

MODULE 2

CHAPTER 1 PUBLIC-KEY CRYPTOGRAPHY AND RSA

- 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.

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.

Terminology Related to Asymmetric Encryption

Asymmetric 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 and signature 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 **total 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 to cryptography.

Public-Key Cryptosystems

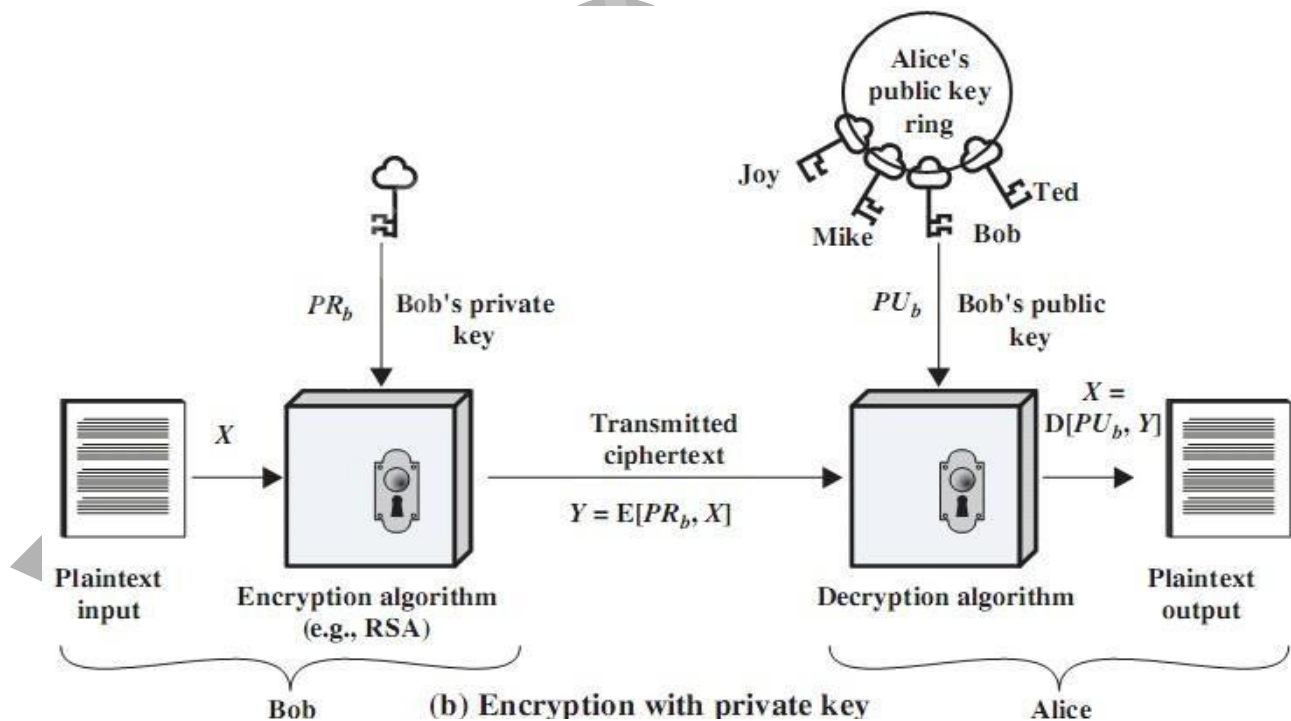
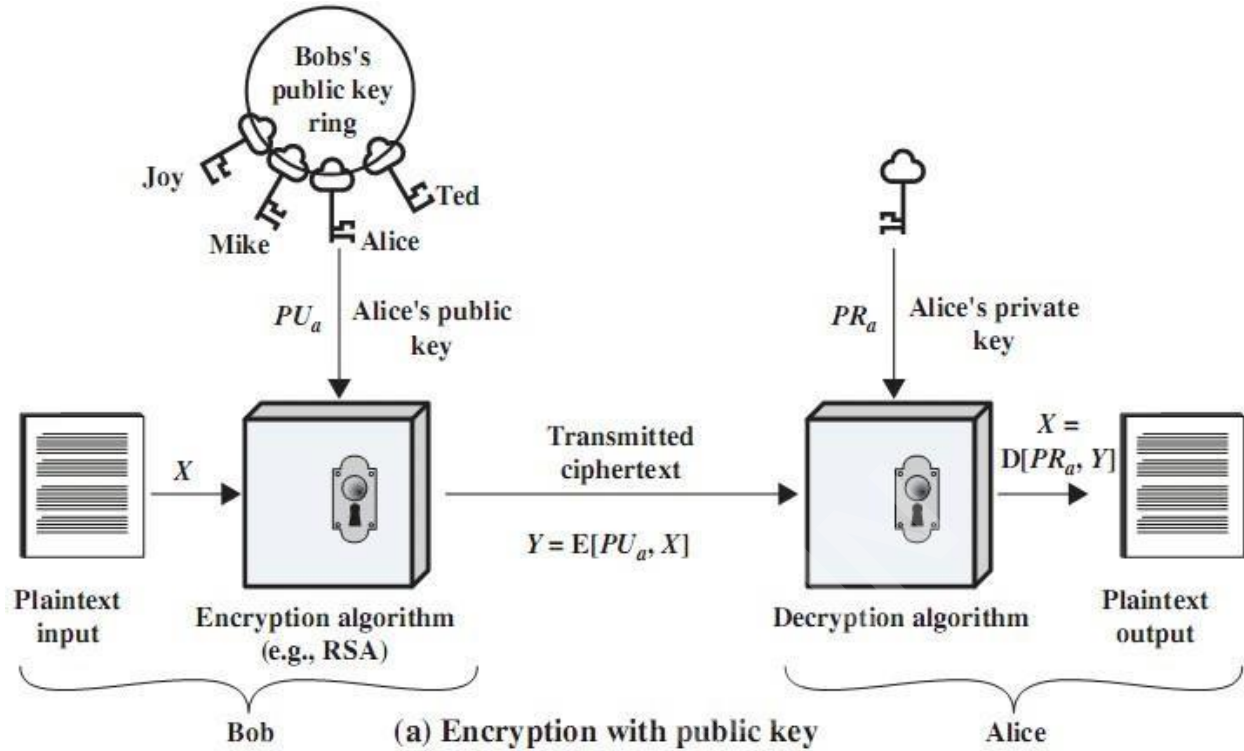
Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristic:

- 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 for decryption.

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 as input.
- **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 if one 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 the plaintext 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.



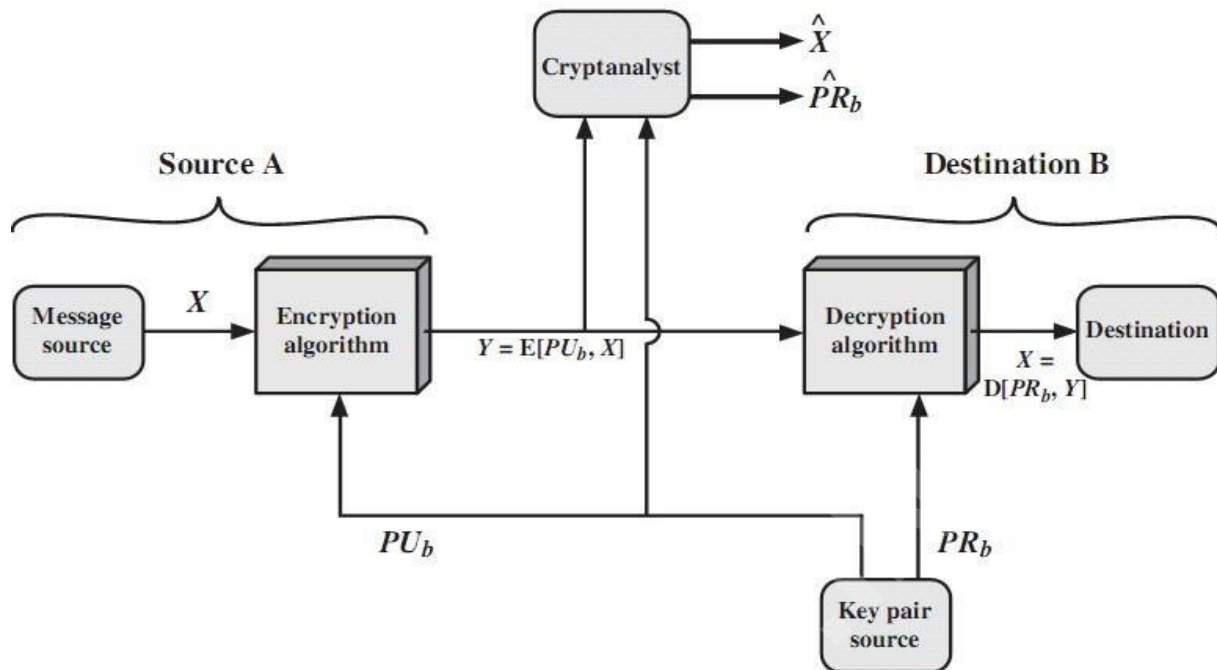
The essential steps are the following:

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

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. 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 the algorithm and the key.	1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the 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 impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys 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, PU_b , and a private key, PR_b . PR_b is known only to B, whereas PU_b is publicly available and therefore accessible by A.
- With the message X and the encryption key PU_b as input, A forms the ciphertext

$$Y = E(PU_b, X)$$

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

$$Y = E(PR_a, X); X = D(PU_a, Y)$$

- To provide both the authentication function and confidentiality have a double use of the public-key scheme (as shown here):

$$Z = E(PU_b, E(PR_a, X))$$

$$X = D(PU_a, D(PR_b, Z))$$

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's public 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 , private key 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.

$$Y = f_k(X) \text{ easy, if } k \text{ and } X \text{ are known}$$

$$X = f_k^{-1}(Y) \text{ easy, if } k \text{ and } Y \text{ are known}$$

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

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.

The RSA Algorithm

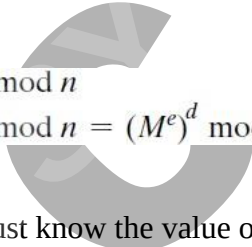
RSA Algorithm was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT.

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 RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and $n - 1$ for some n . A typical size for n is 1024 bits, or 309 decimal digits. That is, n is less than 2^{1024} .

Description of the Algorithm:

RSA makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number n . That is, the blocksize must be less than or equal to $\log_2(n) + 1$. Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C .


$$C = M^e \bmod n$$
$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Both sender and receiver must know the value of n . The sender knows the value of e , and only the receiver knows the value of d . Thus, this is a public-key encryption algorithm with a public key of $PU = \{e, n\}$ and a private key of $PR = \{d, n\}$.

For this algorithm to be satisfactory for public-key encryption, the following requirements must be met.

1. It is possible to find values of e, d, n such that $M^{ed} \bmod n = M$ for all $M < n$.
2. It is relatively easy to calculate $M^e \bmod n$ and $C^d \bmod n$ for all values of $M < n$.
3. It is infeasible to determine d given e and n .

For now, we focus on the first requirement and consider the other questions later. We need to find a relationship of the form

$$M^{ed} \bmod n = M$$

The preceding relationship holds if e and d are multiplicative inverses modulo $\phi(n)$, where $\phi(n)$ is the Euler totient function. The relationship between e and d can be expressed as $ed \equiv 1 \pmod{\phi(n)}$

This is equivalent to saying

$$\begin{aligned} ed &\equiv 1 \pmod{\phi(n)} \\ d &\equiv e^{-1} \pmod{\phi(n)} \end{aligned}$$

That is, e and d are multiplicative inverses mod $\phi(n)$. Note that, according to the rules of modular arithmetic, this is true only if d (and therefore e) is relatively prime to $\phi(n)$. Equivalently, $\gcd(\phi(n), d) = 1$.

We are now ready to state the RSA scheme. The ingredients are the following:

p, q , two prime numbers	(private, chosen)
$n = pq$	(public, calculated)
e , with $\gcd(\phi(n), e) = 1$; $1 < e < \phi(n)$	(public, chosen)
$d \equiv e^{-1} \pmod{\phi(n)}$	(private, calculated)

The private key consists of $\{d, n\}$ and the public key consists of $\{e, n\}$. Suppose that user A has published its public key and that user B wishes to send the message M to A. Then B calculates $C = M^e \pmod{n}$ and transmits C . On receipt of this ciphertext, user A decrypts by calculating $M = C^d \pmod{n}$.

Example:

Alice generates a public/private key pair; Bob encrypts using Alice's public key; and Alice decrypts using her private key. For this example, the keys were generated as follows.

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 * 11 = 187$.
3. Calculate $\phi(n) = (p - 1)(q - 1) = 16 * 10 = 160$.
4. Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.
5. Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$, because $23 * 7 \equiv 1 \pmod{160}$.

* $7 = 161 = (1 * 160) + 1$; d can be calculated using the extended Euclid's algorithm

The resulting keys are public key $PU = \{7, 187\}$ and private key $PR = \{23, 187\}$.

The example shows the use of these keys for a plaintext input of $M = 88$. For encryption, we need to calculate $C = 88^7 \bmod 187$. Exploiting the properties of modular arithmetic, we can do this as follows.

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$

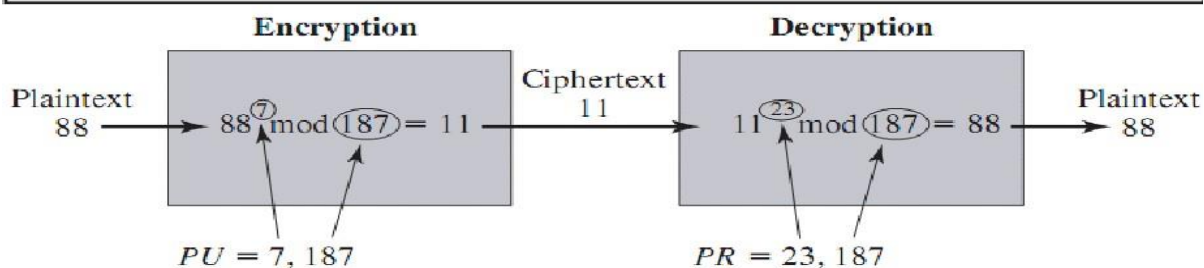
$$88^1 \bmod 187 = 88$$

$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$

$$88^4 \bmod 187 = 59,969,536 \bmod 187 = 132$$

$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894,432 \bmod 187 = 11$$

Key Generation Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$
Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$
Decryption by Alice with Alice's Public Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$



For decryption, we calculate $M = 11^{23} \bmod 187$:

$$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$$

$$11^1 \bmod 187 = 11$$

$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14,641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214,358,881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 79,720,245 \bmod 187 = 88$$

Computational Aspects

We now turn to the issue of the complexity of the computation required to use RSA. There are actually two issues to consider: **encryption/decryption** and **key generation**.

To perform the modular exponentiations, you can use the “**Square and Multiply Algorithm**”, a fast, efficient algorithm for doing exponentiation.

The idea is to repeatedly square the base, and multiply in the ones that are needed to compute the result, as found by examining the binary representation of the exponent.

- Eg. $7^5 = 7^4 \cdot 7^1 = 3 \cdot 7 = 10 \bmod 11$
- eg. $3^{129} = 3^{128} \cdot 3^1 = 5 \cdot 3 = 4 \bmod 11$

The RSA algorithm requires that during key generation, the user selects a value e that is relatively prime to $\phi(n)$. Thus, if a value e is selected first, and the primes p and q are generated, it may turn out that $\gcd(\phi(n), e) \neq 1$. In that case, the user must reject the p, q values and generate a new p, q pair.

RSA Key Generation

- Before the application of the public-key cryptosystem, each participant must generate a pair of keys, which requires finding primes and computing inverses.
- Both the prime generation and the derivation of a suitable pair of inverse exponents may involve trying a number of alternatives.

- Typically make random guesses for a possible p or q , and check using a probabilistic primality test whether the guessed number is indeed prime. If not, try again.
- Note that the prime number theorem shows that the average number of guesses needed is not too large.
- Then compute decryption exponent d using Euclid's Inverse Algorithm, which is quite efficient.

The Security of RSA

Four possible approaches to attacking the RSA algorithm are as follows:

- **Brute force:** This involves trying all possible private keys.
- **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks:** These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

Brute force:

The defense against the brute-force approach is the same for RSA as for other cryptosystems, namely, **use a large key space**. Thus, the larger the number of bits in d , the better. However, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run.

The Factoring Problem:

We can identify three approaches to attacking RSA mathematically:

- Factor n into its two prime factors. This enables calculation of $\phi(n) = (p-1)(q-1)$, which, in turn, enables determination of $d = e^{-1} \bmod \phi(n)$.
- Determine $\phi(n)$ directly, without first determining p and q . Again, this enables determination of $d = e^{-1} \bmod \phi(n)$.
- Determine d directly, without first determining $\phi(n)$.

Determining $\phi(n)$ given n is equivalent to factoring n . With presently known algorithms, determining d given e and n appears to be at least as time-consuming as the

factoring problem. Hence, we can use factoring performance as a benchmark against which to evaluate the security of RSA.

To avoid values of n that may be factored more easily, the algorithm's inventors suggest the following constraints on p and q :

1. p and q should differ in length by only a few digits. Thus, for a 1024-bit key (309 decimal digits), both p and q should be on order of 10^{75} to 10^{100} .
2. Both $(p - 1)$ and $(q - 1)$ should contain a large prime factor
3. $\gcd(p-1, q-1)$ should be small.

Timing Attacks:

- Timing Attacks demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages.
- Timing Attacks are based on observing how long it takes to compute the cryptographic operations. Timing attacks are applicable not just to RSA, but to other public-key cryptography systems.
- This attack is alarming for two reasons: It comes from a completely unexpected direction and it is a ciphertext-only attack.

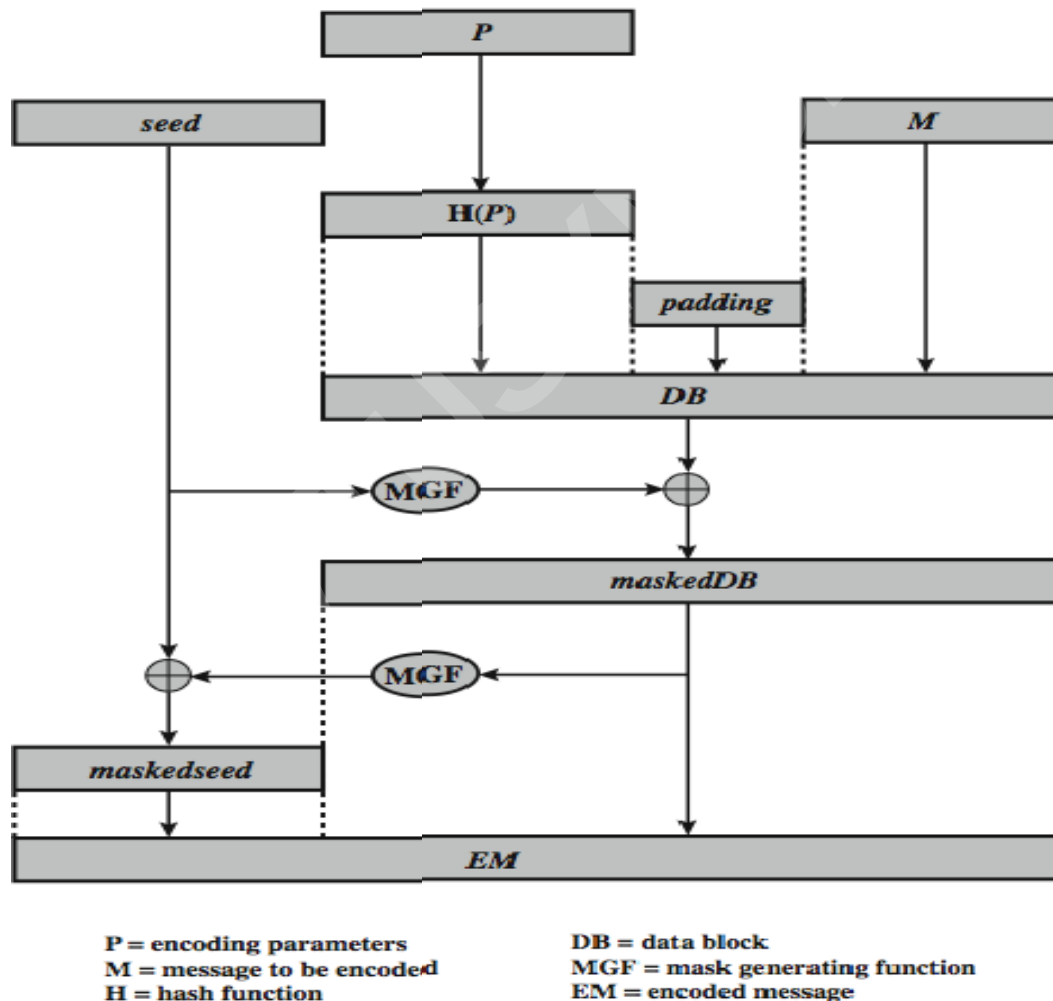
The timing attack is a serious threat; there are simple countermeasures that can be used, including using constant exponentiation time algorithms, adding random delays, or using blind values in calculations.

- **Constant exponentiation time:** Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.
- **Random delay:** Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack. Kocher points out that if defenders don't add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays.
- **Blinding:** Multiply the ciphertext by a random number before performing exponentiation. This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack.

Chosen Ciphertext Attacks:

- The RSA algorithm is vulnerable to a chosen ciphertext attack (CCA).
- CCA is defined as an attack in which adversary chooses a number of ciphertexts and is then given the corresponding plaintexts, decrypted with the target's private key.
- The adversary exploits properties of RSA and selects blocks of data that, when processed using the target's private key, yield information needed for cryptanalysis.
- More sophisticated variants need to modify the plaintext using a procedure known as optimal asymmetric encryption padding (OAEP).

To counter such attacks RSA Security, a leading RSA vendor and former holder of the RSA patent, recommends modifying the plaintext using a procedure known as optimal asymmetric encryption padding (OAEP).



- As a first step the message M to be encrypted is padded. A set of optional parameters P is passed through a hash function H .

- The output is then padded with zeros to get the desired length in the overall data block(DB). Next, a random seed is generated and passed through another hash function, called the mask generating function (MGF).
- The resulting hash value is bit-by-bit XORed with DB to produce a maskedDB.
- The maskedDB is in turn passed through the MGF to form a hash that is XORed with the seed to produce the masked seed.
- The concatenation of the maskedseed and the maskedDB forms the encoded messageEM.
- Note that the EM includes the padded message, masked by the seed, and the seed, masked by the maskedDB. The EM is then encrypted using RSA.

CHAPTER 2. OTHER PUBLIC KEY CRYPTOSYSTEMS

Diffie-Hellman Key Exchange

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages.

If a is a "primitive root" of the prime number p , then the numbers

$$a \bmod p, a^2 \bmod p, \dots, a^{p-1} \bmod p$$

are distinct and consist of the integers from 1 through $p-1$ in some permutation.

Example:

3 is primitive root of 7.

The number 3 is a primitive root modulo 7, because

$$3^1 = 3 = 3^0 \times 3 \equiv 3 \equiv 3 \pmod{7}$$

$$3^2 = 9 = 3^1 \times 3 \equiv 9 \equiv 2 \pmod{7}$$

$$3^3 = 27 = 3^2 \times 3 \equiv 6 \equiv 6 \pmod{7}$$

$$3^4 = 81 = 3^3 \times 3 \equiv 18 \equiv 4 \pmod{7}$$

$$3^5 = 243 = 3^4 \times 3 \equiv 12 \equiv 5 \pmod{7}$$

$$3^6 = 729 = 3^5 \times 3 \equiv 15 \equiv 1 \pmod{7}$$

Here we see that the period of 3^k modulo 7 is 6. The remainders in the period, which are 3, 2, 6, 4, 5, 1, form a rearrangement of all nonzero remainders modulo 7, implying that 3 is indeed a primitive root modulo 7.

The Algorithm:

There are two publicly known numbers: a prime number q and an integer that is a primitive root of q .

Suppose the users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{X_A} \bmod q$. Similarly, user B independently selects a random integer $X_B < q$ and computes $Y_B = \alpha^{X_B} \bmod q$.

Each side keeps the X value private and makes the Y value available publicly to the otherside.

User A computes the key as $K = (Y_B)^{X_A} \bmod q$ and user B computes the key as $K = (Y_A)^{X_B} \bmod q$.
These two calculations produce identical results:

$$\begin{aligned} K &= (Y_B)^{X_A} \bmod q \\ &= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\ &= (\alpha^{X_B})^{X_A} \bmod q && \text{by the rules of modular arithmetic} \\ &= (\alpha^{X_B X_A} \bmod q \\ &= (\alpha^{X_A})^{X_B} \bmod q \\ &= (\alpha^{X_A} \bmod q) \\ &= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\ &= (Y_A)^{X_B} \bmod q \end{aligned}$$

Example:

Prime number $q = 353$ and a primitive root of 353, in this case $\alpha = 3$.

A and B select secret keys $X_A = 97$ and $X_B = 233$, respectively. Each computes its publickey:

A computes $Y_A = 3^{97} \bmod 353 = 40$.

B computes $Y_B = 3^{233} \bmod 353 = 248$.

After they exchange public keys, each can compute the common secret key:

A computes $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$.

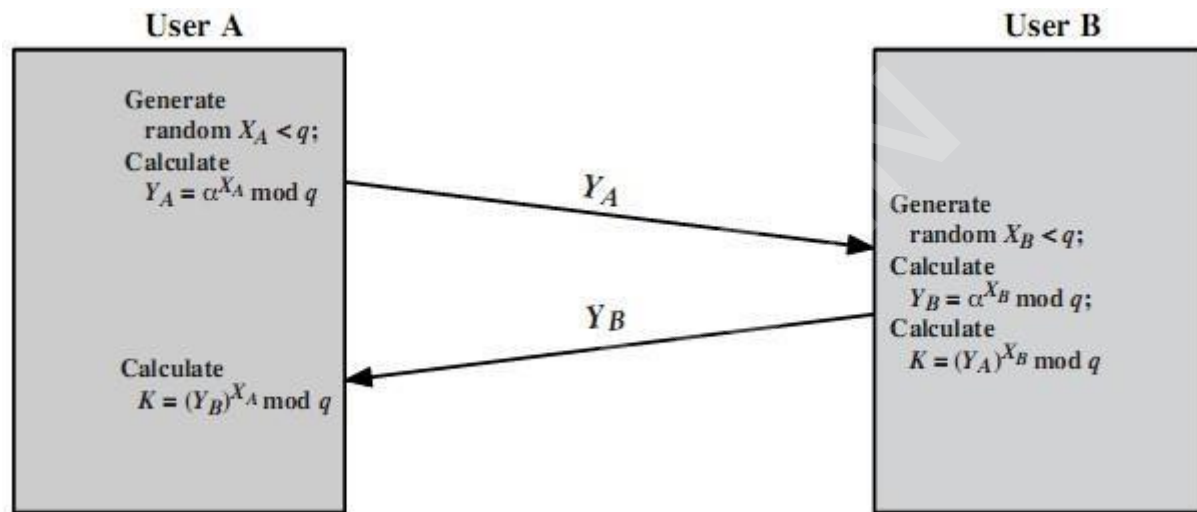
B computes $K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353$.

Key Exchange Protocols:

Suppose that user A wishes to set up a connection with user B and use a secret key to encrypt messages on that connection. User A can generate a one-time private key X_A , calculate Y_A , and send that to user B.

User B responds by generating a private value X_B calculating Y_B , and sending Y_B to user A. Both users can now calculate the key.

The necessary public values q and α would need to be known ahead of time. Alternatively, user A could pick values for q and α and include those in the first message.

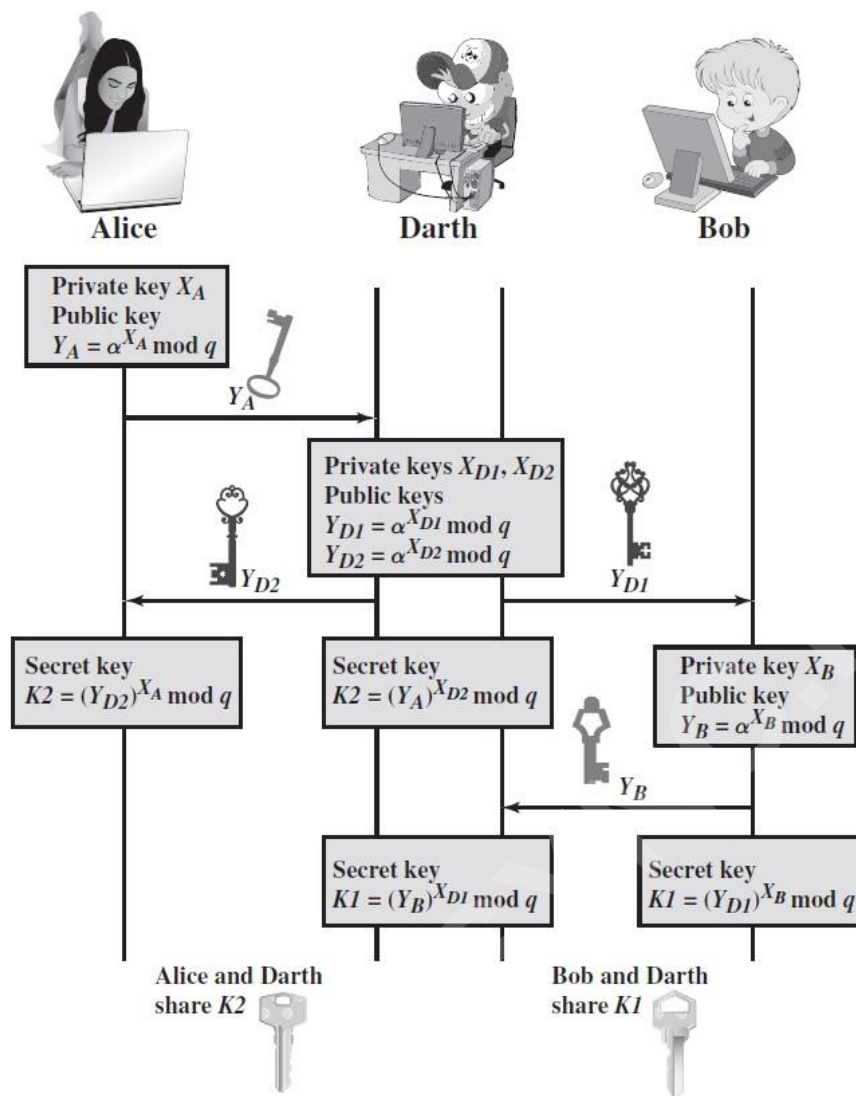


Man-in-the-Middle Attack:

Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows.

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2} .
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates $K2 = (Y_A)^{X_{D2}} \bmod q$.
4. Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_B} \bmod q$.
5. Bob transmits Y_B to Alice.
6. Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates $K1 = (Y_B)^{X_{D1}} \bmod q$.
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$.

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key $K1$ and Alice and Darth share secret key $K2$. All future communication between Bob and Alice is compromised in the following way.



The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates;

ELGAMAL CRYPTOGRAPHIC SYSTEM

As with Diffie-Hellman, the global elements of ElGamal are a prime number q and α , which is a primitive root of q . User A generates a private/public key pair as follows:

1. Generate a random integer X_A , such that $1 < X_A < q-1$
2. Compute $Y_A = \alpha^{X_A} \bmod q$
3. A's private key is X_A ; A's public key is $\{q, \alpha, Y_A\}$

Any user B that has access to A's public key can encrypt a message as follows:

1. Represent the message as an integer M in the range $0 \leq M \leq q - 1$. Longer messages are sent as a sequence of blocks, with each block being an integer less than q .
2. Choose a random integer k such that $1 \leq k \leq q - 1$.
3. Compute a one-time key $K = (Y_A)^k \bmod q$.
4. Encrypt M as the pair of integers (C_1, C_2) where

$$C_1 = \alpha^k \bmod q; C_2 = KM \bmod q$$

User A recovers the plaintext as follows:

1. Recover the key by computing $K = (C_1)^{X_A} \bmod q$.
2. Compute $M = (C_2 K^{-1}) \bmod q$.

User A generates a public/private key pair; user B encrypts using A's public key; and user A decrypts using her private key.

First, how K is recovered by the decryption process:

$K = (Y_A)^k \bmod q$	K is defined during the encryption process
$K = (\alpha^{X_A} \bmod q)^k \bmod q$	substitute using $Y_A = \alpha^{X_A} \bmod q$
$K = \alpha^{kX_A} \bmod q$	by the rules of modular arithmetic
$K = (C_1)^{X_A} \bmod q$	substitute using $C_1 = \alpha^k \bmod q$

Next, using K , we recover the plaintext as

$$C_2 = KM \bmod q$$

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

Global Public Elements	
q	prime number
α	$\alpha < q$ and α a primitive root of q
Key Generation by Alice	
Select private X_A	$X_A < q - 1$
Calculate Y_A	$Y_A = \alpha^{X_A} \bmod q$
Public key	$PU = \{q, \alpha, Y_A\}$
Private key	X_A
Encryption by Bob with Alice's Public Key	
Plaintext:	$M < q$
Select random integer k	$k < q$
Calculate K	$K = (Y_A)^k \bmod q$
Calculate C_1	$C_1 = \alpha^k \bmod q$
Calculate C_2	$C_2 = KM \bmod q$
Ciphertext:	(C_1, C_2)
Decryption by Alice with Alice's Private Key	
Ciphertext:	(C_1, C_2)
Calculate K	$K = (C_1)^{X_A} \bmod q$
Plaintext:	$M = (C_2 K^{-1}) \bmod q$

Example:

Thus, K functions as a one-time key, used to encrypt and decrypt the message. For example, let us start with the prime field GF (19); that is, q 19. It has primitive roots {2, 3, 10, 13, 14, and 15}, as shown in Table 8.3. We choose $\alpha = 10$.

Encryption:

Alice generates a key pair as follows:

1. Alice chooses $X_A = 5$.
2. Then $Y_A = \alpha^{X_A} \bmod q = \alpha^5 \bmod 19 = 3$ (see Table 8.3).
3. Alice's private key is 5 and Alice's public key is $\{q, \alpha, Y_A\} = \{19, 10, 3\}$.

Suppose Bob wants to send the message with the value $M = 17$. Then,

1. Bob chooses $k = 6$.
2. Then $K = (Y_A)^k \bmod q = 3^6 \bmod 19 = 729 \bmod 19 = 7$.
3. So

$$C_1 = \alpha^k \bmod q = \alpha^6 \bmod 19 = 11$$

$$C_2 = KM \bmod q = 7 \times 17 \bmod 19 = 119 \bmod 19 = 5$$
4. Bob sends the ciphertext $(11, 5)$.

Decryption:

1. Alice calculates $K = (C_1)^{X_A} \bmod q = 11^5 \bmod 19 = 161051 \bmod 19 = 7$.
2. Then K^{-1} in $GF(19)$ is $7^{-1} \bmod 19 = 11$.
3. Finally, $M = (C_2 K^{-1}) \bmod q = 5 \times 11 \bmod 19 = 55 \bmod 19 = 17$.

If a message must be broken up into blocks and sent as a sequence of encrypted blocks, a unique value of “k” should be used for each block. If it is used for more than one block, knowledge of one block m_1 of the message enables the user to compute other blocks as follows. Let

$$\begin{aligned} C_{1,1} &= \alpha^k \bmod q; C_{2,1} = KM_1 \bmod q \\ C_{1,2} &= \alpha^k \bmod q; C_{2,2} = KM_2 \bmod q \end{aligned}$$

Then,

$$\frac{C_{2,1}}{C_{2,2}} = \frac{KM_1 \bmod q}{KM_2 \bmod q} = \frac{M_1 \bmod q}{M_2 \bmod q}$$

If M_1 is known, then M_2 is easily computed as

$$M_2 = (C_{2,1})^{-1} C_{2,2} M_1 \bmod q$$

The security of Elgamal is based on the difficulty of computing discrete logarithms. To recover A's private key, an adversary would have to compute $X_A = \text{dlog}_{\alpha,q}(Y_A)$.