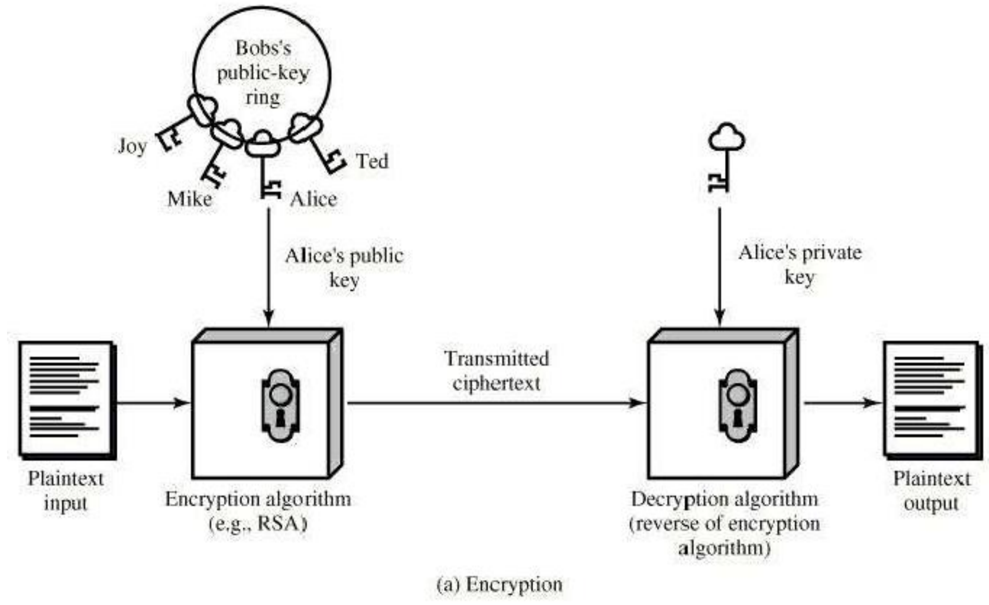
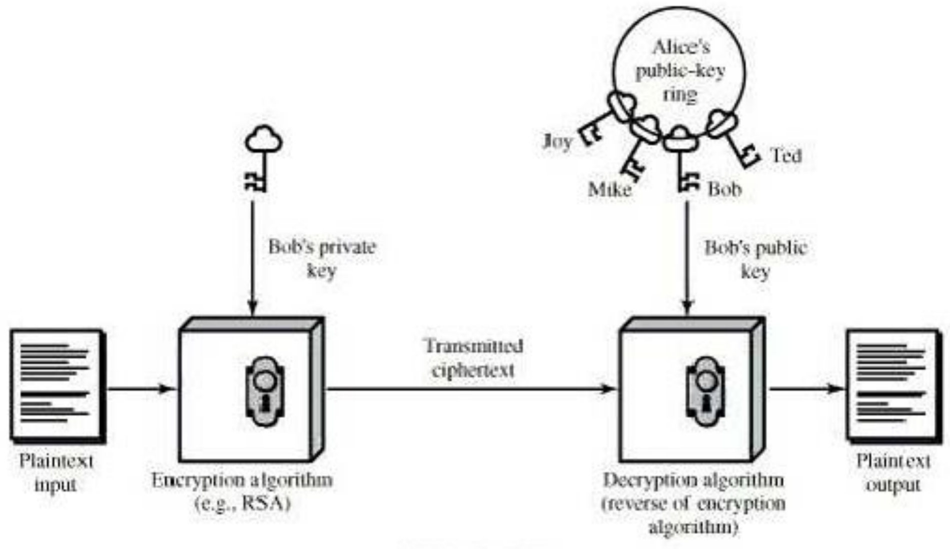
Define Public Key Cryptography

Public-key cryptography is a cryptographic system that uses two separate keys, one of which is secret and the other one is public. The algorithms used for public key cryptography are based on mathematical functions.

Public Key Cryptosystem



(a) Encryption



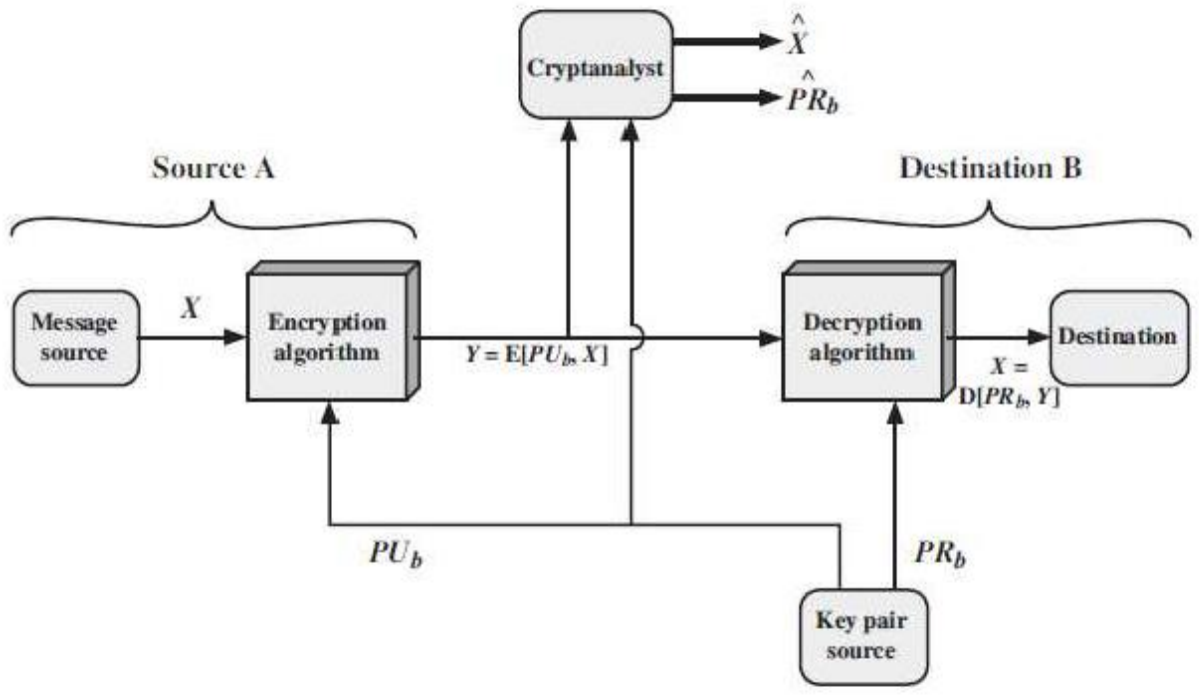
(b) Authentication

- A public-key encryption scheme has six parts.
  - **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.
  - **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key

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

- Any cryptosystem are designed to meet following goal
  1. Secrecy (Encryption)
  2. Authentication
- Now we will discuss how it is maintain in public key cryptosystem

Public Key Cryptosystem: Secrecy



Encryption using public key cryptography

- The essential steps are the following.
  - Each user generates a pair of keys to be used for the encryption and decryption of messages.
  - Each user places one of the two keys in a public register or other accessible file. This is the public key. The other key is kept private.
  - If A wishes to send a confidential message to B, A encrypts the message using B's public key.
  - When B receives the message, it decrypts it using the private key. No other recipient can decrypt the message because only B knows B's private key.
  - As long as a user's private key remains protected and secret, incoming communication is secure.
  - At any time, a system can change its private key and publish the companion public key to replace its old public key.
- Suppose there is some source A that produces a message in plaintext, $X = [X_1, X_2, \ldots, X_M]$ and sends it to B.
- B generates a related pair of keys: a public key, PU_b, and a private key, PR_b. $PU_b$is publicly available and therefore accessible by A.
- With the message X and the encryption key PU_b as input, A forms the ciphertext $Y = [Y_1, Y_2, \ldots, Y_N]$:

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

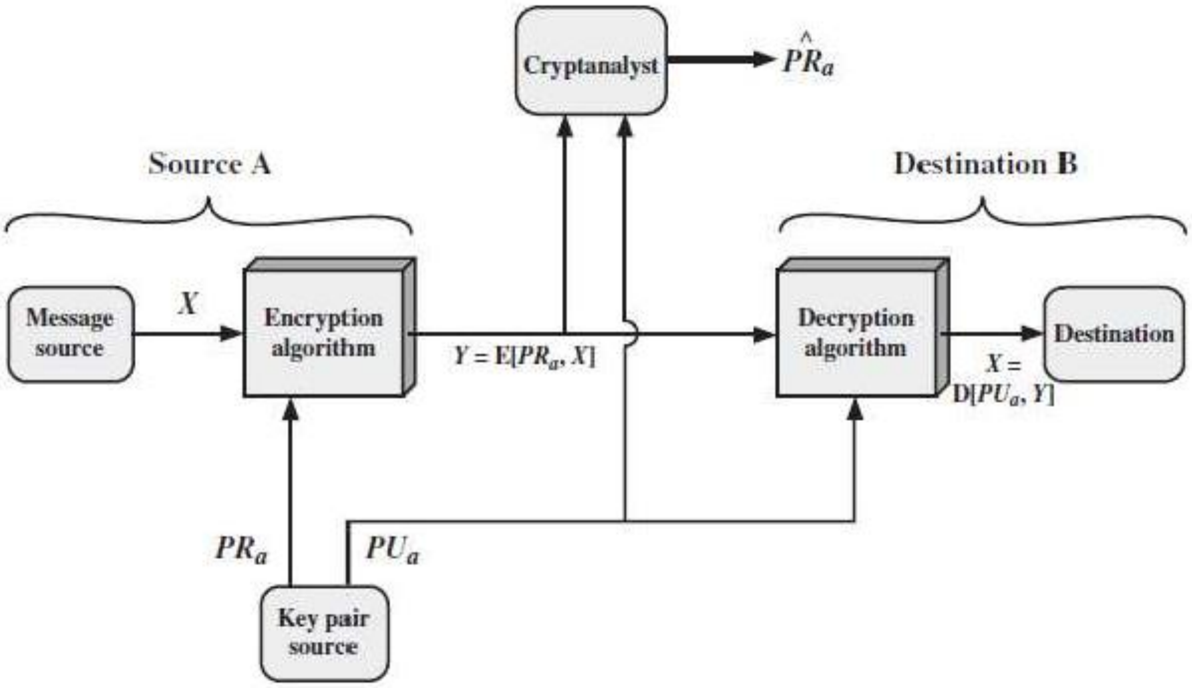
- The intended receiver, having the matching private key, is able to decrypt the message:

$$X = D(PR_b, Y)$$

- An adversary, observing Y and having access to Pu_b only, may attempt to recover X and/or PR_b.

- If the adversary is interested only in this particular message, then the focus of effort is to recover X by generating a plaintext estimate. $\hat{X}$
- Whereas if the adversary is interested in being able to read future messages as well, then he attempts to recover PR_b by generating an estimate $\hat{PR_b}$

Public Key Cryptosystem: Authentication



Authentication using public key cryptography

- However, the above scheme does not provide authentication of sender as, anyone having access to the public key can encrypt the message.
- Public-key encryption can be used to provide authentication in the following manner:
  - When A wishes to send a message to B where confidentiality is not needed but authentication is required, A encrypts the message using PR_a.
  - Anyone having access to PU_a can decrypt the message. However, one thing is sure that the message originated from A since no one except A could have encrypted the message using PR_a.
- A prepares a message to B and encrypts it using A’s private key before transmitting it.

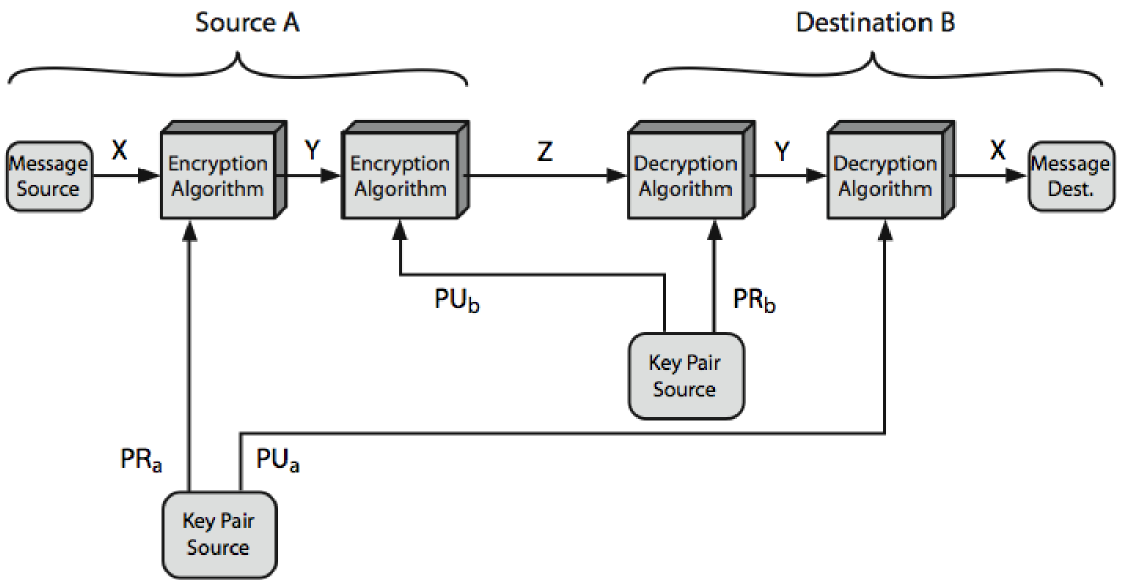
$$Y = E (PRa, X)$$

- B can decrypt the message using A’s public key.

$$X = D (PUa, Y)$$

- Because the message was encrypted using A’s private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a digital signature.
- In addition, it is impossible to alter the message without access to A’s private key, so the message is authenticated both in terms of source and in terms of data integrity.
- However, the entire message needs to be stored to bring up in case of dispute.
- A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document.
- Such a block, called an authenticator.
- It must have the property that it is infeasible to change the document without changing the authenticator.
- If the authenticator is encrypted with the sender’s private key, it serves as a signature.

Public Key Cryptosystem: Authentication and Secrecy



- Authentication and Secrecy both can be achieved by combining above both techniques.
  - First sender A encrypt message X with private key of A.

    Y = E (PRa, X)
  - Then again A encrypt Y with public key of B.

    Z = E (PUb, Y)
  - Then send Z.
  - Only B can decrypt Z as it is encrypted with public key of B. So it gives Secrecy.

    Y = D (PRb, Z)
  - Now Y can be decrypted with public key of A. So it gives authentication.

    X = D (PUa, Y)
- So by using public key cryptography we can achieve secrecy and authentication.

Applications of Public Key Cryptography

- Applications of public-key cryptosystems can classified into three categories:
  1. **Encryption /decryption:** The sender encrypts a message with the recipient's public key.
  2. **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.
  3. **Key exchange:** Two sides cooperate to exchange a session key. Several different methods are possible.
- Some algorithm like RSA and Elliptic Curve are suitable for all three applications whereas others can be used for one or two of these applications.

Requirements for Public Key Cryptography

- Requirements that public key algorithms must fulfill are:
  - It is computationally easy for a party B to generate a key pair.

- It is computationally easy for a sender A, knowing the public key and themessage **M**, to generate the corresponding ciphertext and for the receiver B to decrypt the resulting ciphertextusing the private key to recover the original message.
- It is computationally infeasible for an adversary, knowing the public key, **PU_b**, todetermine the private key,**PR_b**.
- It is computationally infeasible for an adversary, knowing the public key, **PU_b**, and a ciphertext, **C**, to recover the original message, **M.**
- The two keys can be applied in either order:

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

- These are requirements that only a few algorithms have been able to fulfill. Some of these are RSA, elliptic curve cryptography, Diffie-Hellman, & DSS.

Public Key Cryptanalysis

Brute force attack

- This attack includes trying all the alternate keys until the correct key is found.
- Counter measure to this is use large keys.
- However, public-key systems depend on the use of some sort of invertible mathematical function which is really time-consuming and increases overhead.
- Thus, there is a tradeoff. The key size must be large enough to make brute-force attack impractical but small enough for practical encryption and decryption.
- Secure keys are long enough to make encryption decryption really slow.
- Hence, public-key encryption is currently confined to key management and signature applications

Computation of private key from public key

- In this attack, some characteristics of algorithm are exploited to calculate the private key from public key.
- This attack needs many known or chosen plaintext-ciphertext pairs.
- To date it has not been mathematically proven that this form of attack is infeasible for a particular algorithm. Thus any given algorithm is suspect.

Probable message attack

- In this attack, the opponent has some idea about the plaintext and he uses this information to find the private key.
- Suppose that a message consist only 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-bitkey.
- This attack can be prevented by appending some random bits to such simple messages.

The RSA Algorithm

- RSA algorithm processes plaintext blocks, with each block having a binary value less than some number n.
- The block size must be less than or equal to $\log_2(n) + 1$.
- Steps for RSA:

  - Select two large prime numbers **p** and **q.**
  - Calculate n = pq.
  - Calculate ɸ(n) = (p - 1)(q - 1).
  - Select e such that e is relatively prime to ɸ(n).
  - Compute d such that d*e ≡ 1 (mod ɸ(n)).
- RSA is a public key algorithm with public key PU = {e, n} and private key PR = {d, n}.
- Encryption and decryption are of the following form, forsome plaintext block M and ciphertext block C:

$$\mathbf{C = M^{e} mod\ n}$$
$$\mathbf{M = C^{d} mod\ n}$$
$$\mathbf{M= (M^{e})^{d}\ mod\ n}$$

- For the above equation to be true, d must be an inverse of e.
- D can be calculated from e using extended Euclid’s algorithm.
- 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.
- RSA can also be subjected to various attacks like brute-force attack, various mathematical attacks, timing attacks and chosen ciphertext attacks.
- Some of these attacks exploit the mathematical characteristics of RSA.

RSA Example

- Let p = 17 and q = 11.
- n = pq = 17 X 11 = 187
- ɸ(n) = (p-1)(q-1) = 16 X 10 = 160
- Let e be 7.
- d = e^{-1} mod 160 = 23 (can be calculated by extended Euclid’s algorithm).
- Now, PU = { 7, 187 } and PR = { 23, 187 }
- If M = 88, then by RSA

Encryption

C = 88^7 mod 187
= [88 X 88^2 X 88^4] mod 187
= 11

Decryption

- Here, C = 11.

M = 11^{23} mod 187
= [11 X 11^2 X 11^4 X 11^8 X 11^8] mod 187
= 88

Computational Aspects of RSA

- **Use of modular arithmetic makes calculation practical:**
  - Both encryption and decryption in RSA involve calculating huge exponents, mod n.
  - If the exponentiation is done over the integers and then reduced modulo n, the intermediate values would be extremely large.
  - However, the following property of modular arithmetic makes the calculation practical:
    [(a mod n) * (b mod n)] mod n = (a * b) mod n]

- **Efficiency of exponentiation:**
  - RSA deals with very large exponents.
  - But this operation can be implemented efficiently.
  - Consider x^{16}. A straightforward approach requires multiplying x 16 times.
  - But, the same can be achieved by only four multiplications - x^2,$(x^2)^2=x^4$, $(x^4)^2=x^8$, $(x^8)^2=x^{16}$.
- **Efficient operation using the public key:**
  - To speed up the operation of the RSA algorithm using the public key, a specific choice of e is usually made.
  - The most common choice is 65537 ($2^{16} + 1$).

The Security of RSA

Four possible approaches to attacking the RSA algorithm are

Brute force

- This involves trying all possible private keys.
- The defense against this attack is to use a large key.
- However, the key should not be so large that it makes calculation too time consuming and hence impractical.
- Thus, there is a tradeoff between key size and security of RSA.

Mathematical attacks

- There are three approaches to attacking RSA mathematically, all of which are equivalent in effort to the factoring the product of two primes.
  - Factor n into its two prime factors. This enables calculation of $\phi(n) = (p - 1)(q - 1)$, which in turn enables determination of $d = e^{-1} \pmod{\phi(n)}$.
  - Determine $\phi(n)$ directly, without first determining p and q. Again, this enables determination of $d = e^{-1} \pmod{\phi(n)}$. This is equivalent to factoring n.
  - Determine d directly, without first determining $\phi(n)$ which is at least as time-consuming as the factoring problem.
- Size of **n** should be considerably large.
- To avoid values of **n** that may be factored more easily, the
  - **P** and **q** should differ in length by only a few digits.
  - Both (p - 1) and (q - 1) should contain a large prime factor.
  - gcd(p - 1, q - 1) should be small.

Timing attacks

- These depend on the running time of the decryption algorithm.
- It is a ciphertext only attack.
- In RSA, modular exponentiation is done bit by bit. Suppose the system uses a modular multiplication function that is very fast in almost all cases but in a few cases takes much more time than an entire average modular exponentiation.
- The attack proceeds as follows:
  - Suppose that the first j bits are known.
  - For a given ciphertext, the attacker can complete the first j iterations of the for-loop.
  - The operationof the subsequent step depends on the unknown exponent bit.

- For a few values of e and d, the modular multiplication will be extremely slow, and the attacker knows which these are.
- Therefore, if the observed time to execute the decryption algorithm is always slow when this particular iteration is slow with a 1 bit, then this bit is assumed to be 1.
- If a number of observed execution times for the entire algorithm are fast, then this bit is assumed to be 0.

- Generally modular exponentiation implementations do not have such extreme timing variations but there is enough variation to make this attack practical.
- Countermeasures to this attack are:
  - **Constant exponentiation time:** Ensure that all exponentiations take the same amount of time before returning a result. However, this degrades performance.
  - **Random delay:** Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack. But if the defenders don't add enough noise, attackers could still succeed by 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 that is essential to the timing attack. Steps for blinding are:
    - ✓ Generate a secret random number **r** between 0 and n – 1.
    - ✓ **$C' = C^{(r^e)}$ mod n**, where e is the public exponent.
    - ✓ Compute **$M' = (C')^d$ mod n**
    - ✓ Compute **$M = M' r^{-1}$ mod n**, r^{-1} is the multiplicative inverse of r mod n

    Implementing blinding incurs only a 2 to 10% performance penalty.

Chosen ciphertext attacks

- This type of attack exploits properties of the RSA algorithm.
- The adversary could select a plaintext, encrypt it with the target's public key, and then gets the plaintext back by having it decrypted with the private key.
- This provides no new information. Instead, the adversary exploits properties of RSA and selects blocks of data that, when processed using the target's private key, gives information needed for cryptanalysis.
- An example of one such attack is that the attacker exploits the following property of RSA.

$$\mathbf{E(PU,M1) \times E(PU,M2) = E(PU, [M1 \times M2])}$$

  - Compute $X = (C \times 2^e) \bmod n$.
  - Submit X as a chosen ciphertext and receive back $Y = X^d \bmod n$
  - But now note that

$$\begin{aligned} X &= (C \bmod n) * (2^e \bmod n) \\ &= (M^e \bmod n) * (2^e \bmod n) \\ X &= (2M)^e \bmod n \end{aligned}$$

  - Therefore, $Y = (2M) \bmod n$.
- To overcome this simple attack, randomly pad the plaintext before encryption.
- This randomizes the ciphertext so that the Equation no longer holds.

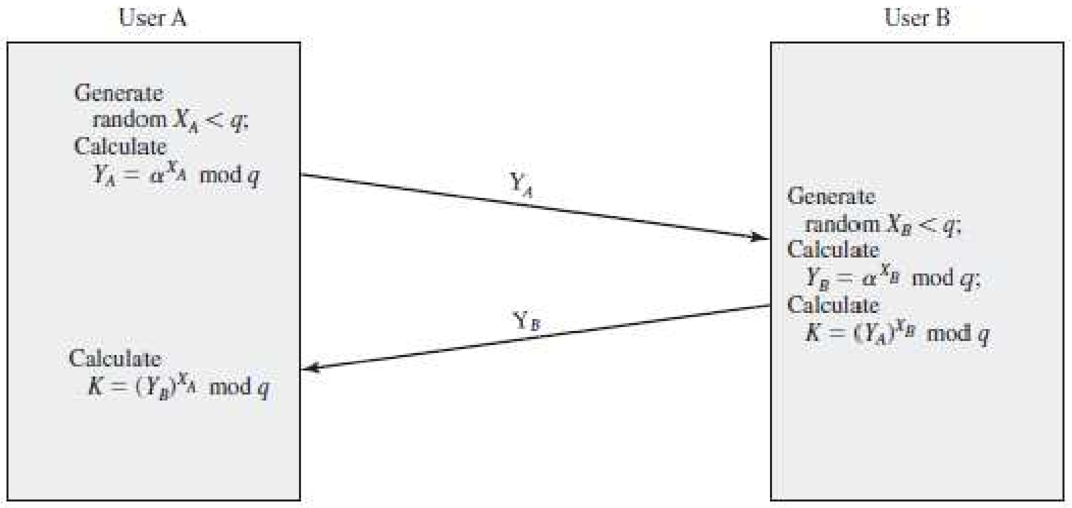
Diffie-Hellman key exchange

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

- The Diffie-Hellman algorithm depends on the difficulty of computing discrete logarithms.
- For this scheme, there are two publicly known numbers: a prime number **q** and an integer **α** that is a primitive root of **q.**
- Suppose the users A and B wish to exchange a key.
- User A selects a random private integer $X_A < q$ and computes public integer
  $Y_A = \alpha^{XA} \bmod q.$
- Similarly, user B independently selects a random private integer $X_B < q$ and computes public integer
  $Y_B = \alpha^{XB} \bmod q.$
- Each side keeps the **X** value private and makes the **Y** value available publicly to the other side.
- User A computes the key as
  $K = (Y_B)^{XA} \bmod q$ and
- User B computes the key as
  $K = (Y_A)^{XB} \bmod q.$
- These two calculations produce identical results:

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

- The result is that the two sides have exchanged a secret value.
- Furthermore, because X_A and X_B are private, an adversary only has the following information: q, α, Y_A and Y_B.
- Thus, the adversary is forced to take a discrete logarithm to determine the key.
- For example, to determine the private key of user B, an adversary must compute
  $X_B = \mathrm{dlog}_{a,q}(Y_B)$
- The adversary can then calculate the key K.
- The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms.
- For large primes, calculating discrete logarithms is considered infeasible.


User A
User B
Generate random $X_A < q$;
Calculate $Y_A = \alpha^{X_A} \bmod q$
Y_A
Generate random $X_B < q$;
Calculate $Y_B = \alpha^{X_B} \bmod q$;
Calculate $K = (Y_A)^{X_B} \bmod q$
Y_B
Calculate $K = (Y_B)^{X_A} \bmod q$

Diffie-Hellman Key Exchange

- Because only A and B can determine the key, no other user can read the message (confidentiality).
- Recipient B knows that only user A could have created a message using this key (authentication).
- However, the technique does not protect against replay attacks. One such example is Man-in-the-Middle Attack.

Man-in-the-Middle Attack

- Suppose A and B wish to exchange keys, and E is the attacker.
- The attack proceeds as follows.
  - E generates two random private keys $\mathbf{X_{E1}}$andand$\mathbf{X_{E2}}$then computing the corresponding public keys $\mathbf{Y_{E1}}$and $\mathbf{Y_{E2}}$.
  - A transmits **YA** to B.
  - E intercepts $\mathbf{Y_A}$ and transmits ‘ $\mathbf{Y_{E1}}$ to B.
  - B receives $\mathbf{Y_{E1}}$ and calculates $\mathbf{K_1 = (Y_{E1})^{XB} mod\ q.}$
  - B transmits $\mathbf{Y_B}$to A.
  - E intercepts $\mathbf{Y_B}$and transmits $\mathbf{Y_{E2}}$ to A.
  - A receives$\mathbf{Y_{E2}}$ and calculates $\mathbf{K_2 = (Y_{E2})^{XA}\ mod\ q.}$
  - E also calculates $\mathbf{K_1 = (Y_B)^{XE1} mod\ q}$and $\mathbf{K_2 = (Y_A)^{XE2}\ mod\ q}$
- At this point, B and A think that they share a secret key, but instead Band E share secret key and A and E share secret key.
- All future communication between B and A is compromised in the following way.
  - A sends an encrypted message **M** as $\mathbf{E(K_2, M)}$.
  - E intercepts the encrypted message and decrypts it to recover M.
  - E sends $\mathbf{E(K_1, M)}$ or $\mathbf{E(K_1, M')}$ to B , where **M’** is any message.
- 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.