φ (N). The decryption time depends on the bit-size of dp and dq, while not on the bit-size of d, the decryption time is minimized. But the encryption time depends on the bit-size of **e**, this will make the encryption for the original Rebalanced RSA-CRT very time-consuming.

The algorithm takes two security parameters as input: n (typically 1024) and k (typically 160) where k < n/2. It next picks two primes p and q verifying gcd (p−1, q−1) = 2 and whose bit length is n/2. The modulus N is N = pq. It then randomly picks two k-bit values dp and dq such that gcd (dp, p−1) = 1, gcd (dq, q−1) = 1 and dp = dq mod 2. The secret exponent d must verify: d = dp (mod p−1) and d = dq (mod q−1). Cannot directly compute d with the Chinese remainder theorem because p−1 and q−1 are not relatively prime (they are both even). But we chose them such that gcd (p −1, q−1) = 2, therefore:

$$gcd\left(\frac{p-1}{2}, \frac{q-1}{2}\right) = 1$$

We also know that $dp = dq \pmod 2$. To compute the public exponent e, we just have to compute the inverse of *d* modulo Ø (N) = (p − 1) (q − 1). This is allowed because gcd (dp, p − 1) = gcd ($dq, q-1$) = 1. Therefore, gcd (d, p −1) = gcd (d, q−1) = 1. And finally gcd (d, (p −1)(q−1)) = 1. We have no control over ***e*** which is of the order of *N*. The encryption won't be as fast as in standard RSA but we manage to increase the speed of the decryption stage.
Now we describe three constituent algorithms: key generation, encryption, and decryption.

1. Rebalanced RSA-CRT Key generation:

The key generation of the Rebalanced RSA is as follows:

Algorithm: Key Generation in Rebalanced RSA-CRT
Input: N, k;
Output: p, q, dp, dq, e, d;

1. Randomly select two 512-bit primes p= 2p1+1 and q= 2q1+1 such that gcd (p1, q1) = 1.
2. Compute $p1inv = p1^{-1} \pmod{q1}$
3. Randomly select two distinct odd numbers dp and dq of 160 bits such that gcd (dp, p−1) =1 and

gcd (dq, q − 1) = 1.

4. Compute d = CRT (dp, dq, p1, q1, p1 inv, N).

5. Compute $e \equiv d^{-1} \bmod (p-1)(q-1)$.

6. Return (p, q, dp, dq, e, d). [5]

The public key is given by the pair *(e, N)* and the private key is given by the tuple *(dp, dq, p, q)*. Note that *e* will be roughly the same order of magnitude as *Ø(N)*. Thus, in order to reduce the decryption time even further than RSA-CRT, the encryption time is essentially maximized.

2. Rebalanced RSA-CRT Encryption

This is exactly the same as in standard RSA, that is $C \equiv (mod\ N)$, except that e is much larger. The public key is (N, e). The complexity of this algorithm Take time $O(e\ M(n))$, where $M(n)$ is the cost of multiplying two n-bit integers and can be taken as $O(n^2)$. And $|e| \approx |n|$, thus these algorithms take time $O(n^3)$, compare to $O(n^2)$ in RSA Standard

3. Rebalanced RSA-CRT Decryption

The decryption is the same as the CRT decryption. The main difference is that in Rebalanced RSA-CRT, the CRT-exponents *dp* and *dq* are only of *160* bits which are much shorter than the CRT exponents of *512* bits in RSA-CRT. Thus, the decryption in Rebalanced RSA-CRT is about *512/160 = 3.2* times faster than that in RSA-CRT.

The private key is *(p, q, dp, dq)*. Can decrypt a cipher text *C* by Using the Chinese remainder theorem, we are then able to recover the plaintext M, verifying M = Mp mod P and M = Mq mod q. as follows: Algorithm: Rebalanced RSA-CRT Decryption

Input: N, k;

Output: C, N, dp, dq, p, q;

1. Mp ← C dp mod p;
2. Mq ← C dq mod q;
3. pinv ← P -1 mod p;
4. M ←CRT(Mp, Mq, p, q, (pinv), N);
5. Return(M); [9]

Complexity of Rebalanced RSA-CRT algorithm

Recall that dp, dq in Rebalanced RSA are k-bits each. The cost of modular multiplications is the same as that of RSA-CRT; the main difference is the number of multiplications computed during each modular exponentiation.

The decryption in Rebalanced RSA require tow times $O(k(n/2)^2)$. Compared to the $O(n^3)$ decryption ofthe RSA standard, then a theoretical speedup of Rebalanced RSA is

Rebalanced RSA-CRT = $n^3/2k(n/2)^2 = 2n/k$

So, for modulo of 1024 bits with k = 160, Rebalanced RSA is theoretically 12.8 Faster than RSA standard and 3.2 faster than RSA-CRT.

IMPROVING REBALANCED RSA-CRT ENCRYPTION TIME

According to the key generation in the original Rebalanced RSA-CRT, if we first select the small CRT- exponents *dp* and *dq*, the public exponent *e* will be of the same bit-size as modulus *Ø(N)*. This causes heavy encryption cost. If we can make the public exponent *e* much shorter than *Ø(N)*, it will be more convenient and practical in many applications.

Two variants of Rebalanced RSA-CRT, Scheme A and Scheme B have been design to achieve the above goal [5]. The two key generation algorithms for Rebalanced RSA-CRT which generates public

exponents much smaller than *Ø(N)*. Each key generation algorithm is based on the following fundamental theorem from number theory.

Theorem 3.1: If a and b are relatively prime, i.e. *gcd(a, b) = 1*, then we can find a unique pair *(u , v)*

satisfying $au - bv = 1$, for any integer $u,v \in z$ [7] .

A. Rebalanced RSA-CRT Scheme A

The first key generation algorithm, scheme A, produces a *568-bit* public exponent, e.g., $e = 2^{567} + 1$, andtwo 160-bit CRT-exponents *dp, dq*. The key generation of the algorithm is as follows:

Algorithm: Key Generation in Scheme AInput: *n, k*;

Output *dp, dq, p, q, e;*

1. Randomly select an odd number ***e*** of *568* bits.
2. Randomly select a number *kp1* of *160* bits, such that *gcd (kp1, e) = 1*.
3. Based on Theorem 3.1, we can uniquely determine two numbers *dp, kp1 < dp < 2 kp1*, and P, *e < P <2e*, satisfying *e dp = kp1 P + 1.*
4. Factor *P* as $P = kp2 \cdot p'$ such that *kp2* is a number of *56* bits and p= (p` +1) is a prime number. If this isinfeasible, then go to Step 2.
5. Randomly select a number *kq1* of *160* bits, such that gcd (*kq1*, e) = 1.
6. Based on Theorem 3.1, we can uniquely determine two numbers *dq,kq1 < dq <2 kq1*, and q, *e < q < 2e,*satisfying *e dq = kq1 q + 1.*
7. Factor *q* as $q = kq2 \cdot q'$ such that *kq2* is a number of *56* bits and q= (q´ +1) is a prime number. If thisis infeasible, then go to Step 5.
8. Return *(dp, dq, p, q, e).* [5]

Complexity of Rebalanced RSA-CRT Scheme A

The complexity of encryption in Rebalanced is the same as the complexity of decryption in RSA, because the public exponent *e* will be of the same bit-size as modulus *Ø(N)*.then *C* = mod *N* take $1.5\ n^3 = O(n^3)$. Scheme A, use a *512*-bit public exponent, thus Scheme A take $k.n^2 = O(kn^2)$, where k is bit length of public exponent. The theoretical speedup of Scheme A is

Schema A = $1.5\ n^3 / kn^2 = 1.5\ n/k$

So, for modulo of 1024 bits with k = 568, Scheme A is theoretically 2.7 Faster than Rebalanced RSA-CRT.

B. Rebalanced RSA-CRT Scheme B

The second key generation algorithm, called Scheme B. In Scheme B, The Key Generation produces a *512*- bit public exponent, e.g., $e = 2^{511} + 1$, and two *198-bit* CRT-exponents *dp, dq*. The encryption is about *3* times faster than that of Rebalanced RSA-CRT [5].

Algorithm: Key Generation in Scheme B Input: *n, k;*

Output *dp, dq, p, q, e;*

1. Randomly select an odd number *e* of *512* bits.
2. Randomly select an odd number *kp* of *198* bits, such that *gcd (kp, e) = 1*.
3. Based on Theorem 1, we can uniquely determine two numbers dp, kp < dp < 2kp, and p´, e < p´ < 2e, satisfying *e.dp – kp.p´ = 1.*

4. If $p = p' + 1$ is not a prime number, then go to Step 2.
5. Randomly select an odd number kq of 198 bits, such that $gcd(kq, e) = 1$.
6. Based on Theorem 3.1, we can uniquely determine two numbers dq, $kq < dq < 2kq$, and q', $e < q' < 2e$, satisfying $e\,dq - kq\,q' = 1$.
7. If $q = q' + 1$ is not a prime number, then go to Step 5.
8. The public key is (N, e); the secret key is (dp, dq, p, q). [5]

The key generation algorithm for this scheme is much more efficient than the one in Scheme A. The runtime of the algorithm is dominated by two loops, like Scheme A, but each iteration requires much less computational effort as there is no factoring required [7].

1. Complexity of RSA-CRT Rebalanced Scheme B

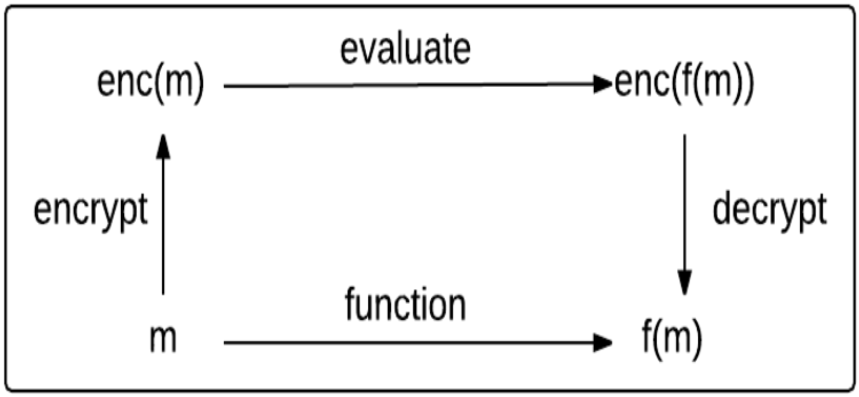
The Complexity of Scheme B the same as in Scheme A, the main different is bit length of e = 512

$$Scheme\ B = \frac{1.5n^3}{kn^2} = 1.5\ n/k$$

So, for modulo of 1024 bits with k = 512, Scheme B is theoretically 3 times faster than Rebalanced RSA- CRT. But the decryptions in two Schemes are a little slower than that of Rebalanced RSA-CRT.

➢ Homomorphic Encryption techniques

Homomorphic encryption is a cryptographic method that allows mathematical operations on data to be carried out on cipher text, instead of on the actual data itself. The cipher text is an encrypted version of the input data (also called plain text). It is operated on and then decrypted to obtain the desired output. The critical property of homomorphic encryption is that the same output should be obtained from decrypting the operated cipher text as from simply operating on the initial plain text.



The following graphic explains how homomorphism is used in encryption. The process begins with some plain text message, mm. It could be the name of someone in a computer system, "jane doe". The goal is to perform some function ff on it, like capitalizing the name. That's possible, but it would be safer to encrypt the message using enhance before performing any functions on it so the person performing the function never learns Jane's name. So the message is encrypted to some cipher text 163726. Then, it is evaluated, or transformed, into another value using some other

function, f(x) = x + 127f(x)=x+127, for example. The output, 163853, is another completely encrypted message. This message can then be decrypted to "Jane Doe".

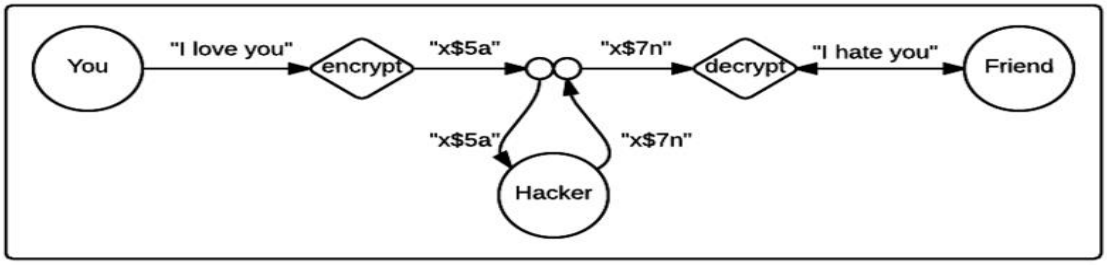
Current homomorphic encryption exists in partial and full forms. Partially homomorphic systems only allow certain operations on encrypted data, typically multiplication or division, and have existed for many years. Fully homomorphic systems, which allow all operations on encrypted data, remained an open problem in cryptography for 30 years, but they were finally solved in 2009. It is typically desirable because it allows a system to chain together multiple operations that are all performed on encrypted data so as not to expose unencrypted data.

In cryptography, it is assumed that there is some sort of communication going on between two people. Let's call them Alice and Bob. Maybe Alice wants to give Bob a suitcase full of money to count and give back to her. Alice hands Bob the suitcase, and Bob asks for the key to open it. Alice doesn't want to give him the key because she's not very trusting. Bob, however, doesn't know how to count the money if Alice won't let him open the box. So, the couple have two options:

1. Alice could give Bob the key. This would work. However, there is also an attacker out there who wants to steal the money, Eve. She could be impersonating Bob, or she could steal both the suitcase and key from Bob later.
2. Devise a new kind of lock, even better than a suitcase that needs a key. This lock won't let anyone who isn't Alice get to the money or even know how much money is in there. But it will allow anyone with the right permission to count it. This is the concept behind homomorphic encryption and why it's so powerful.

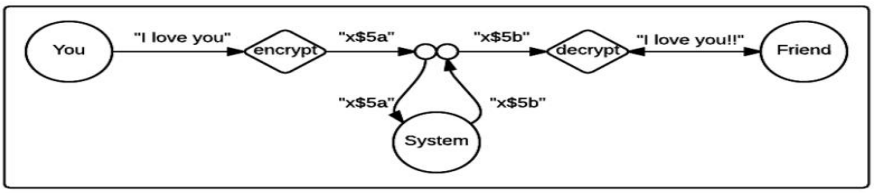
Properties

Homomorphic encryption is **malleable** by design. A malleable crypto-system is one in which anyone can intercept a cipher text, transform it into another cipher text, and then decrypt that into a plain text that makes sense. Malleability is generally considered undesirable in a crypto-system. Imagine you're trying to send the message "I love you" to your friend using encryption. You encrypt it and send it off. But, it is intercepted by a hacker on the way. All they see is some cipher text, but they can change that cipher text to something that will decrypt to "I hate you" when your friend tries to decrypt it. That is why malleability to not usually wanted.



A malleable crypto-system

However, homomorphic systems should be malleable. Maybe You want to build a system that simply adds exclamation marks to whatever you send your friend. But, you don't want the system to know what you're sending your friend, that's a secret. You encrypt your message and send it along to your system. Your system only sees the cipher text, so it doesn't know what you're saying, but it can still transform the cipher text. It transforms the cipher in a text that will decrypt to the same input message, plus a few exclamation marks. So, now it looks like this.



Another malleable crypto-system

Malleable systems allow for multiple parties, especially in cloud-based environments, to operate on data without ever exposing it.

Applications

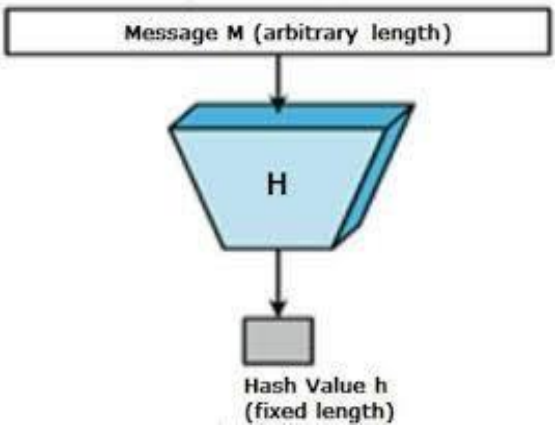
Homomorphic encryption has seen revitalization in recent years, especially due to cloud computing. Resources for computation are so cheap that many system outsource their computation to companies like Amazon or Microsoft and their huge data centers. However, those systems don't want to expose that data for attack. So, they encrypt the data, send it to a data center, it is operated on, and then it is send back to the original system for decryption.

Privacy is very important in today's computing environment. Private information that is used by various systems often uses homomorphic encryption if that data needs to stay encrypted. E-voting, for example, might want to use homomorphic encryption to protect voting privacy.

E-cash systems might also use homomorphic encryption. If you send money to a friend, you might not want the system to know how much money you are sending. The system could retrieve both you and your friend's encrypted data, add them together, and then send them back to your friend.

➢ Message Authentication and Hash Functions

A **hash function** H accepts a variable-length block of data M as input and produces a fixed-size hash value h = H(M). A "good" hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and apparently random. In general terms, the principal object of a hash function is data integrity. A change to any bit or bits in M results, with high probability, in a change to the hash value.



The kind of hash function needed for security applications is referred to as a **cryptographic hash function**. A cryptographic hash function is an algorithm for which it is computationally infeasible (because no attack is significantly more efficient than brute force) to find either

(a) a data object that maps to a pre-specified hash result (the one-way property) or

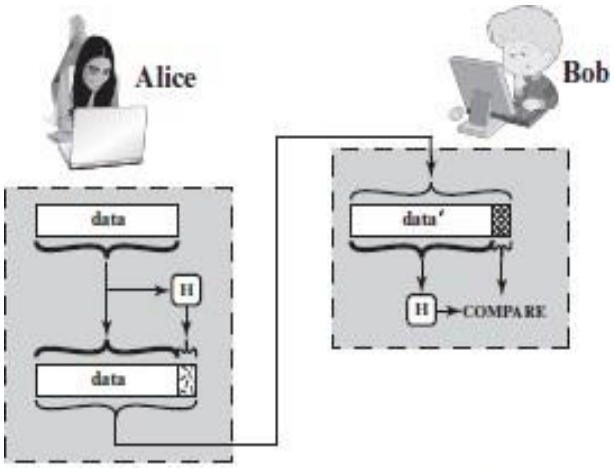
(b) two data objects that map to the same hash result (the collision-free property).

Because of these characteristics, hash functions are often used to determine whether or not data has changed.

Figure: Cryptographic Hash Function; h = H(M)

Message Authentication:

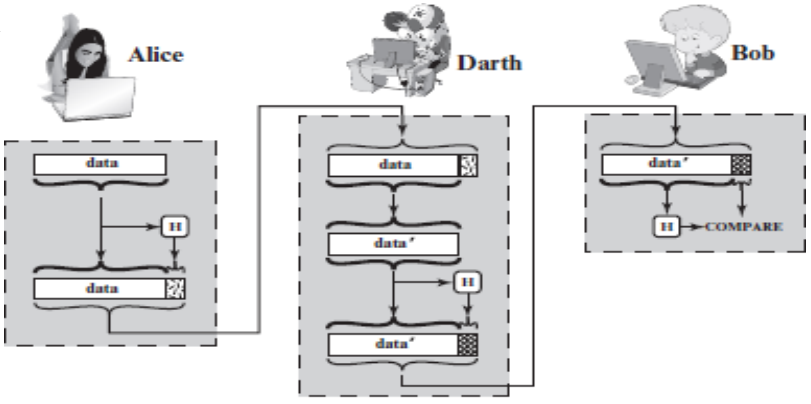
- Message authentication is a mechanism or service used to verify the integrity of a message.
- Message authentication assures that data received are exactly as sent (i.e., there is no modification, insertion, deletion, or replay).
- In many cases, there is a requirement that the authentication mechanism assures that purported identity of the sender is valid.
- When a hash function is used to provide message authentication, the hash function value is often referred to as a message digest.
- The essence of the use of a hash function for message integrity is as follows.
- The sender computes a hash value as a function of the bits in the message and transmits both the hash value and the message. The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value.
- If there is a mismatch, the receiver knows that the message (or possibly the hash value) has beenaltered.
- The hash value must be transmitted in a secure fashion. That is, the hash value must be protected so that if an adversary alters or replaces the message, it is not feasible for adversary to also alter the hash value to fool the receiver. This type of attack is shown in Figure (b).
- In this example, Alice transmits a data block and attaches a hash value. Darth intercepts the message, alters or replaces the data block, and calculates and attaches a new hash value. Bob receives the altered data with the new hash value and does not detect the change. To prevent this attack, the hash value generated by Alice must be protected.



(a) Use of hash function to check data integrity

Figure below illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows.

a) The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.



(b) Man-in-the-middle attack

b) Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

c) It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S. A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because B

possesses S, it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

d) Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.

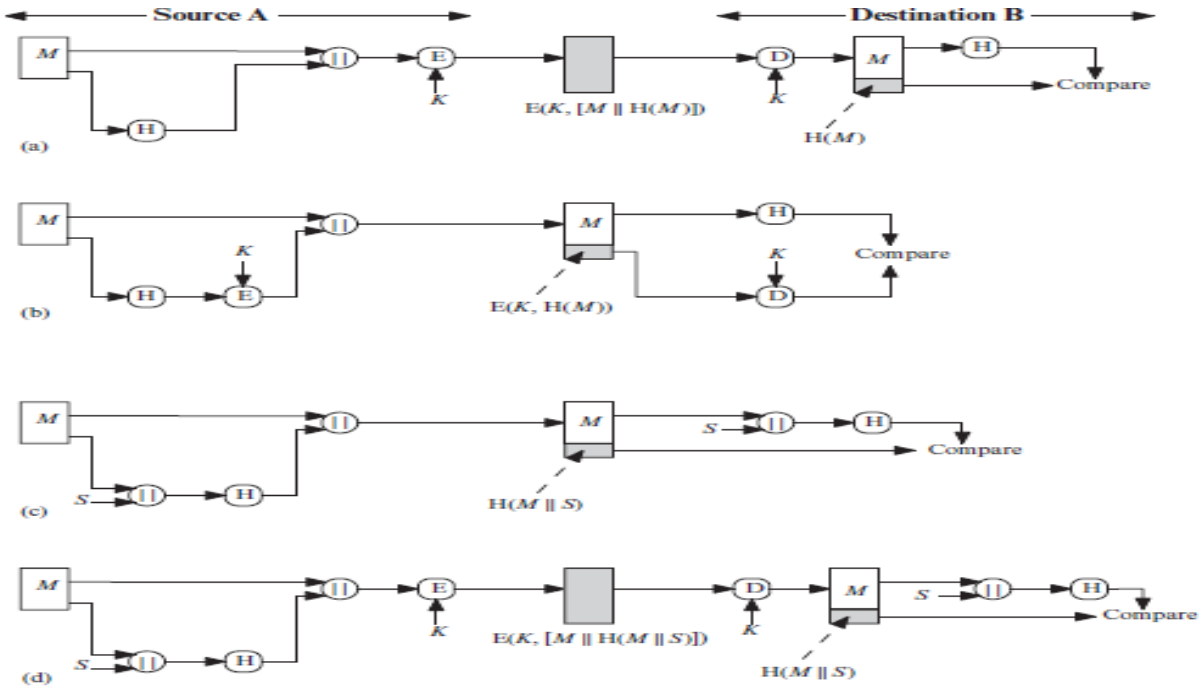


Figure 11.3 Simplified Examples of the Use of a Hash Function for Message Authentication

When confidentiality is not required, method (b) has an advantage over methods (a) and (d), which encrypts the entire message, in that less computation is required. Nevertheless, there has been growing interest in techniques that avoid encryption.

Several reasons for this interest are pointed out as:

- Encryption software is relatively slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.
- Encryption hardware costs are not negligible. Low-cost chip implementations of DES are available,but the cost adds up if all nodes in a network must have this capability.
- Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.
- Encryption algorithms may be covered by patents, and there is a cost associated with licensingtheir use.

Message Authentication Functions:

The message authentication function is concerned with the types of functions that may be used to pro- duce an authenticator. These may be grouped into three classes:

- Hash function: A function that maps a message of any length into a fixed- length hash value, whichserves as the authenticator.
- Message encryption: The ciphertext of the entire message serves as its authenticator.
- Message authentication code (MAC): A function of the message and a secret key that produces afixed-length value that serves as the authenticator.

Message Authentication Code (MAC):

A authentication technique that involves the use of a secret key to generate a small fixed-size block of data that is appended to the message is known as Message Authentication Code (MAC). This technique assumes that two communicating parties, say A and B, share a common secret key K. When A has a message to send to B, it calculates the MAC as a function of the message and the key:

MAC = C (K, M)

where

M = input messageC = MAC function

K = shared secret key

MAC = message authentication code

The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC.

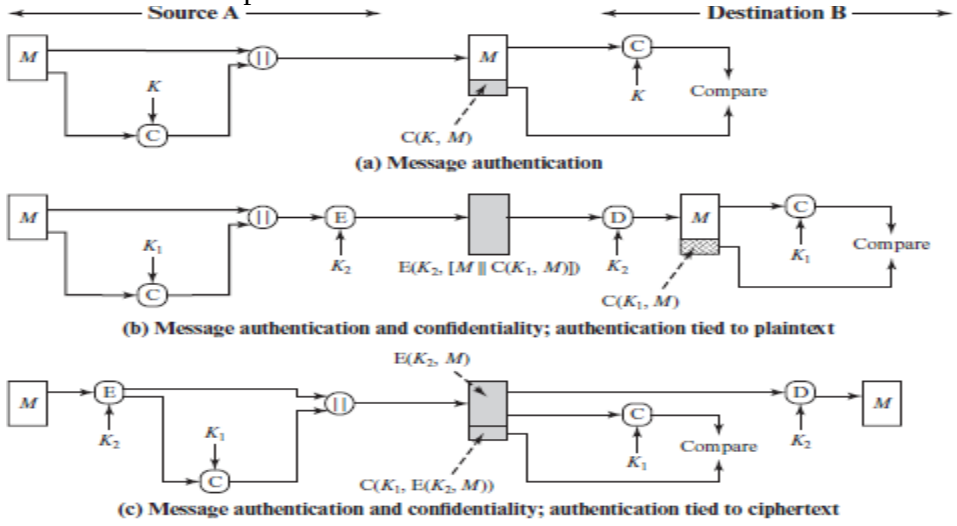


Figure 12.4 Basic Uses of Message Authentication code (MAC)

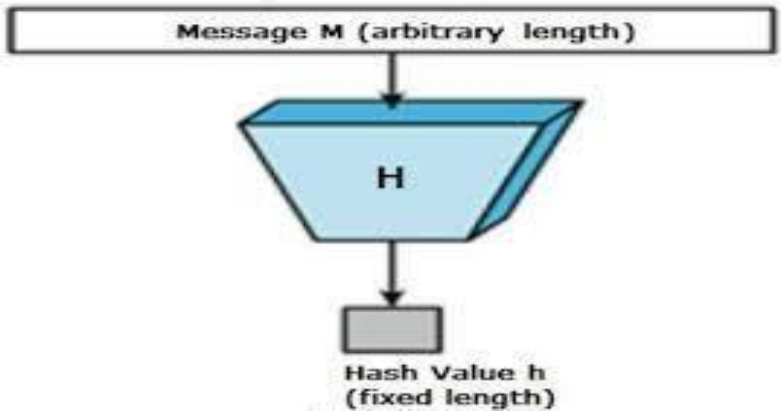
If we assume that only the receiver and the sender know the identity of the secret key, and if the receivedMAC matches the calculated MAC, then:

- The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC. Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.
- The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
- If the message includes a sequence number (such as is used with TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must be for decryption. In general, the MAC function is a many-to-one function. The domain of the function consists of messages of some arbitrary length, whereas the range consists of all possible MACs and all possible keys.

Cryptographic Hash Function:

- A cryptographic hash function (CHF) is a hash function that is suitable for use in cryptography.
- It is a mathematical algorithm that maps data of arbitrary size (often called the "message") to a bit string of a fixed size (the "hash value", "hash", or "message digest") and is a one-way function, that is, a function which is practically infeasible to invert.



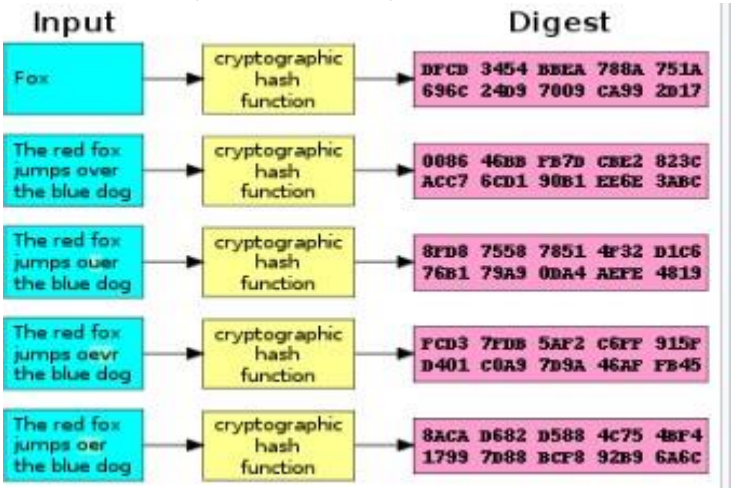
- Ideally, the only way to find a message that produces a given hash is to attempt a brute-force search of possible inputs to see if they produce a match, or use a table of matched hashes.
- Cryptographic hash functions are a basic tool of modern cryptography.

Figure: Cryptographic Hash Function; h = H(M)

Properties of Hash Function:

The ideal cryptographic hash function has the following main properties:

- it is deterministic, meaning that the same message always results in the same hash
- it is quick to compute the hash value for any given message
- it is infeasible to generate a message that yields a given hash value (Pre-Image Resistant)
- it is infeasible to find two different messages with the same hash value (Collision Resistance)
- A small change to a message should change the hash value so extensively that the new hash valueappears uncorrelated with the old hash value (avalanche effect)



Applications of Hash Function:

Given that the hash depends on the input to the hash function and will change with the input hash functions are used:

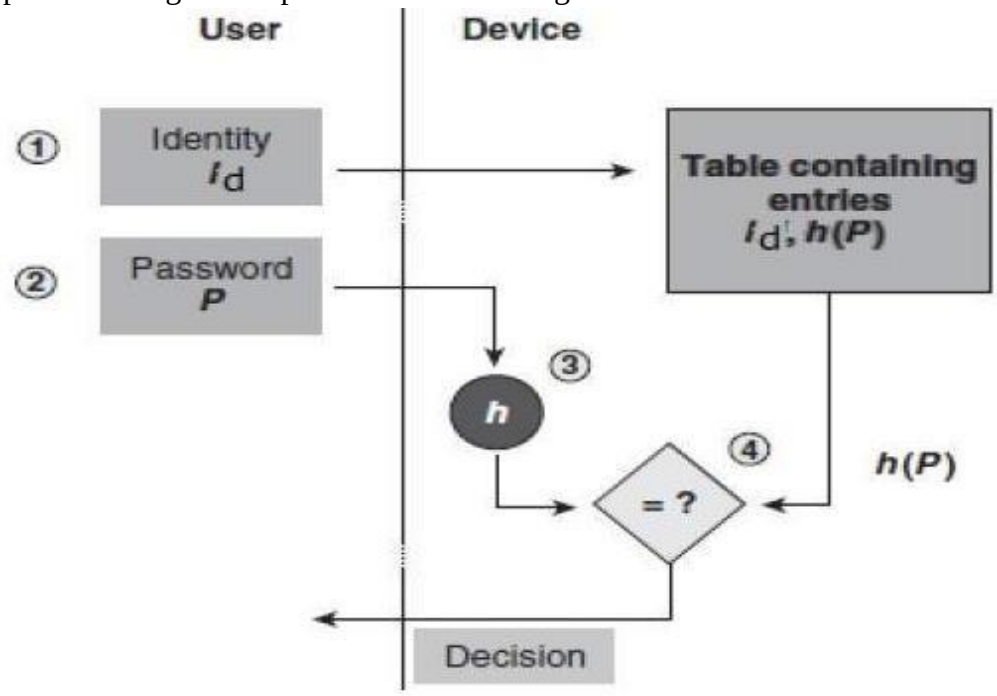
- to ensure that messages have not been tampered with (message authentication, digital signatures, checksums) and
- to check for equality while preserving secrecy / efficiently (for example, checking if password is correct without storing the actual password, or checking for duplicates in lists of large items).
- They are also used as proof-of-work (for example, in cryptocurrencies like bitcoin), error- correcting codes, and randomization and to make cryptographic algorithms more efficient.

There are two direct applications of hash function based on its cryptographic properties.

Password Storage

Hash functions provide protection to password storage.

- Instead of storing password in clear, mostly all logon processes store the hash values of passwords in the file.
- The Password file consists of a table of pairs which are in the form (user id, h(P)).
- The process of logon is depicted in the following illustration:

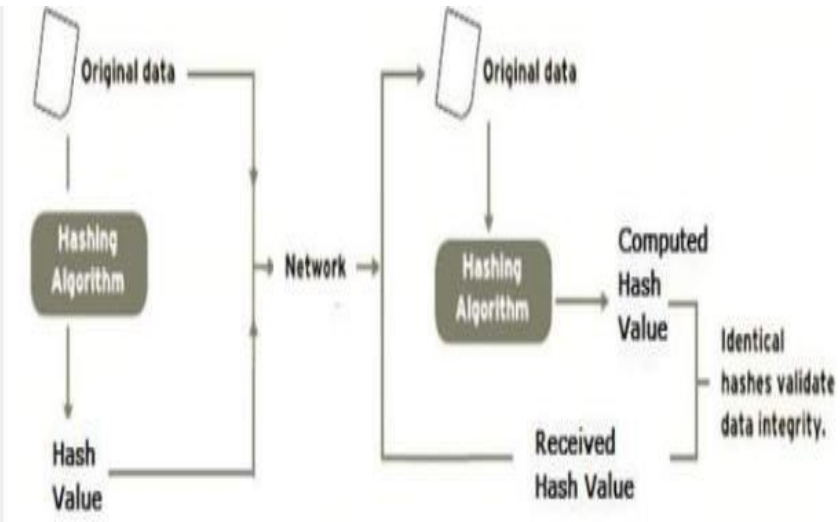


- An intruder can only see the hashes of passwords, even if he accessed the password. He can neither logon using hash nor can he derive the password from hash value since hash function possesses the property of pre-image resistance.

Data Integrity Check

Data integrity check is a most common application of the hash functions. It is used to generate the checksums on data files. This application provides assurance to the user about correctness ofthe data.

The process is depicted in the following illustration:



The integrity check helps the user to detect any changes made to original file. It however, does not provide any assurance about originality. The attacker, instead of modifying file data, can change the entire file and compute all together new hash and send to the receiver. This integrity check application is useful only if the user is sure about the originality of file.

➢ Hash algorithms

A cryptographic hash function is an algorithm that takes an arbitrary amount of data input—a credential—and produces a fixed-size output of enciphered text called a hash value, or just "hash." That enciphered text can then be stored instead of the password itself, and later used to verify the user.
Certain properties of cryptographic hash functions impact the security of password storage.

- **Non-reversibility, or one-way function.** A good hash should make it very hard to reconstruct the original password from the output or hash.
- **Diffusion, or avalanche effect**. A change in just one bit of the original password should result in change to half the bits of its hash. In other words, when a password is changed slightly, the output of enciphered text should change significantly and unpredictably.
- **Determinism.** A given password must always generate the same hash value or enciphered text.
- **Collision resistance.** It should be hard to find two different passwords that hash to the same enciphered text.
- **Non-predictable.** The hash value should not be predictable from the password.

There are variations that can improve your hash function and provide a greater barrier against attacks.

Salted hashes

Salting adds random data to each plaintext credential. The result: two identical plaintext passwords are now differentiated in enciphered text form so that duplicates cannot be detected.

Keyed hash functions

A keyed hash function (also known as a hash message authentication code, or HMAC) is an algorithm that uses a cryptographic key AND a cryptographic hash function to produce a message authentication code that is keyed and hashed.

Adaptive hash functions

An adaptive one-way function is any function that is designed to iterate on its inner workings, feeding the output back as input, in a manner that causes it to–ultimately–take longer to execute. It is adaptive because the developer can adjust how many iterations occur. To protect stored passwords, architects have applied the adaptive design to hash functions (such as PBKDF2) and to encryption schemes.

The trade-off of cryptographic hash functions

Cryptographic hash functions do provide barriers to attackers, like speed bumps slowing down a speeding motorcycle. But it's critical to remember that eventually the motorcycle will still make it down the street. However, these barriers will slow down your defenders as well—normal users and your server. Set the speed bump too high, and you run the risk of annoying your user—and overtaxing your server.

➢ Message Authentication Code (MAC)

MAC algorithm is a symmetric key cryptographic technique to provide message authentication. For establishing MAC process, the sender and receiver share a symmetric key K.

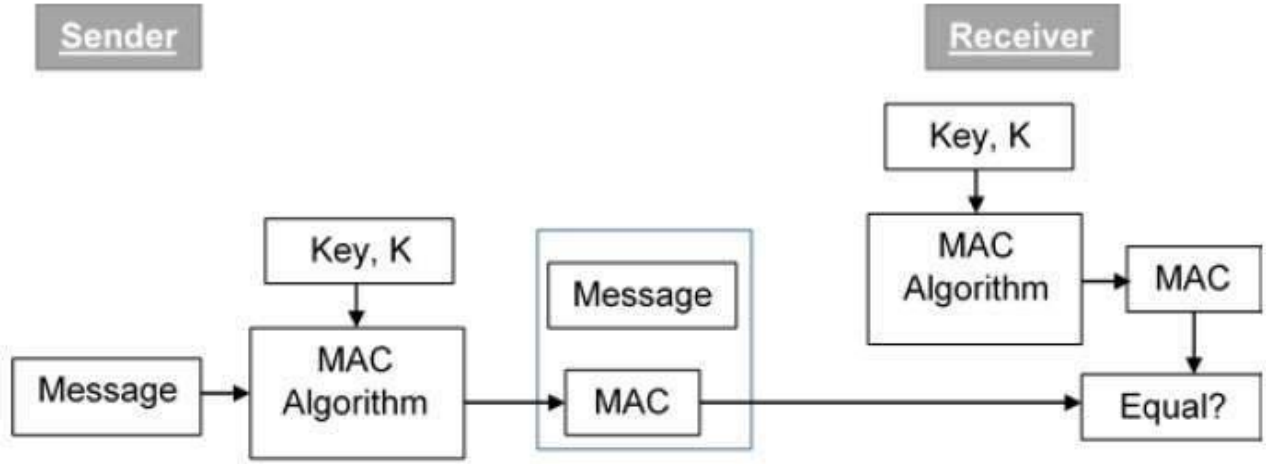
Essentially, a MAC is an encrypted checksum generated on the underlying message that is sent along with a message to ensure message authentication.

The process of using MAC for authentication is depicted in the following illustration –

Let us now try to understand the entire process in detail −

- The sender uses some publicly known MAC algorithm, inputs the message and the secret key K and produces a MAC value.
- Similar to hash, MAC function also compresses an arbitrary long input into a fixed length output. The major difference between hash and MAC is that MAC uses secret key during the compression.

- The sender forwards the message along with the MAC. Here, we assume that the message is



sent in the clear, as we are concerned of providing message origin authentication, not confidentiality. If confidentiality is required then the message needs encryption.

- On receipt of the message and the MAC, the receiver feeds the received message and the shared secret key K into the MAC algorithm and re-computes the MAC value.
- The receiver now checks equality of freshly computed MAC with the MAC received from the sender. If they match, then the receiver accepts the message and assures himself that the message has been sent by the intended sender.
- If the computed MAC does not match the MAC sent by the sender, the receiver cannot determine whether it is the message that has been altered or it is the origin that has been falsified. As a bottom-line, a receiver safely assumes that the message is not the genuine.

Limitations of MAC

There are two major limitations of MAC, both due to its symmetric nature of operation −

- **Establishment of Shared Secret.**
  - It can provide message authentication among pre-decided legitimate users who have shared key.
  - This requires establishment of shared secret prior to use of MAC.
- **Inability to Provide Non-Repudiation**
  - Non-repudiation is the assurance that a message originator cannot deny any previously sent messages and commitments or actions.
  - MAC technique does not provide a non-repudiation service. If the sender and receiver get involved in a dispute over message origination, MACs cannot provide a proof that a message was indeed sent by the sender.
  - Though no third party can compute the MAC, still sender could deny having sent the message and claim that the receiver forged it, as it is impossible to determine which of the two parties computed the MAC.

MACs based on hash functions: HMAC

HMAC Design Objectives

RFC 2104 lists the following design objectives for HMAC.

- To use, without modifications, available hash functions. In particular, to use hash functions that performwell in software and for which code is freely and widely available.
- To allow for easy replaceability of the embedded hash function in case faster or more

secure hashfunctions are found or required.

- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism basedon reasonable assumptions about the embedded hash function.

HMAC Algorithm

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)

IV = initial value input to hash function

M = message input to HMAC (including the padding specified in the embeddedhash function)

Yi _ *i* th block of *M*, 0 ≤*i* ≤(*L* – 1)

L _ number of blocks in *M*

b _ number of bits in a block

n _ length of hash code produced by embedded hash function

K _ secret key; recommended length is ≥*n*; if key length is greater than *b*, the key is input to the hash functionto produce an *n*-bit key

K+ _ *K* padded with zeros on the left so that the result is *b* bits in lengthipad _ 00110110 (36 in hexadecimal) repeated *b*/8 times

opad _ 01011100 (5C in hexadecimal) repeated *b*/8 times

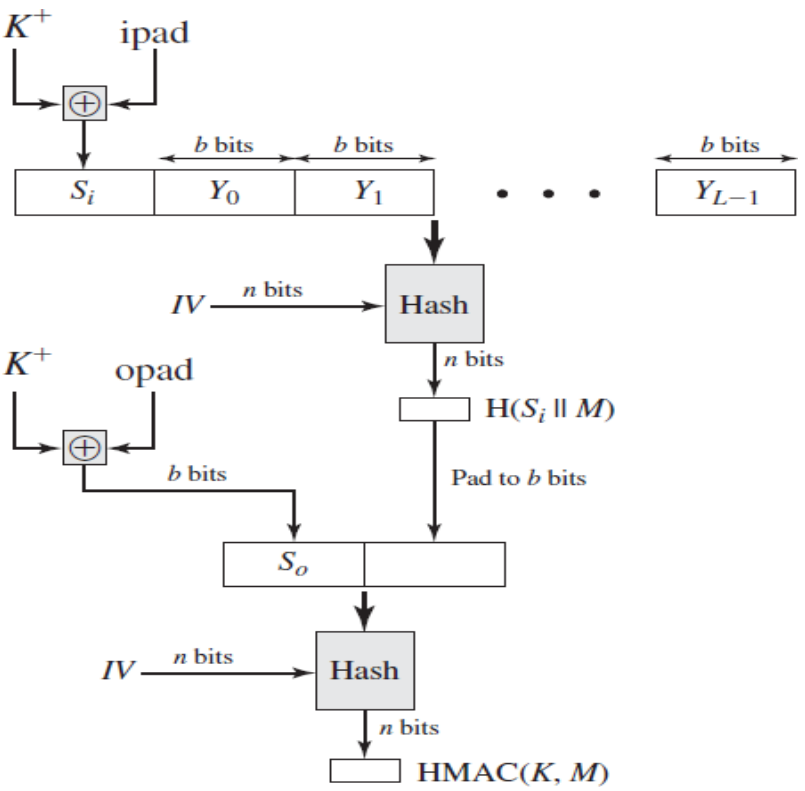


fig: HMAC Structure

HMAC can be expressed as: HMAC(*K*, *M*) = H[(*K*+ ⊕ opad) || H[(*K*+ ⊕ ipad) || *M*]]

The algorithm is as follows:

1. Append zeros to the left end of K to create a b-bit string K^+ (e.g., if K is of length 160 bits and , then will be appended with 44 zeroes).

2. XOR (bitwise exclusive-OR) with ipad to produce the b-bit block Si.

3. Append M to Si.

4. Apply H to the stream generated in step 3.

5. XOR K^+ with opad to produce the b-bit block S_O

6. Append the hash result from step 4 to S_O.

7. Apply H to the stream generated in step 6 and output the result.

- RSA Encryption Algorithm

RSA encryption algorithm is a type of public-key encryption algorithm. To better understand RSA, lets first understand what is public-key encryption algorithm.

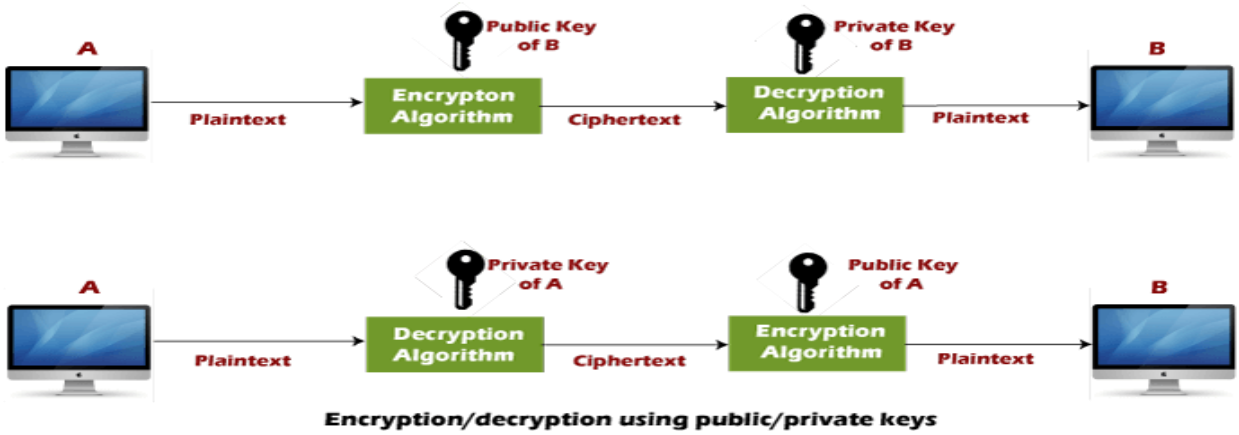
Public key encryption algorithm:

Public Key encryption algorithm is also called the Asymmetric algorithm. Asymmetric algorithms are those algorithms in which sender and receiver use different keys for encryption and decryption. Each sender is assigned a pair of keys:

- **Public key**
- **Private key**

The **Public key** is used for encryption, and the **Private Key** is used for decryption. Decryption cannot be done using a public key. The two keys are linked, but the private key cannot be derived from the public key. The public key is well known, but the private key is secret and it is known only to the user who owns the key. It means that everybody can send a message to the user using user's public key. But only the user can decrypt the message using his private key.

The Public key algorithm operates in the following manner:

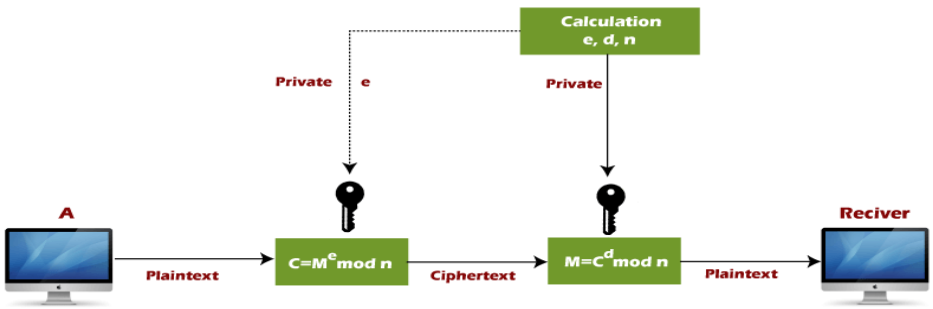


Encryption/decryption using public/private keys

- The data to be sent is encrypted by sender **A** using the public key of the intended receiver
- B decrypts the received ciphertext using its private key, which is known only to B. B replies to A encrypting its message using A's public key.
- A decrypts the received ciphertext using its private key, which is known only to him.

RSA encryption algorithm:

RSA is the most common public-key algorithm, named after its inventors **Rivest, Shamir, and Adelman (RSA).**



RSA

RSA algorithm uses the following procedure to generate public and private keys:

- Select two large prime numbers, p and **q**.

- Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.
- Choose a number **e** less than **n**, such that n is relatively prime to **(p - 1) x (q -1).** It means that **e** and **(p - 1) x (q - 1)** have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n), **gcd (e,d(n)) =1**
- If **n = p x q,** then the public key is <e, n>. A plaintext message **m** is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C. **$C = m^e \bmod n$**
- Here**, m** must be less than **n**. A larger message (>n) is treated as a concatenation of messages, each of which is encrypted separately.
- To determine the private key, we use the following formula to calculate the d such that: **$D_e \bmod \{(p - 1) \times (q - 1)\} = 1$**
- **Or**
  $D_e \bmod \varphi (n) = 1$
- The private key is <d, n>. A ciphertext message **c** is decrypted using private key <d, n>. To calculate plain text **m** from the ciphertext c following formula is used to get plain text m. **$m = c^d \bmod n$**

Let's take some example of RSA encryption algorithm:

Example 1:

This example shows how we can encrypt plaintext 9 using the RSA public-key encryption algorithm. This example uses prime numbers 7 and 11 to generate the public and private keys.

Explanation:

Step 1: Select two large prime numbers, p, and **q**.

p = 7

q = 11

Step 2: Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.

First, we calculate

n = p x q

n = 7 x 11

n = 77

Step 3: Choose a number **e** less that **n**, such that n is relatively prime to **(p - 1) x (q -1).** It means that **e** and **(p - 1) x (q - 1)** have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n), gcd (e, d (n)) =1.

Second, we calculate

φ (n) = (p - 1) x (q-1)

φ (n) = (7 - 1) x (11 - 1)

φ (n) = 6 x 10

φ (n) = 60

Let us now choose relative prime e of 60 as 7.

Thus the public key is <e, n> = (7, 77)

Step 4: A plaintext message **m** is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C.

To find ciphertext from the plain text following formula is used to get ciphertext C.

$C = m^e \bmod n$

$C = 9^7 \bmod 77$

C = 37

Step 5: The private key is <d, n>. To determine the private key, we use the following formula d such that:

D_e mod {(p - 1) x (q - 1)} = 1

7d mod 60 = 1, which gives d = 43

The private key is <d, n> = (43, 77)

Step 6: A ciphertext message **c** is decrypted using private key <d, n>. To calculate plain text **m** from the ciphertext c following formula is used to get plain text m.

$m = c^d$ mod n

$m = 37^{43}$ mod 77

m = 9

In this example, Plain text = 9 and the ciphertext = 37