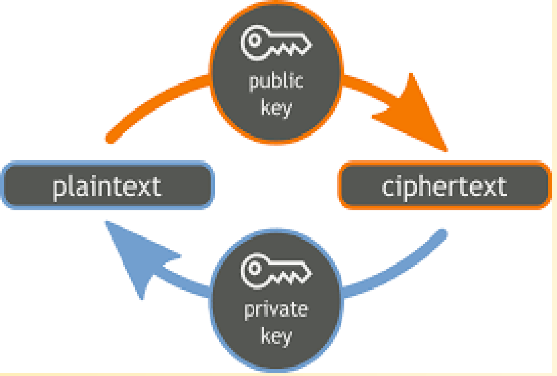


Public Key Cryptography
(Asymmetric Key Cryptography)

Mukesh Chinta
Assistant Professor
CSE, VRSEC

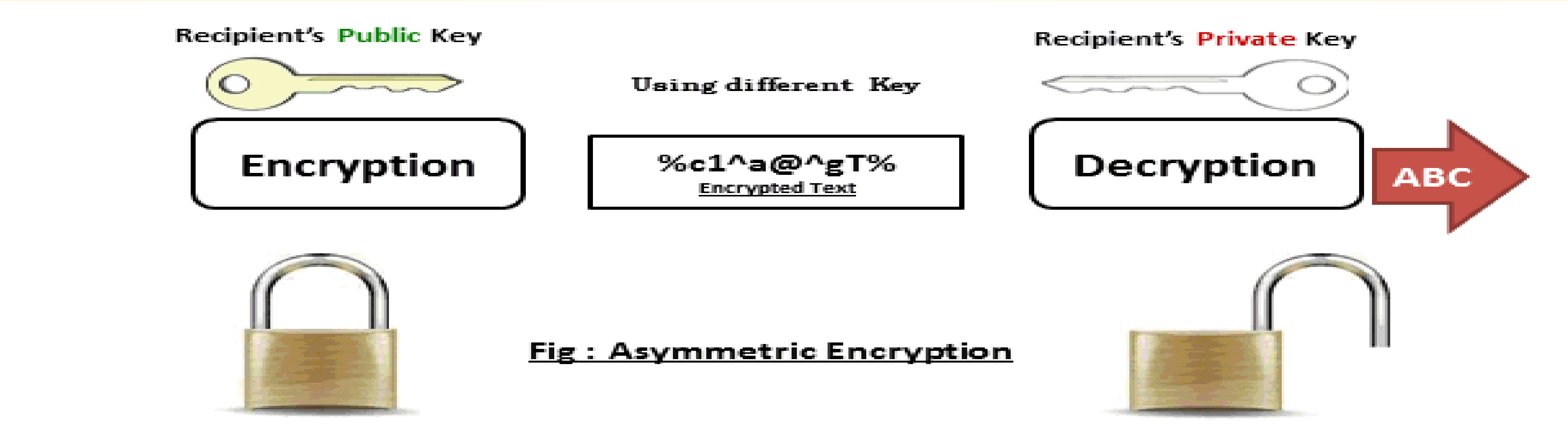


Fig : Asymmetric Encryption

Asymmetric Encryption – Public Key Cryptography

Every Egyptian received two names, which were known respectively as the true name and the good name, or the great name and the little name; and while the good or little name was made public, the true or great name appears to have been carefully concealed.

—The Golden Bough, Sir James George Frazer

- ☺ **Asymmetric encryption is a form of cryptosystem in which encryption and decryption are performed using the different keys—one a public key and one a private key. It is also known as public-key encryption**

- ☺ **Asymmetric encryption transforms plaintext into ciphertext using a one of two keys and an encryption algorithm. Using the paired key and a decryption algorithm, the plaintext is recovered from the ciphertext**

Public Key Cryptography

- The development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography

- It is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, that uses only one key. Anyone knowing the public key can encrypt messages or verify signatures, but cannot decrypt messages or create signatures, counter-intuitive though this may seem.

- The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

- It works by the clever use of number theory problems that are easy one way but hard the other.

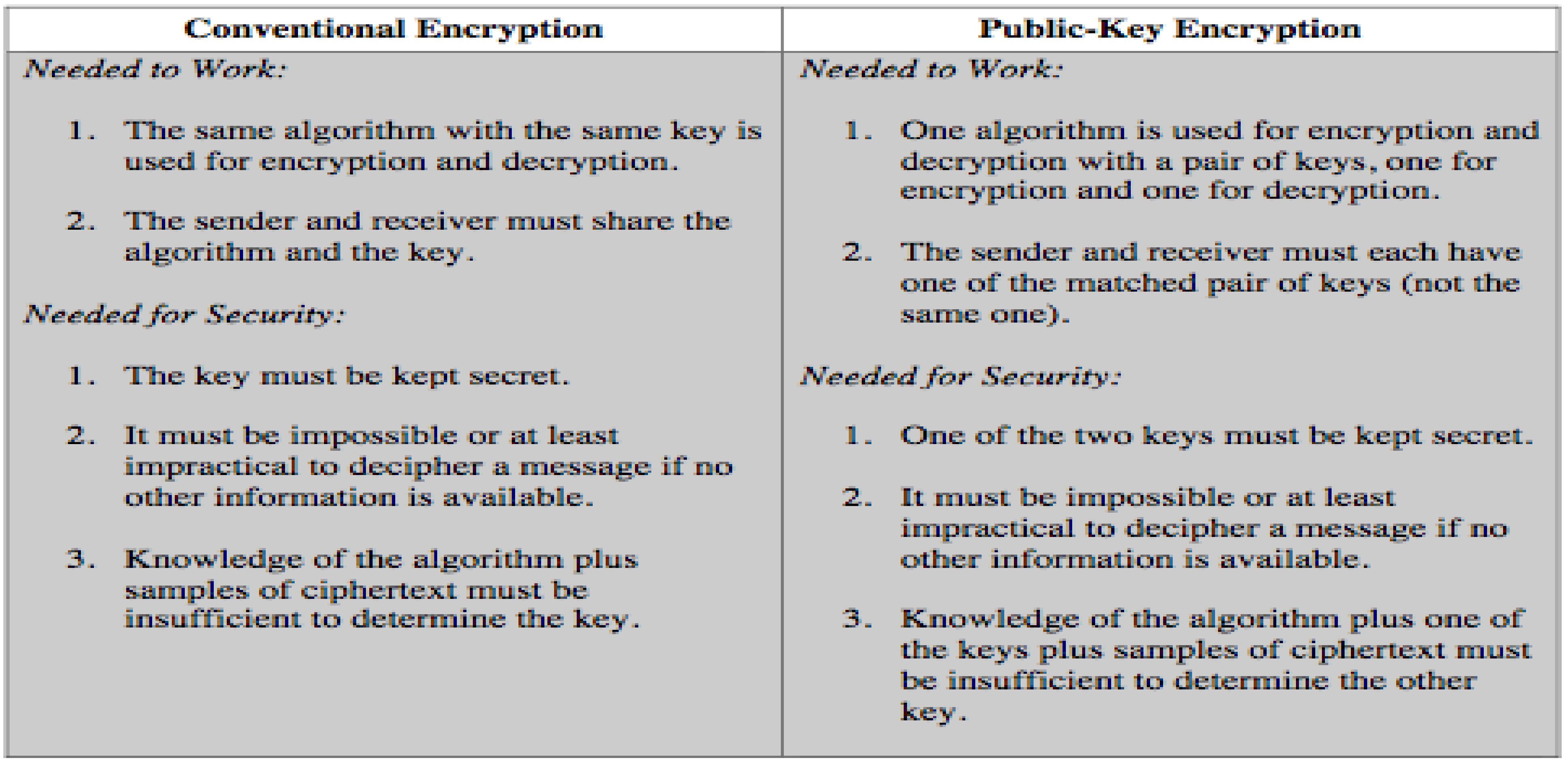
- Asymmetric cryptography complements **rather than** replaces symmetric cryptography: the advantages of one can compensate for the disadvantages of the other.

- Symmetric and asymmetric-key cryptography will exist in parallel and continue to serve the community.

History

- ❖The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption:
  - ►**Key distribution**: how to have secure communications in general without having to trust a KDC with your key
  - ►**Digital signatures**: how to verify a message comes intact from the claimed sender
- ❖The idea of public key schemes, and the first practical scheme, which was for key distribution only, was published in 1976 by Diffie & Hellman. The concept had been previously described in a classified report in 1970 by James Ellis (UK CESG) - and subsequently declassified [ELLI99]
- ❖Its interesting to note that they discovered RSA first, then Diffie-Hellman, opposite to the order of public discovery!

A Quick Comparison of both techniques



Conventional Encryption	Public-Key Encryption
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.	*Needed to Work:* 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: 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.	*Needed for Security:* 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/two-key/Asymmetric cryptography involves the use of two keys:

- a **public-key**, which may be known by anybody, and can be used to encrypt messages, and verify signatures
- a **related private-key**, known only to the recipient, used to decrypt messages, and sign (create) signatures

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.

In addition, some algorithms, such as RSA, also exhibit the following characteristic:

- 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.

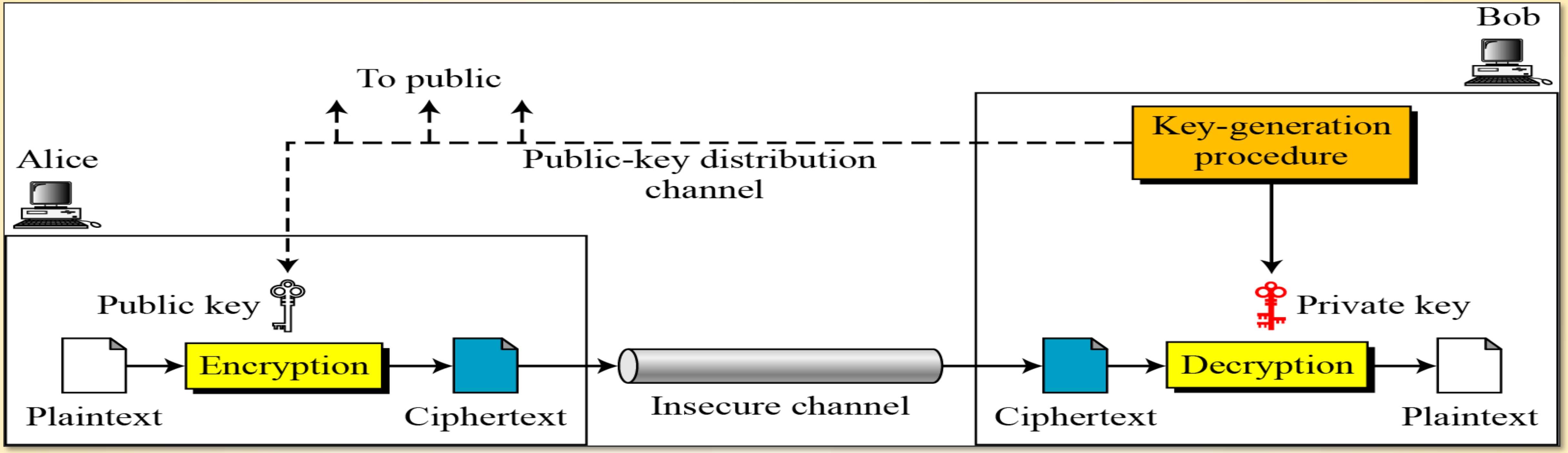
A Public-key encryption scheme has six ingredients**:**

- Plaintext
- Encryption Algorithm
- Public and private keys
- Ciphertext
- Decryption Algorithm

Steps for Asymmetric Encryption

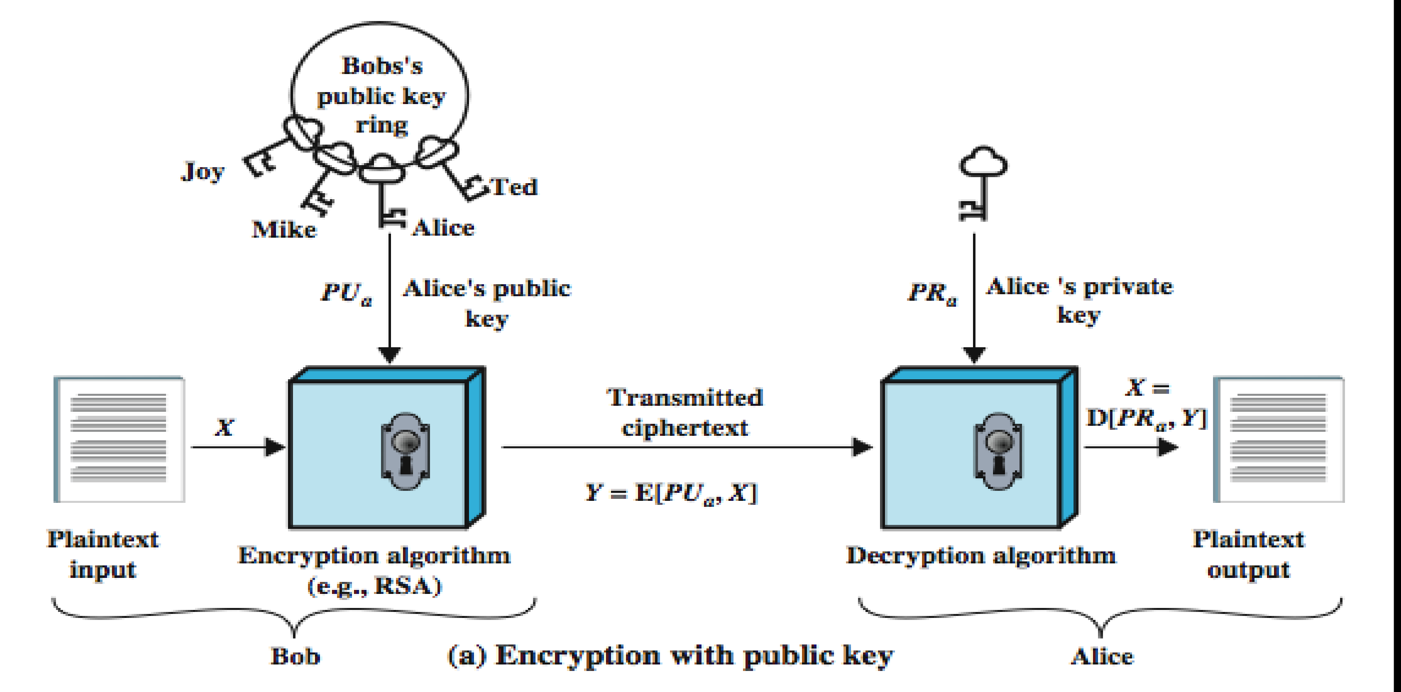
- Each user generates a pair of keys to be used for the encryption and decryption of messages.
- Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key (private key) is kept. Each user maintains a collection of public keys obtained from others.
- If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
- 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.

Illustration of Public Key Cryptography



- ✓ 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

Symmetric-key cryptography is based on sharing secrecy; asymmetric-key cryptography is based on personal secrecy.



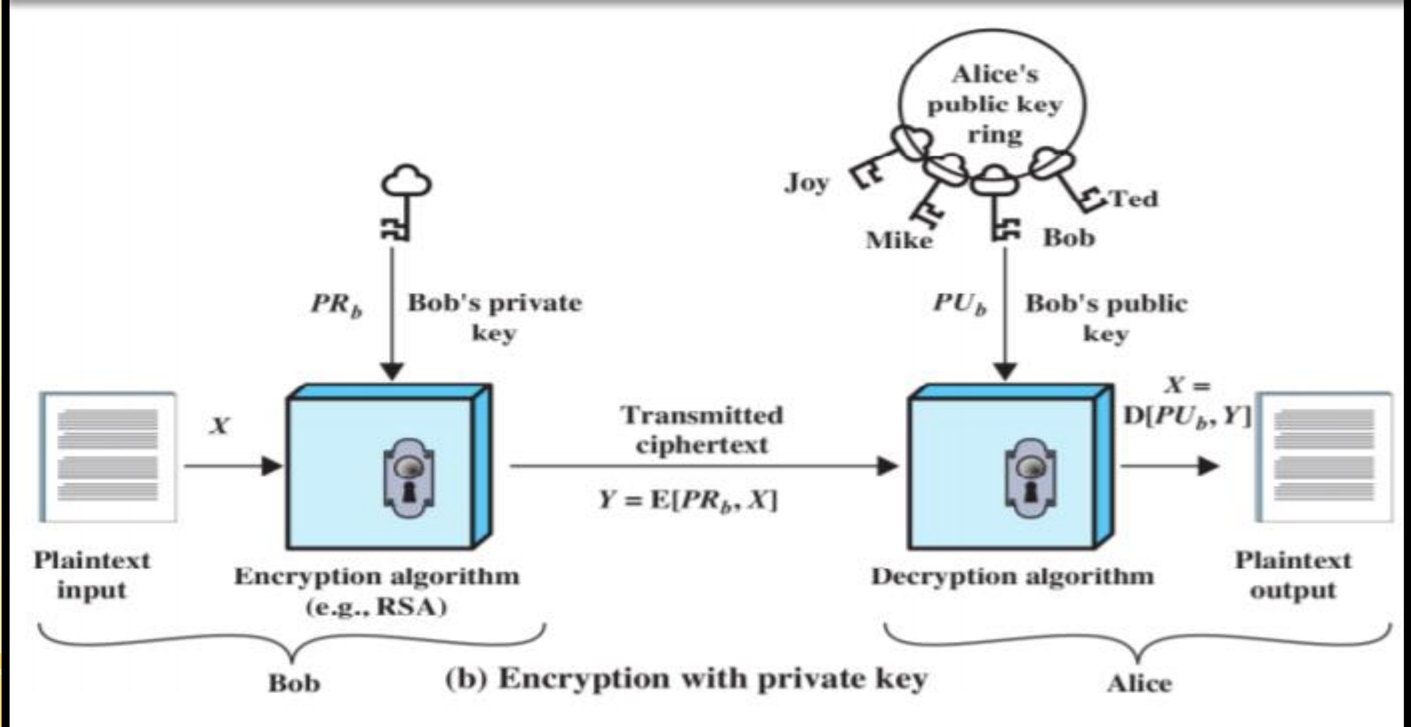
(a) Encryption with public key

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

Provides secrecy

Encryption/Decryption

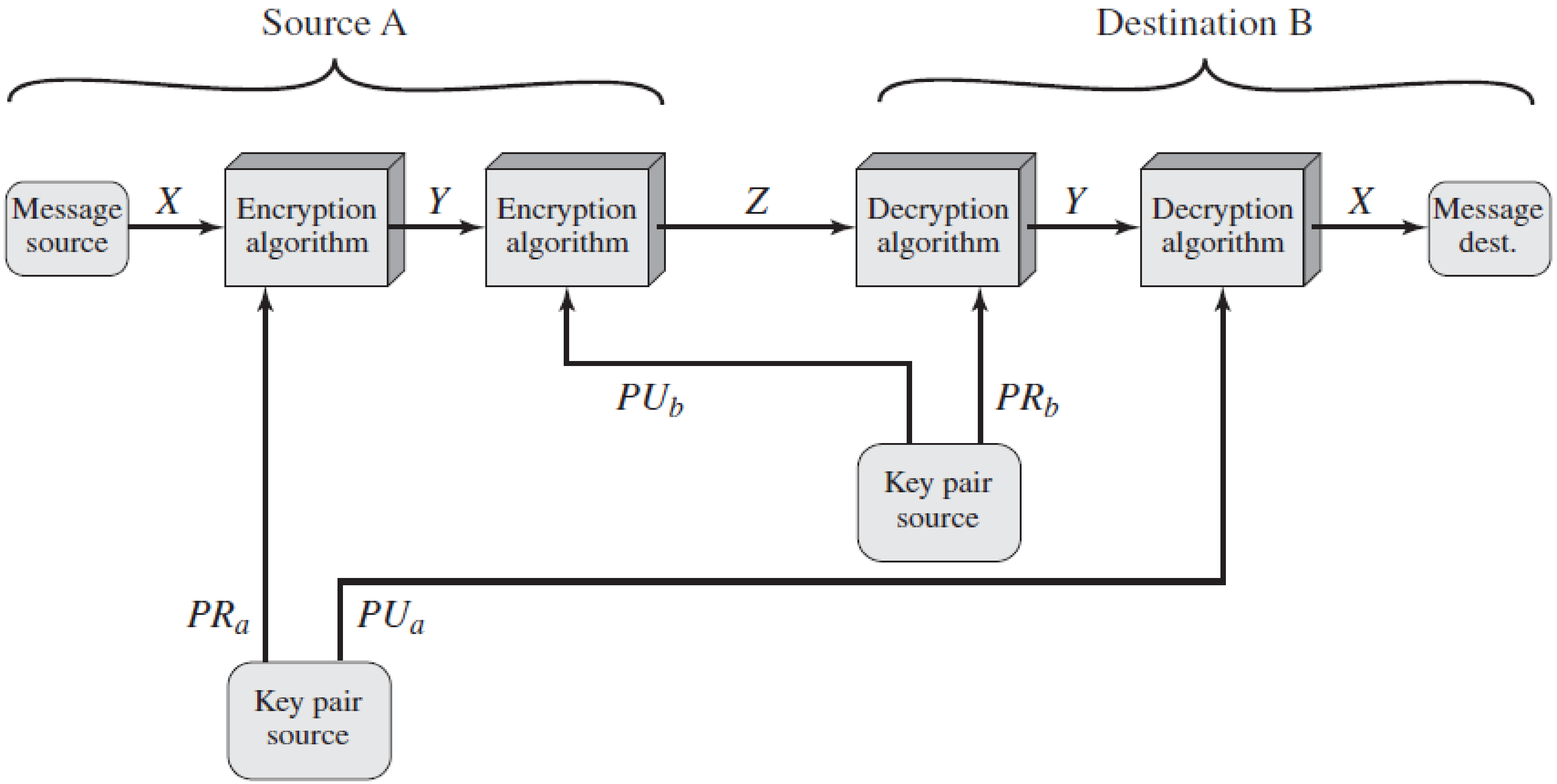
$$C = e(K_{public}, P) \qquad P = d(K_{private}, C)$$



(b) Encryption with private key

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

Provides Digital Signature

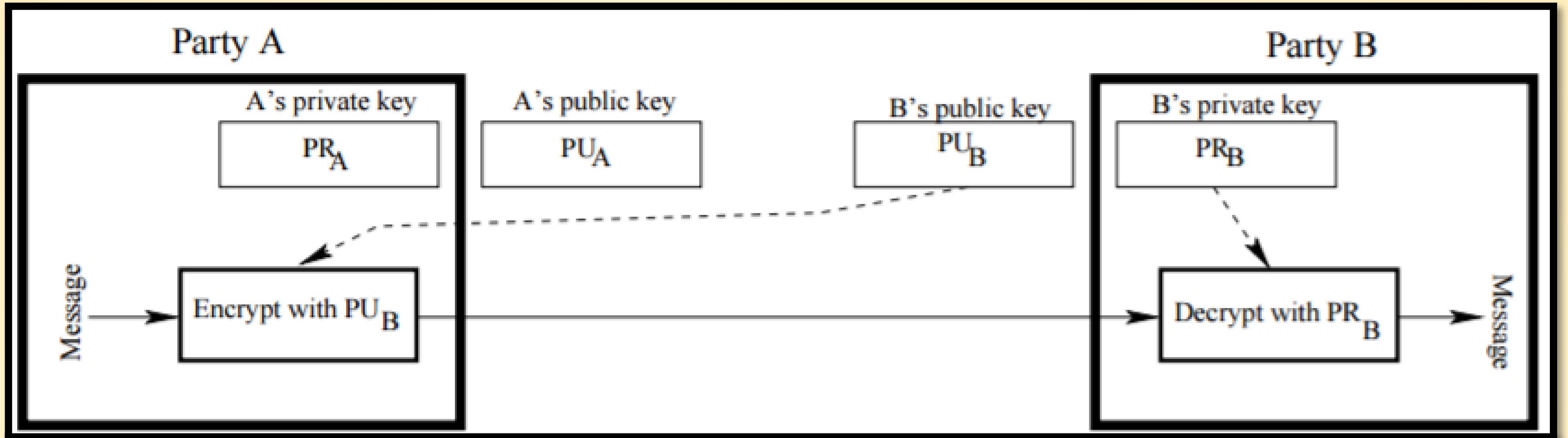


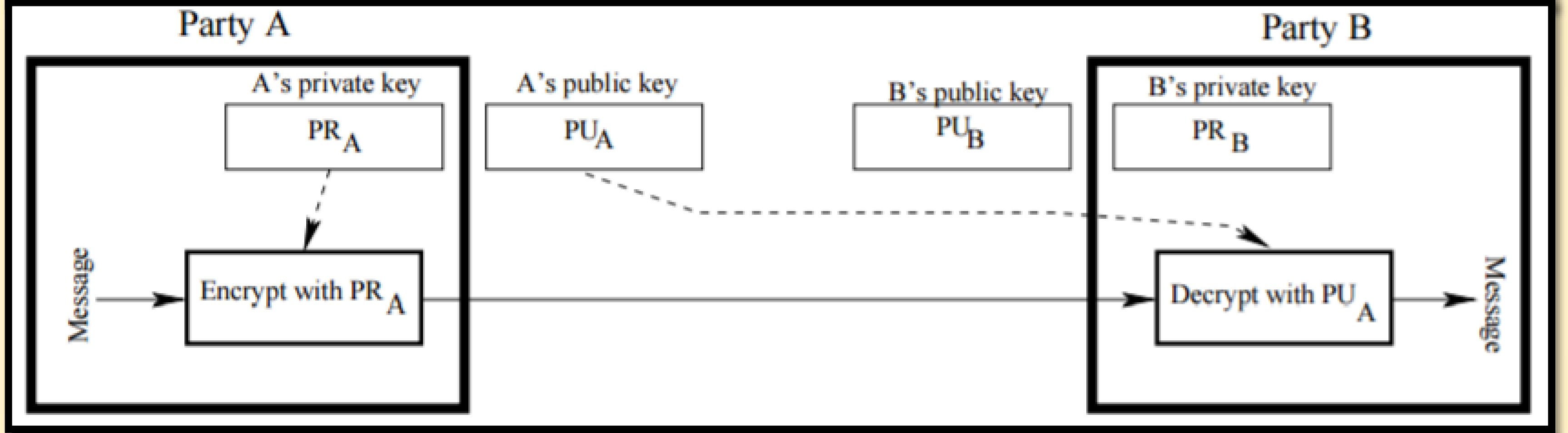
$$Z = E(PU_b, E(PR_a, X))$$

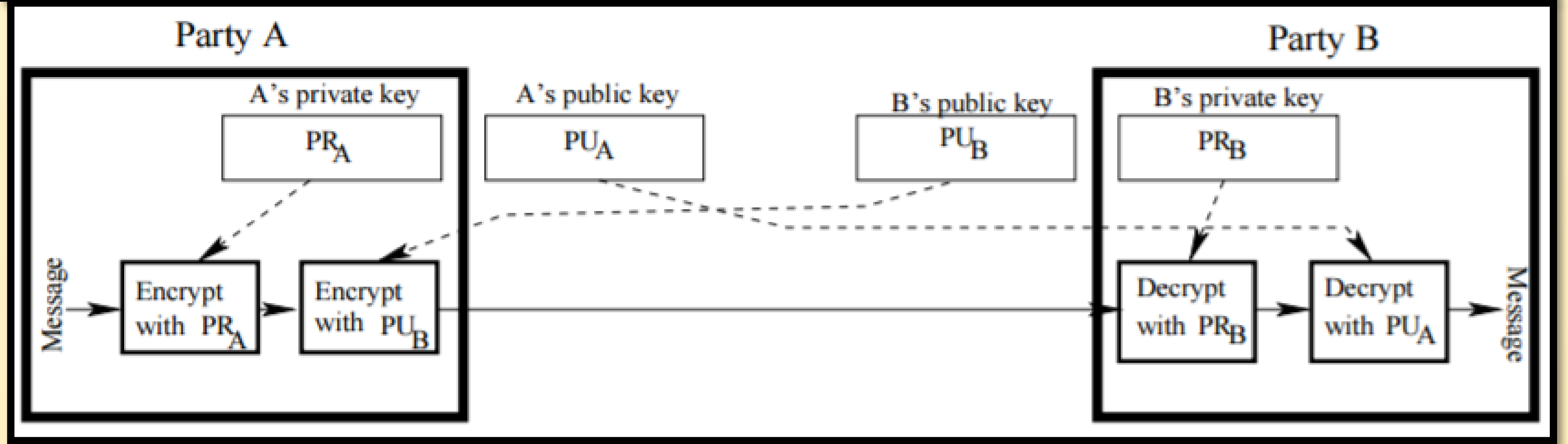
$$X = D(PU_a, D(PR_b, Z))$$

- First the message is encrypted, using the sender's private key. This provides the digital signature.
- Next, we encrypt again, using the receiver's public key.
- The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided.
- The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication

IDENTIFY

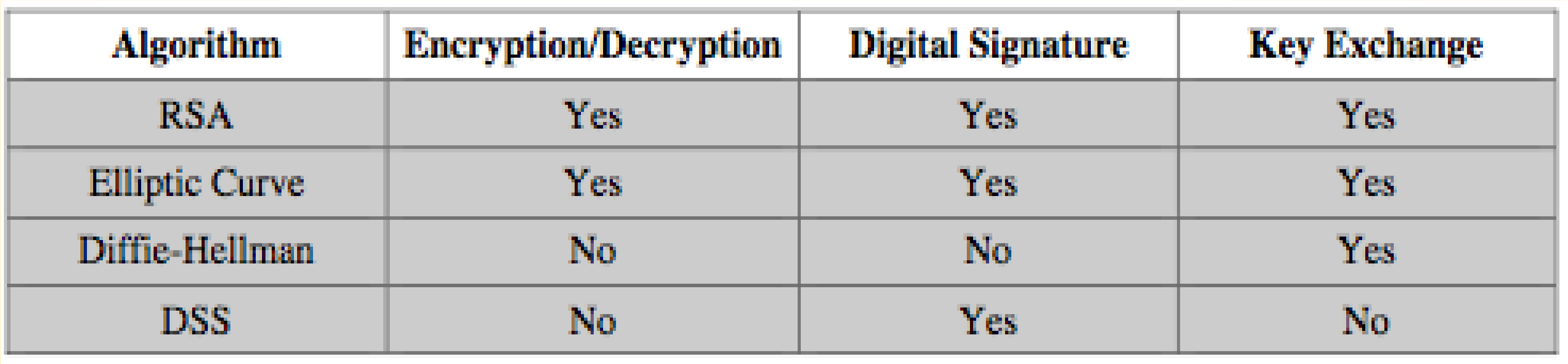

Party A
Party B
A’s private key
PR_A
A’s public key
PU_A
B’s public key
PU_B
B’s private key
PR_B
Message
Encrypt with PU_B
Decrypt with PR_B
Message


Party A
Party B
A’s private key
PR_A
A’s public key
PU_A
B’s public key
PU_B
B’s private key
PR_B
Message
Encrypt with PR_A
Decrypt with PU_A
Message


Party A
Party B
A’s private key
PR_A
A’s public key
PU_A
B’s public key
PU_B
B’s private key
PR_B
Message
Encrypt with PR_A
Encrypt with PU_B
Decrypt with PR_B
Decrypt with PU_A
Message

Public-Key Applications

- Public-key systems are characterized by the use of a cryptographic type of algorithm with two keys. Depending on the application, the sender uses either the sender's private key or the receiver's public key, or both, to perform some type of cryptographic function. In broad terms, we can classify the use of public-key cryptosystems into the 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, either to the whole 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



Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Requirements of Public Key Cryptography

Diffie and Hellman postulated the asymmetric system without demonstrating that such algorithms exist. However, they did lay out the conditions that such algorithms must fulfill:

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

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

These are formidable requirements which only a few algorithms have satisfied

Security of Public Key Cryptography

- 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:
  - $Y = f(X)$ easy
  - $X = f^{-1}(Y)$ infeasible
- A **trap-door one-way function**, which is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known.
  - $Y = f_k(X)$ easy, if k and X are known
  - $X = f_k^{-1}(Y)$ easy, if k and Y are known
  - $X = f_k^{-1}(Y)$ infeasible, if Y known but k not known
- A practical public-key scheme depends on a suitable trap-door one-way function.
- Public key schemes are no more or less secure than private key schemes - in both cases the size of the key determines the security
- Security relies on a large enough difference in difficulty between easy (en/decrypt) and hard (cryptanalyse) problems. More generally the hard problem is known, but is made hard enough to be impractical to break
- Requires the use of very large numbers (1024 bits) and hence is slow compared to private key schemes

When *n* is large, $\boldsymbol{n = p \times q}$ is a one-way function.

Easy Given *p* and *q* → calculate n

Difficult Given n → calculate p and q

This is the factorization problem.

When *n* is large, the function $\boldsymbol{y = x^k \bmod n}$ is a trapdoor one-way function.

Easy Given *x*, *k*, and n → calculate y

Difficult Given *y*, *k*, and *n* → *calculate x*

This is the discrete logarithm problem.

However, if we know the trapdoor, k′ such that $\boldsymbol{k \times k' = 1 \bmod \phi(n)}$, we can use $x = y^{k'} \bmod n$ to find x.

The main idea behind asymmetric-key cryptography is the concept of the trapdoor one-way function.

RSA Algorithm

- RSA was first described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman of the Massachusetts Institute of Technology and was first published in 1978 [RIVE78].
- In RSA cryptography, both the public and the private keys can encrypt a message; the opposite key from the one used to encrypt a message is used to decrypt it. This attribute is one reason why RSA has become the most widely used asymmetric algorithm: It provides a method of assuring the confidentiality, integrity, authenticity and non-reputability of electronic communications and data storage.
- Plaintext is encrypted in blocks, with each block having a binary value less than some number n. The actual RSA encryption and decryption computations are each simply a **single exponentiation mod (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}.**

To encrypt a message M the sender:
obtains public key of recipient PU={e,n}
computes: $C = M^e \bmod n$, where $0 \le M < n$

To decrypt the ciphertext C the owner:
uses their private key PR={d,n}
computes: $M = C^d \bmod n$

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

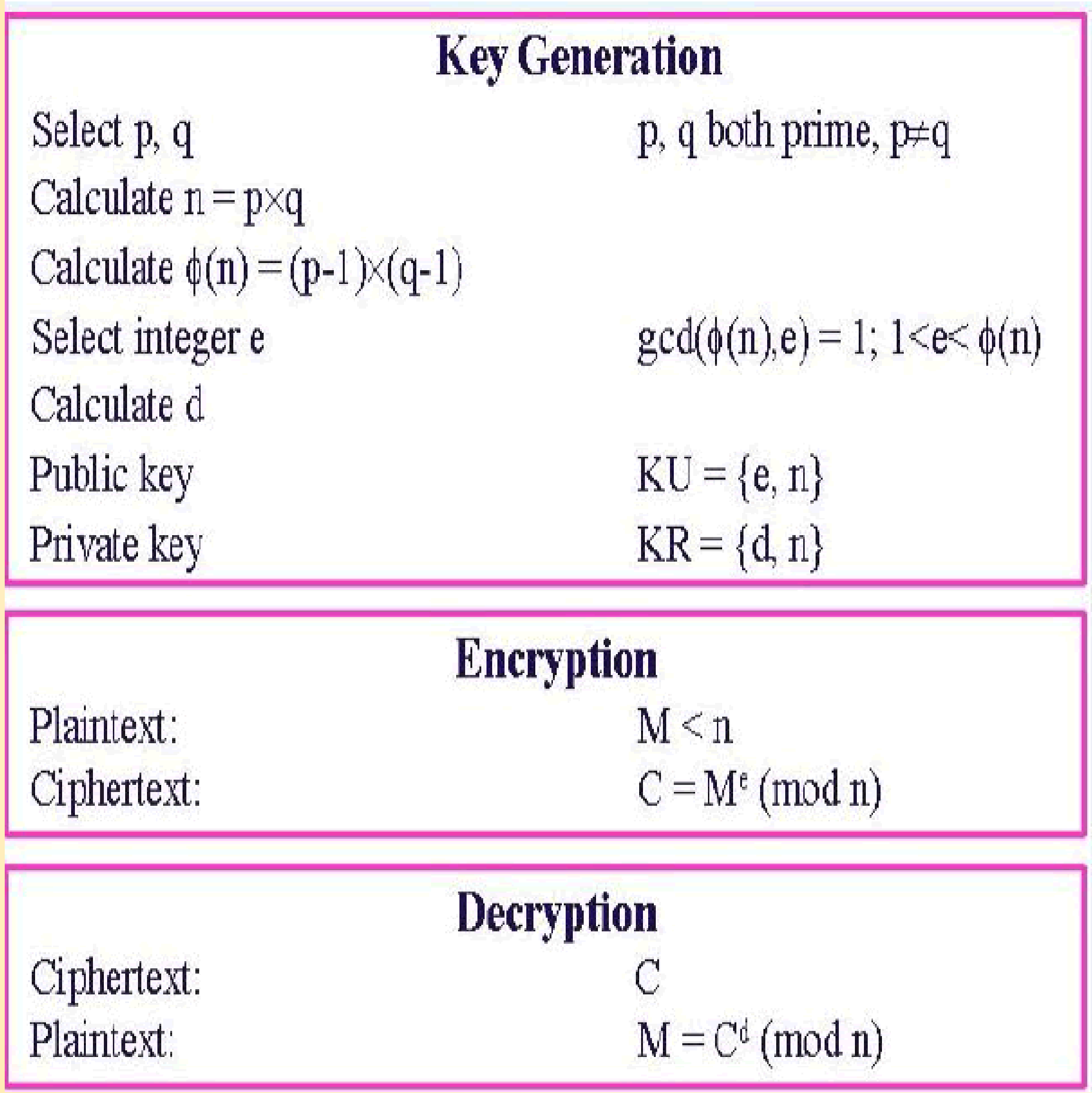
1. It is possible to find values of e, d, n such that **M^{ed} mod n = M** for all M < n.
2. It is relatively easy to calculate **M^e mod n** and **C^d mod n** for all values of M < n.
3. It is infeasible to determine **d** given e and n.

The ingredients of RSA scheme are given as:

- ❖p, q, two prime numbers (private, chosen)
- ❖n = pq (public, calculated)
- ❖e, with gcd(Ø(n), e) = 1; 1 < e < Ø(n) (public, chosen)
- ❖d = e^{-1} (mod Ø (n)) (private, calculated)

Only the public key (e,n) is published; all the other numbers involved (p,q,φ,d) must be kept private! The main property of this construction is that it is 'difficult' to compute d just from the numbers e and n ('difficult' in the sense that the computation is more and more time-consuming)

RSA Algorithm



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

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

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

1. Select primes: **$p=3$ & $q=11$**
2. Calculate **$n = pq = 3 \times 11 = 33$**
3. Calculate **$ø(n)=(p–1)(q-1)=2 \times 10=20$**
4. Select **e: $\gcd(e,20)=1$**; choose **$e=7$**
5. Determine **d: $de=1 \bmod 20$** and $d < 20$

   Value is **$d=3$** since $3 \times 7=21 \bmod 20$

1. Publish public key **PU={7,33}**
2. Keep secret private key **PR={3,33}**

sample RSA encryption/decryption is:

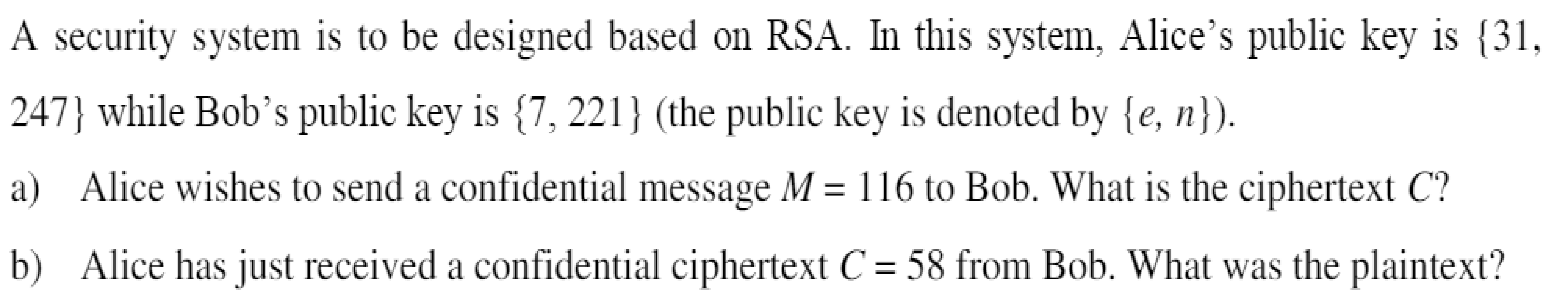
given message M = 2 (nt. 2<33)

encryption:

$$C = 2^7 \bmod 33 = 29$$

decryption:

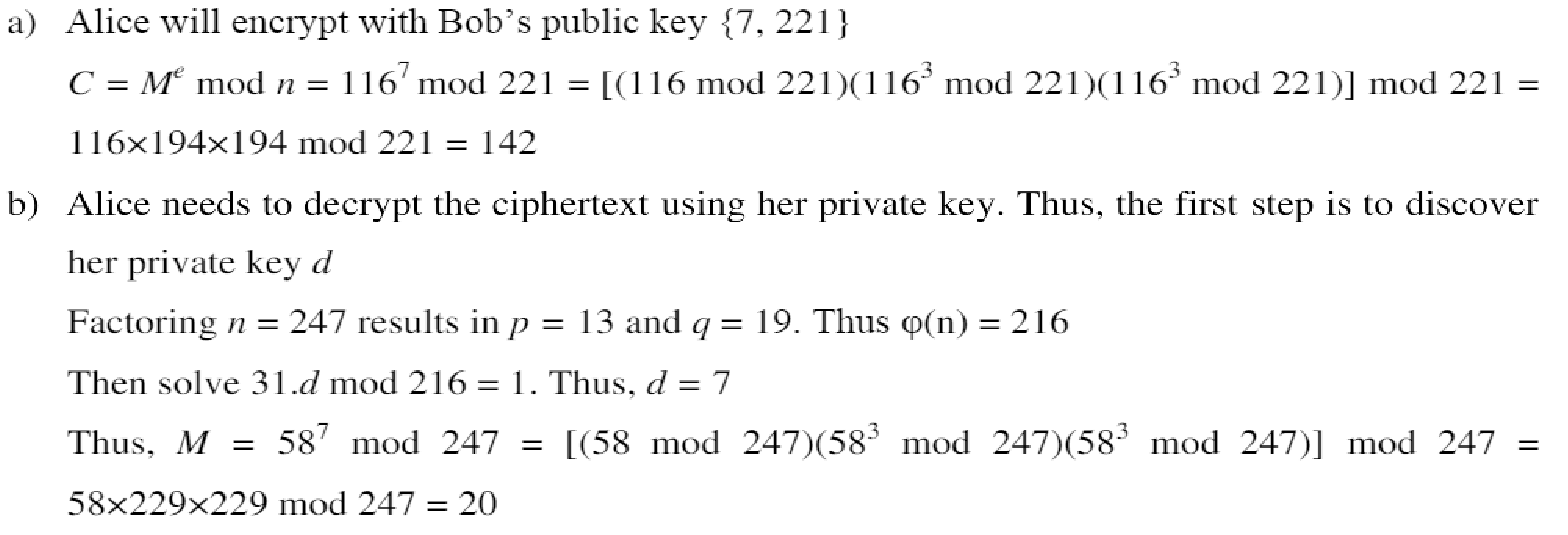
$$M = 29^3 \bmod 33 = 2$$



A security system is to be designed based on RSA. In this system, Alice's public key is {31, 247} while Bob's public key is {7, 221} (the public key is denoted by $\{e, n\}$).

a) Alice wishes to send a confidential message $M = 116$ to Bob. What is the ciphertext C?

b) Alice has just received a confidential ciphertext $C = 58$ from Bob. What was the plaintext?



a) Alice will encrypt with Bob's public key {7, 221}

$C = M^e \bmod n = 116^7 \bmod 221 = [(116 \bmod 221)(116^3 \bmod 221)(116^3 \bmod 221)] \bmod 221 = 116 \times 194 \times 194 \bmod 221 = 142$

b) Alice needs to decrypt the ciphertext using her private key. Thus, the first step is to discover her private key d

Factoring $n = 247$ results in $p = 13$ and $q = 19$. Thus $\varphi(\mathrm{n}) = 216$

Then solve $31.d \bmod 216 = 1$. Thus, $d = 7$

Thus, $M = 58^7 \bmod 247 = [(58 \bmod 247)(58^3 \bmod 247)(58^3 \bmod 247)] \bmod 247 = 58 \times 229 \times 229 \bmod 247 = 20$

Examples for RSA Algorithm

Exercise -1:

Select p= 5 and q = 11, E is selected as 13. What is the secret key? Encrypt the message M = 7

Solution – d = 37

Exercise -2:

Public key is (5,119). What is the private key? Encrypt the message M= 19

Solution – d = 77, C= 66

Exercise -3:

Public key is (7,77). Ciphertext is 53. What is the private key and Plaintext message?

Solution – d = 43, M= 25

Exercise -4:

Detected n = 187, E is selected as 7. Compute p & q. What is the secret key

Solution – p = 17, q = 11; d = 23

p=885320963, q=238855417,

n=p. q=211463707796206571

Let e=9007, ∴ d=116402471153538991

M="cat"=30120, C=113535859035722866

n=11413=101x113, so p=101, q=113

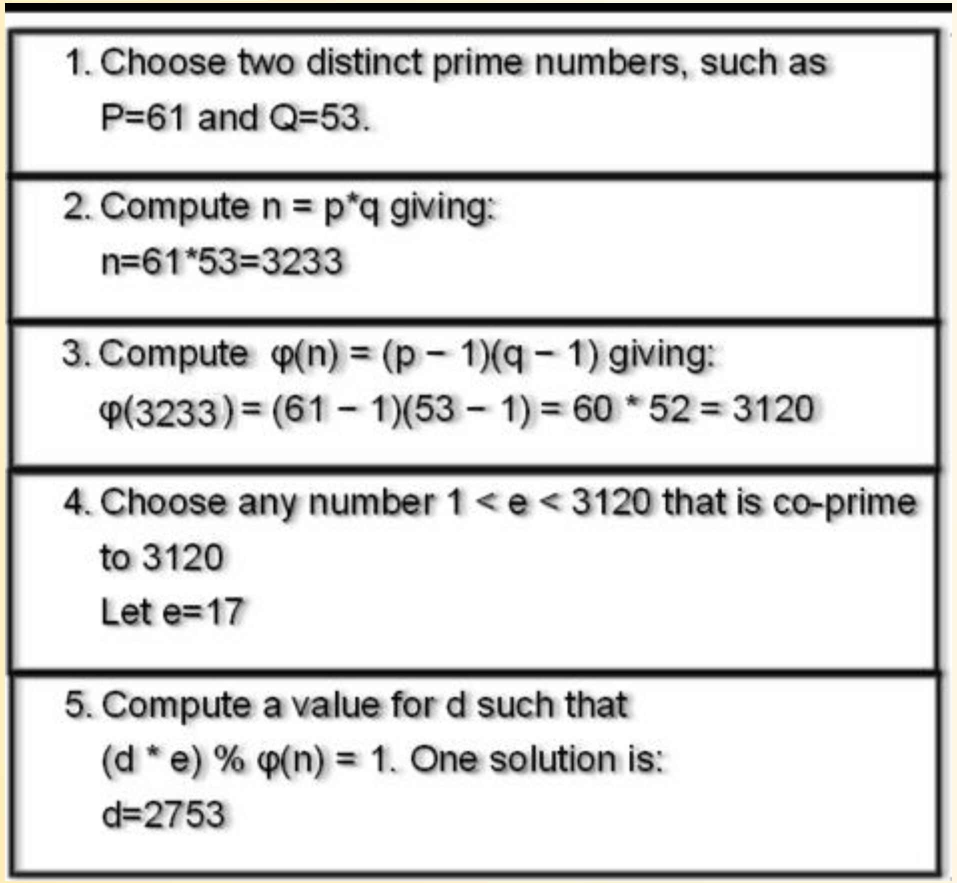
Ø(n)=(p-1)x(q-1) = 100x112=11200

Choose e=7467, then gcd(e, Ø(n))=1

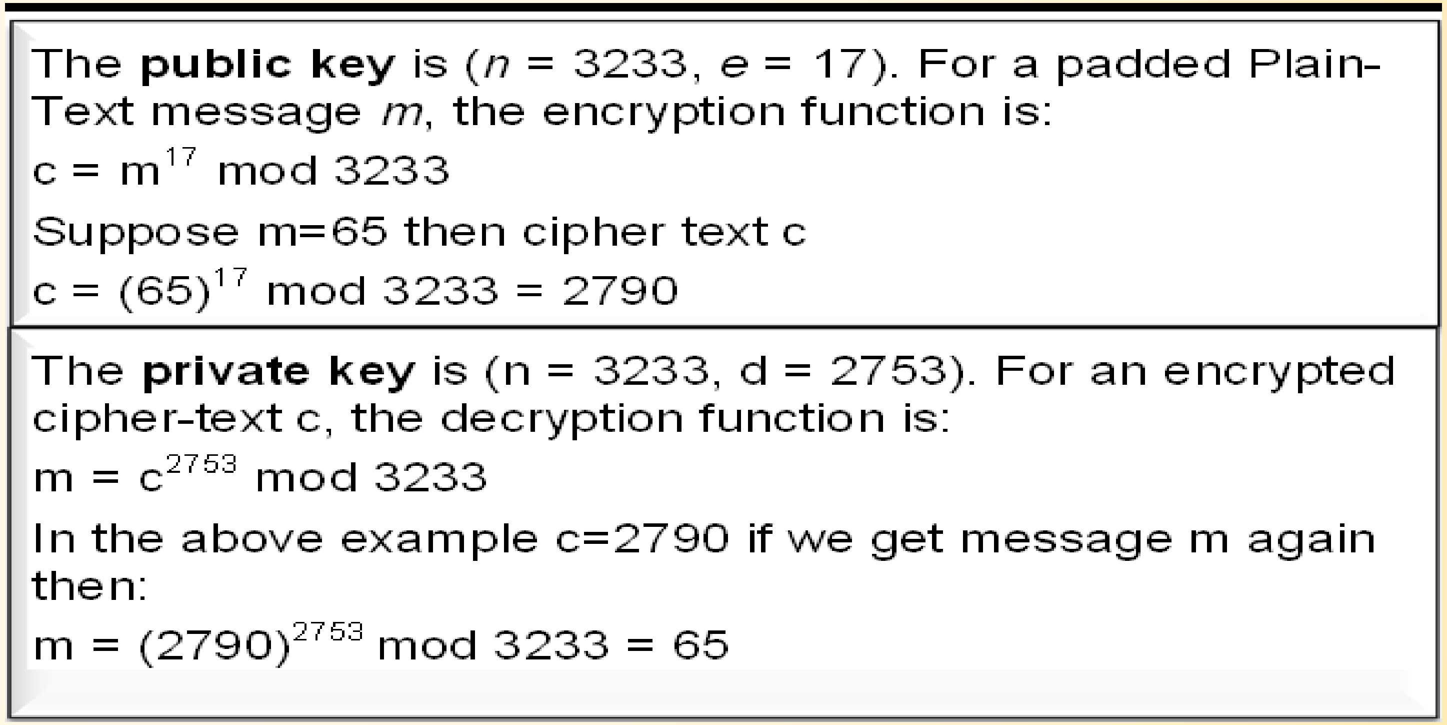
Solve de≡1 (mod Ø (n)) to get d=3

If the ciphertext C=5859, then the plaintext

$M \equiv C^d \equiv 5859^3 \equiv 1415 \pmod{11413}$



1. Choose two distinct prime numbers, such as
   P=61 and Q=53.
2. Compute n = p*q giving:
   n=61*53=3233
3. Compute φ(n) = (p − 1)(q − 1) giving:
   φ(3233) = (61 − 1)(53 − 1) = 60 * 52 = 3120
4. Choose any number 1 < e < 3120 that is co-prime to 3120
   Let e=17
5. Compute a value for d such that
   (d * e) % φ(n) = 1. One solution is:
   d=2753



The **public key** is (*n* = 3233, *e* = 17). For a padded Plain-Text message *m*, the encryption function is:

$c = m^{17} \bmod 3233$

Suppose m=65 then cipher text c

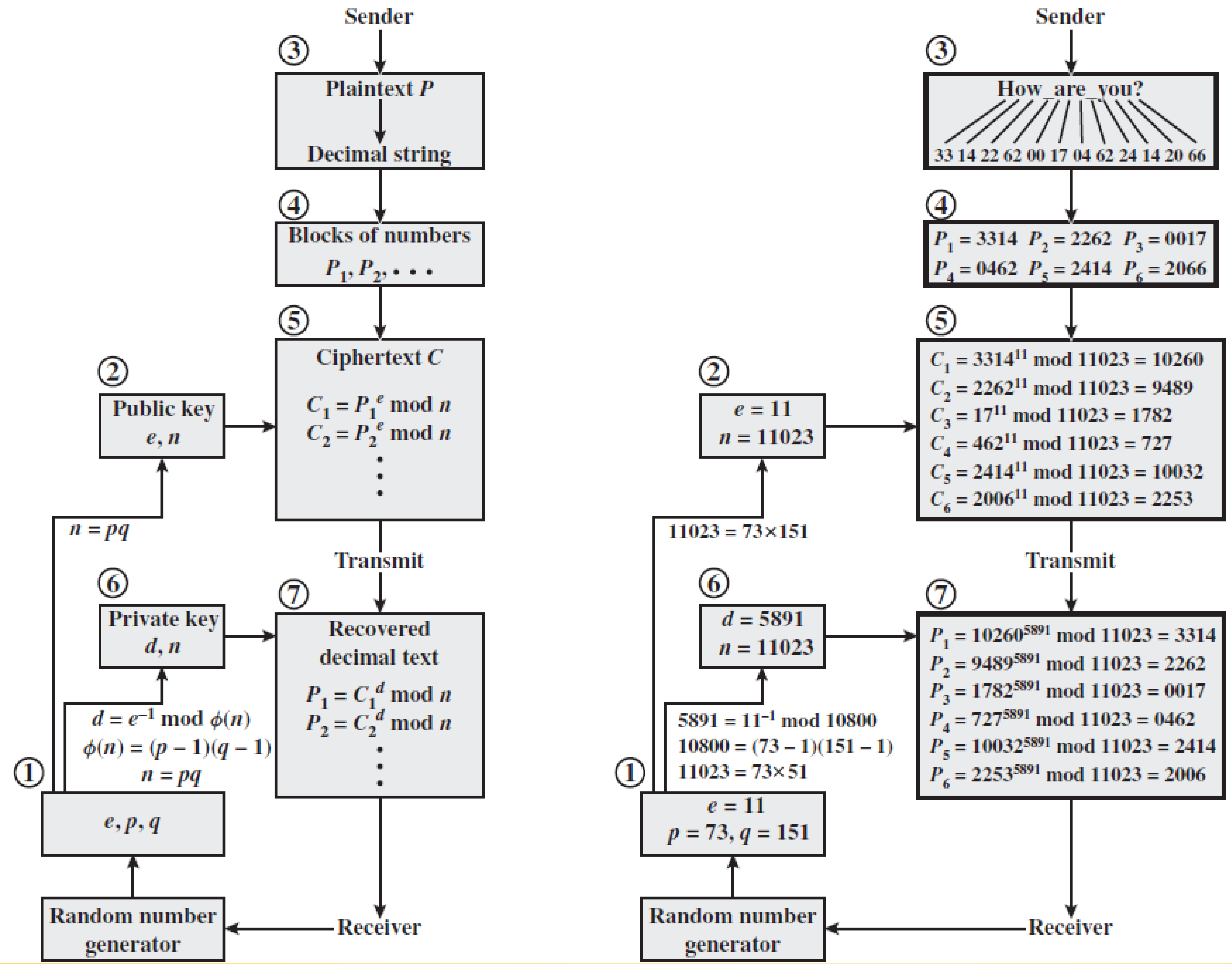
$c = (65)^{17} \bmod 3233 = 2790$

The **private key** is (n = 3233, d = 2753). For an encrypted cipher-text c, the decryption function is:

$m = c^{2753} \bmod 3233$

In the above example c=2790 if we get message m again then:

$m = (2790)^{2753} \bmod 3233 = 65$



Sender

③ Plaintext P

Decimal string

④ Blocks of numbers

$P_1, P_2, \ldots$

⑤ Ciphertext C

$C_1 = P_1^{\,e} \bmod n$

$C_2 = P_2^{\,e} \bmod n$

$\vdots$

② Public key

e, n

$n = pq$

Transmit

⑥ Private key

d, n

⑦ Recovered decimal text

$P_1 = C_1^{\,d} \bmod n$

$P_2 = C_2^{\,d} \bmod n$

$\vdots$

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

$\phi(n) = (p - 1)(q - 1)$

$n = pq$

① e, p, q

Random number generator

Receiver

Sender

③ How_are_you?

33 14 22 62 00 17 04 62 24 14 20 66

④ $P_1 = 3314$ $P_2 = 2262$ $P_3 = 0017$

$P_4 = 0462$ $P_5 = 2414$ $P_6 = 2066$

⑤ $C_1 = 3314^{11} \bmod 11023 = 10260$

$C_2 = 2262^{11} \bmod 11023 = 9489$

$C_3 = 17^{11} \bmod 11023 = 1782$

$C_4 = 462^{11} \bmod 11023 = 727$

$C_5 = 2414^{11} \bmod 11023 = 10032$

$C_6 = 2006^{11} \bmod 11023 = 2253$

② $e = 11$

$n = 11023$

$11023 = 73 \times 151$

Transmit

⑥ $d = 5891$

$n = 11023$

⑦ $P_1 = 10260^{5891} \bmod 11023 = 3314$

$P_2 = 9489^{5891} \bmod 11023 = 2262$

$P_3 = 1782^{5891} \bmod 11023 = 0017$

$P_4 = 727^{5891} \bmod 11023 = 0462$

$P_5 = 10032^{5891} \bmod 11023 = 2414$

$P_6 = 2253^{5891} \bmod 11023 = 2006$

$5891 = 11^{-1} \bmod 10800$

$10800 = (73 - 1)(151 - 1)$

$11023 = 73 \times 51$

① $e = 11$

$p = 73, q = 151$

Random number generator

Receiver

- Usage of RSA to process multiple blocks of data according to HELL79.

- The plaintext is an alphanumeric string. Each plaintext symbol is assigned a unique code of two decimal digits (e.g., a = 00, A = 26).

- A plaintext block consists of four decimal digits, or two alphanumeric characters.

- The circled numbers indicate the order in which operations are performed.

Security of RSA Algorithm

Four possible approaches to attacking the RSA algorithm are

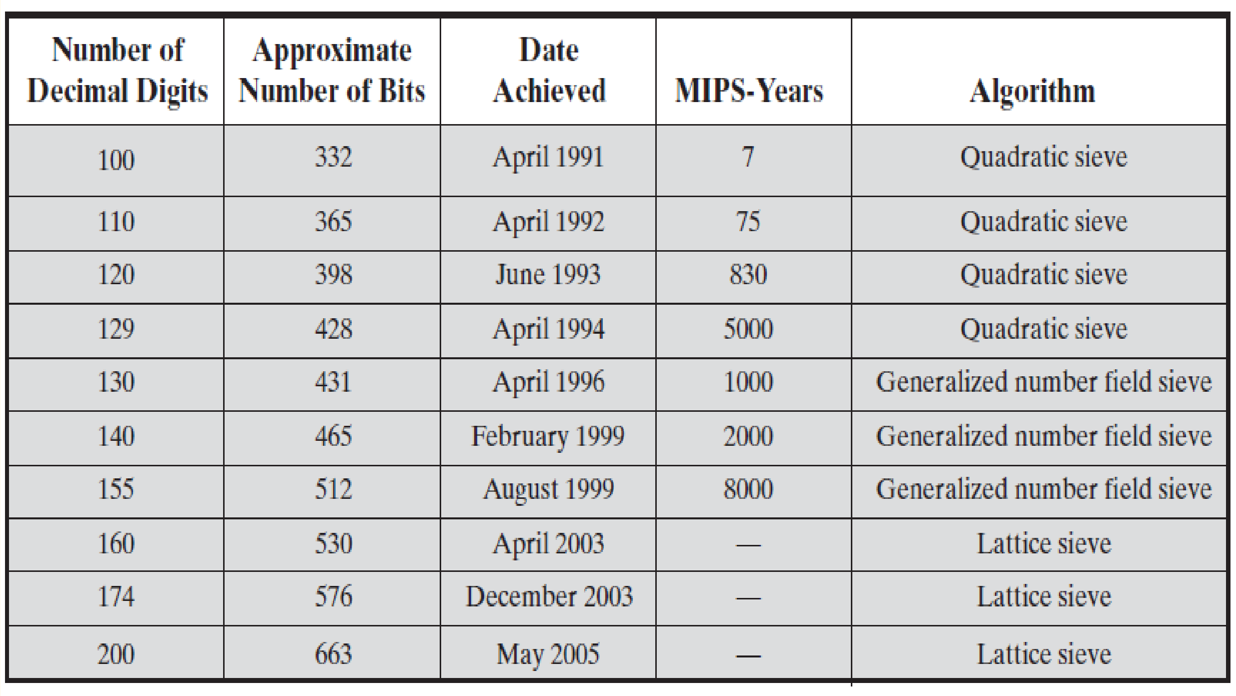
- **Brute force**: This involves trying all possible private keys.
- **Mathematical attacks**: There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks**: These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks**: This type of attack exploits properties of the RSA algorithm
- **Hardware fault-based Attack**: involves inducing hardware faults in the processor that is generating the digital signature.

- 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

Mathematical Attacks - The Factoring problem

- Three approaches of attacking RSA mathematically
  - factor n=p.q, hence compute ø(n) and then d
  - determine ø(n) directly and compute d
  - find d directly

The threat to larger key sizes is twofold: the continuing increase in computing power and the continuing refinement of factoring algorithms. For the near future, a key size in the range of 1024 to 2048 bits seems reasonable.



Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-Years	Algorithm
100	332	April 1991	7	Quadratic sieve
110	365	April 1992	75	Quadratic sieve
120	398	June 1993	830	Quadratic sieve
129	428	April 1994	5000	Quadratic sieve
130	431	April 1996	1000	Generalized number field sieve
140	465	February 1999	2000	Generalized number field sieve
155	512	August 1999	8000	Generalized number field sieve
160	530	April 2003	—	Lattice sieve
174	576	December 2003	—	Lattice sieve
200	663	May 2005	—	Lattice sieve

❖ 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 the order of magnitude 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.

The Timing Attacks

- **Paul Kocher**, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages [KOCH96, KALI96b]. This attack is alarming for two reasons: It comes from a completely unexpected direction, and it is a ciphertext-only attack.

- Although the timing attack is a serious threat, there are simple countermeasures that can be used, including the following.
  - **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 ATTACK AND OPTIMAL ASYMMETRIC ENCRYPTION PADDING

- The basic RSA algorithm is vulnerable to a **chosen ciphertext attack** (CCA). CCA is defined as an attack in which the adversary chooses a number of ciphertexts and is then given the corresponding plaintexts, decrypted with the target's private key.

- A simple example of a CCA against RSA takes advantage of the following property of RSA:

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

We can decrypt $C = M^e \bmod n$ using a CCA as follows.

1. Compute $X = (C \times 2^e) \bmod n$.

2. Submit X as a chosen ciphertext and receive back $Y = X^d \bmod n$.

But now note that

$X = (C \bmod n) \times (2^e \bmod n)$

$= (M^e \bmod n) \times (2^e \bmod n)$

$= (2M)^e \bmod n$

Therefore $Y = (2M) \bmod n$. From this, we can deduce M.

RSA Security Inc., 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).

The Diffie-Hellman Algorithm

- Diffie-Hellman key exchange is a simple public-key algorithm. This protocol enables two users to establish a secret key using a public-key scheme based on discrete logarithms.

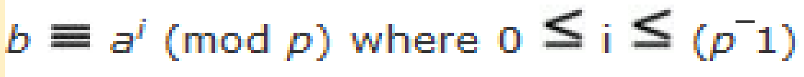
- The Diffie-Hellman key agreement protocol (1976) was the first practical method for establishing a shared secret over an un secured communication channel.

- **The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms**.

- First, a primitive root of a prime number p, can be defined as one whose powers generate all the integers from 1 to p-1. If a is a primitive root of the prime number p, then the numbers,

  a* mod *p*, *a*^{2} mod *p*,..., *a*$^{p-1}$ mod *p, are distinct and consist of the integers from 1 through p-1 in some permutation.

- For any integer ***b*** and a primitive root ***a*** of prime number ***p***, we can find a unique exponent **i** such that

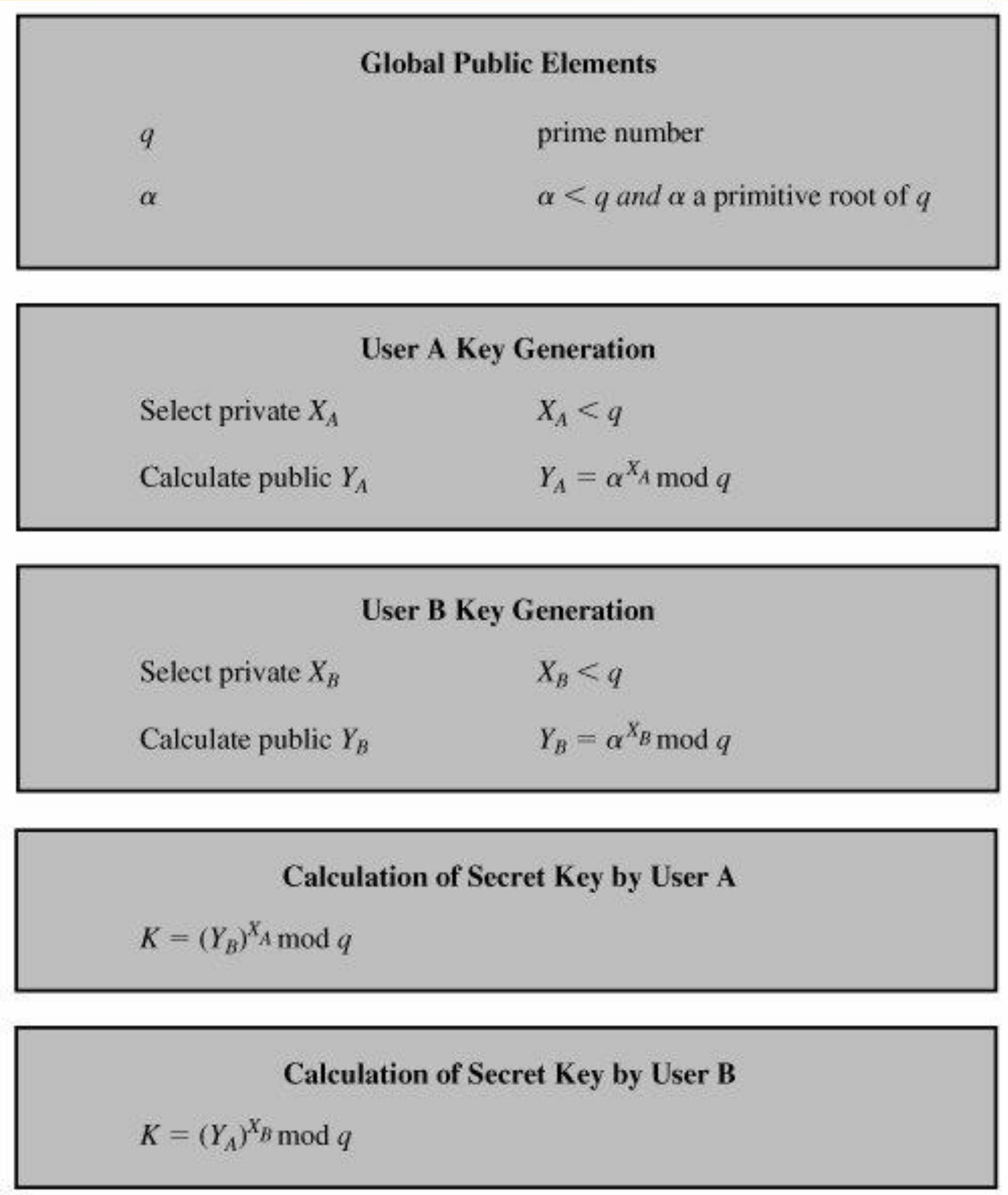
  

  $b \equiv a^{i} \pmod{p}$ where $0 \leq i \leq (p^{-}1)$

- The exponent i is referred to as the discrete logarithm of b for the base a, mod p.

- Examples: If p=7, then 3 is a primitive root for p because the powers of 3 are 1, 3, 2, 6, 4, 5---that is, every number mod 7 occurs except 0. But 2 isn't a primitive root because the powers of 2 are 1, 2, 4, 1, 2, 4, 1, 2, 4...missing several values.
- Example: If p=13, then 2 is a primitive root because the powers of 2 are 1, 2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7---which is all of the classes mod 13 except 0. There are other primitive roots for 13.
- Theorem: If p is a prime, then there is a primitive root for p.
- How to find a primitive root for p? Just try possibilities, starting with 2, 3, ..., until you find one that works.

Steps of D-H Algorithm



Global Public Elements

q	prime number
α	$\alpha < q$ *and* α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Calculation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Calculation of Secret Key by User B

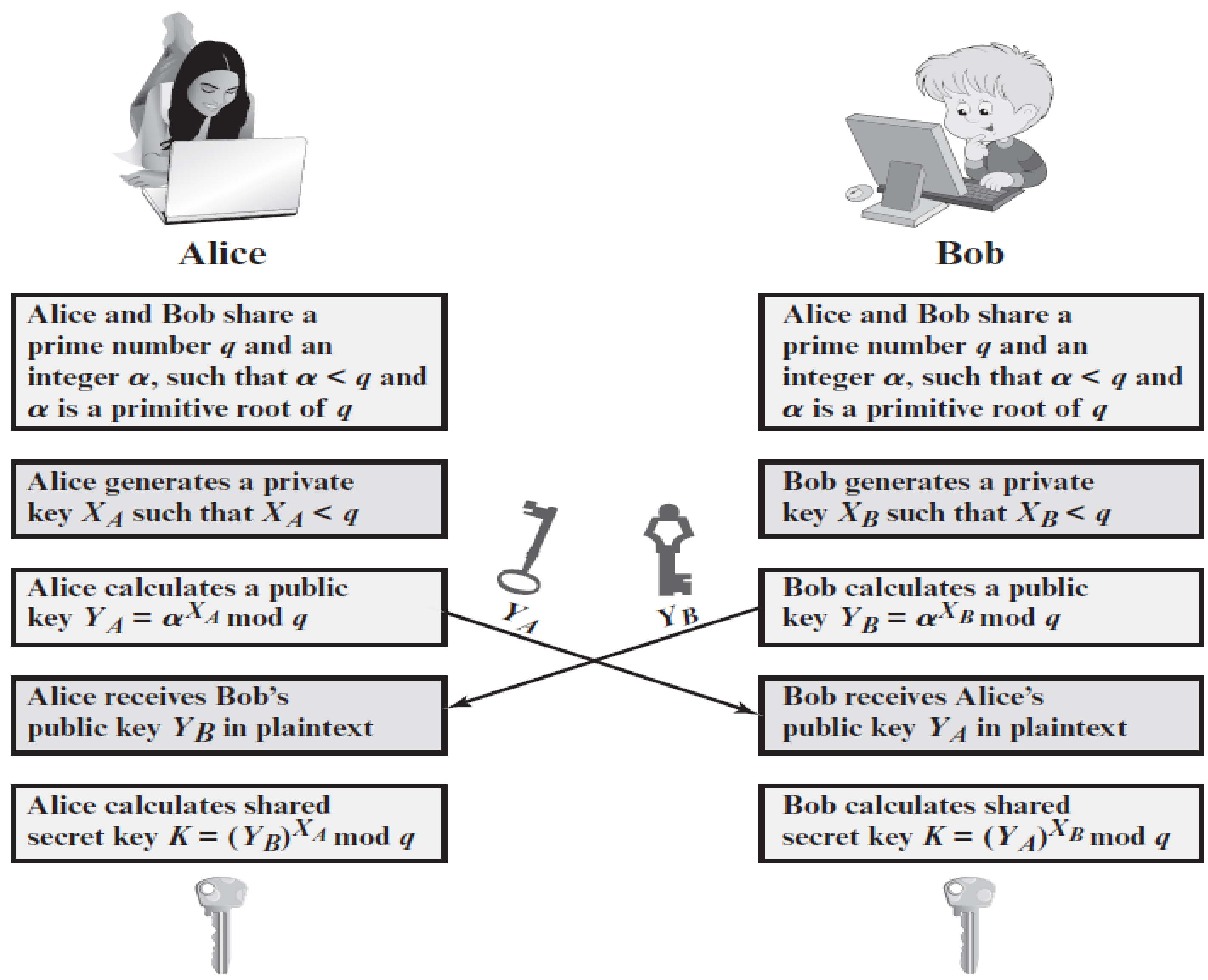
$K = (Y_A)^{X_B} \bmod q$

- For the D-H 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 integer $X_A < q$ and computes

$$Y_A = \alpha^{XA} \bmod q$$

- Similarly, user B independently selects a random integer $X_B < q$ and computes $Y_B = \alpha^{XB} \bmod q$
- 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

$$
\begin{aligned}
K &= (Y_B)^{X_A} \bmod q \\
&= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\
&= (\alpha^{X_B})^{X_A} \bmod q \\
&= \alpha^{X_B X_A} \bmod q \\
&= (\alpha^{X_A})^{X_B} \bmod q \\
&= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\
&= (Y_A)^{X_B} \bmod q
\end{aligned}
$$



Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q

Alice generates a private key X_A such that $X_A < q$

Alice calculates a public key $Y_A = \alpha^{X_A} \bmod q$

Alice receives Bob's public key Y_B in plaintext

Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$

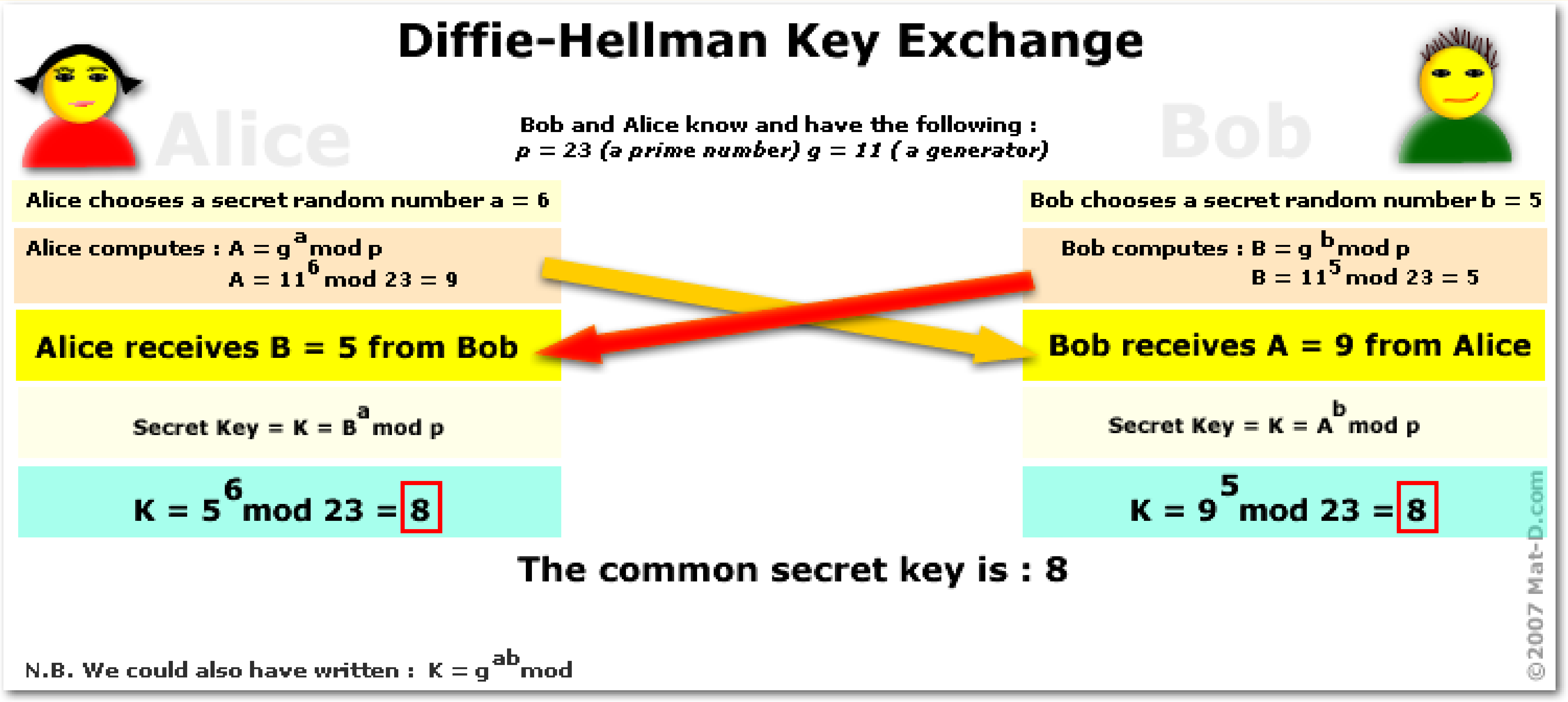
Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q

Bob generates a private key X_B such that $X_B < q$

Bob calculates a public key $Y_B = \alpha^{X_B} \bmod q$

Bob receives Alice's public key Y_A in plaintext

Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$



Diffie-Hellman Key Exchange

Bob and Alice know and have the following :
p = 23 (a prime number) g = 11 (a generator)

Alice chooses a secret random number a = 6

Alice computes : $A = g^{a} \bmod p$
$A = 11^{6} \bmod 23 = 9$

Alice receives B = 5 from Bob

Secret Key = K = $B^{a} \bmod p$

$K = 5^{6} \bmod 23 = \boxed{8}$

Bob chooses a secret random number b = 5

Bob computes : $B = g^{b} \bmod p$
$B = 11^{5} \bmod 23 = 5$

Bob receives A = 9 from Alice

Secret Key = K = $A^{b} \bmod p$

$K = 9^{5} \bmod 23 = \boxed{8}$

The common secret key is : 8

N.B. We could also have written : $K = g^{ab} \bmod$

Examples of D-H Algorithm

p = 23 and g = 5.
Alice chooses a = 6 and sends 5^6mod 23 = 8
Bob chooses b = 15 and sends 5^{15} mod 23 = 19.
Alice computes 19^6 mod 23 = 2
Bob computes 8^{15} mod 23 = 2.

Let p = 37 and g = 13.
Let Alice pick a = 10. Alice calculates 13^{10} (mod 37) which is 4 and sends that to Bob.
Let Bob pick b = 7. Bob calculates 13^7 (mod 37) which is 32 and sends that to Alice.
(Note: 6 and 7 are secret to Alice and Bob, respectively, but both 4 and 32 are known by all.)
Alice receives 32 and calculates 32^{10} (mod 37) which is 30, the secret key.
Bob receives 4 and calculates 4^7 (mod 37) which is 30, the same secret key.

Let p = 47 and g = 5.
Let Alice pick a = 18. Alice calculates 5^{18} (mod 47) which is 2 and sends that to Bob.
Let Bob pick b = 22. Bob calculates 5^{22} (mod 47) which is 28 and sends that to Alice.
Alice receives 28 and calculates 28^{18} (mod 47) which is 24, the secret key.
Bob receives 2 and calculates 2^{22} (mod 47) which is 24, the same secret key

Diffie-Hellman Example

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

Discrete Log Problem

- The (discrete) exponentiation problem is as follows: Given a base a, an exponent b and a modulus p, calculate c such that $a^b \equiv c \pmod{p}$ and $0 \leq c < p$.
- It turns out that this problem is fairly easy and can be calculated "quickly" using fast-exponentiation.
- The discrete log problem is the inverse problem:
- Given a base a, a result c ($0 \leq c < p$) and a modulus p, calculate the exponent b such that

$$a^b \equiv c \pmod{p}.$$

- It turns out that no one has found a quick way to solve this problem
- Given a = 2, b = 7 and p = 37, calculate c: $2^7 = 128$, and $128 \equiv 17 \pmod{37}$
- Given a = 2, c = 17, and p = 37, calculate b: Try each value of b until you find one that works! (For large prime numbers, this is too slow.)

- With DLP, if P had 300 digits, X_a and X_b have more than 100 digits, it would take longer than the life of the universe to crack the method.

Man-in-the-middle attack on D-H Algorithm

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

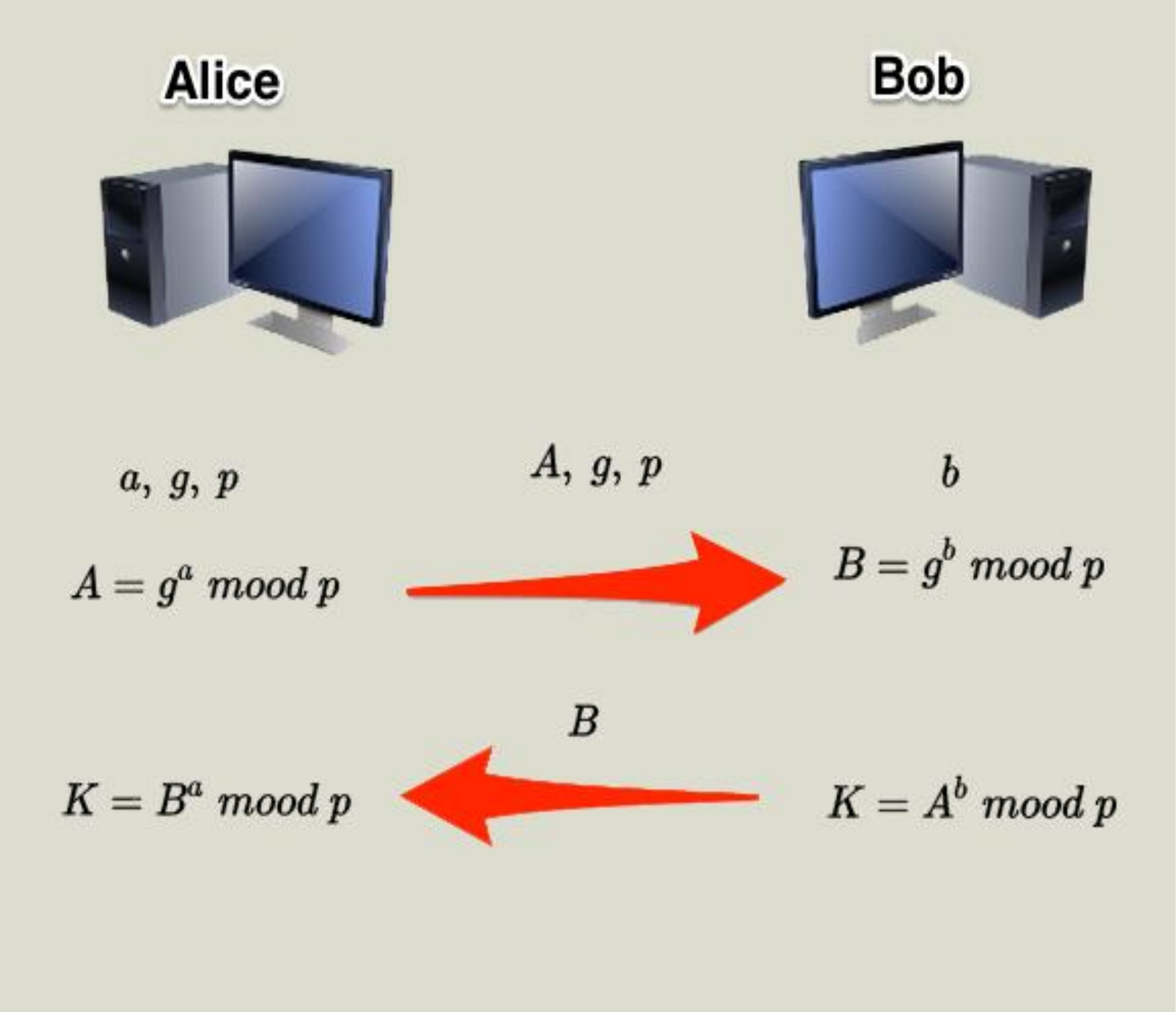
1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates K2 = $(Y_A)^{X_{D2}}$mod q.
4. Bob receives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

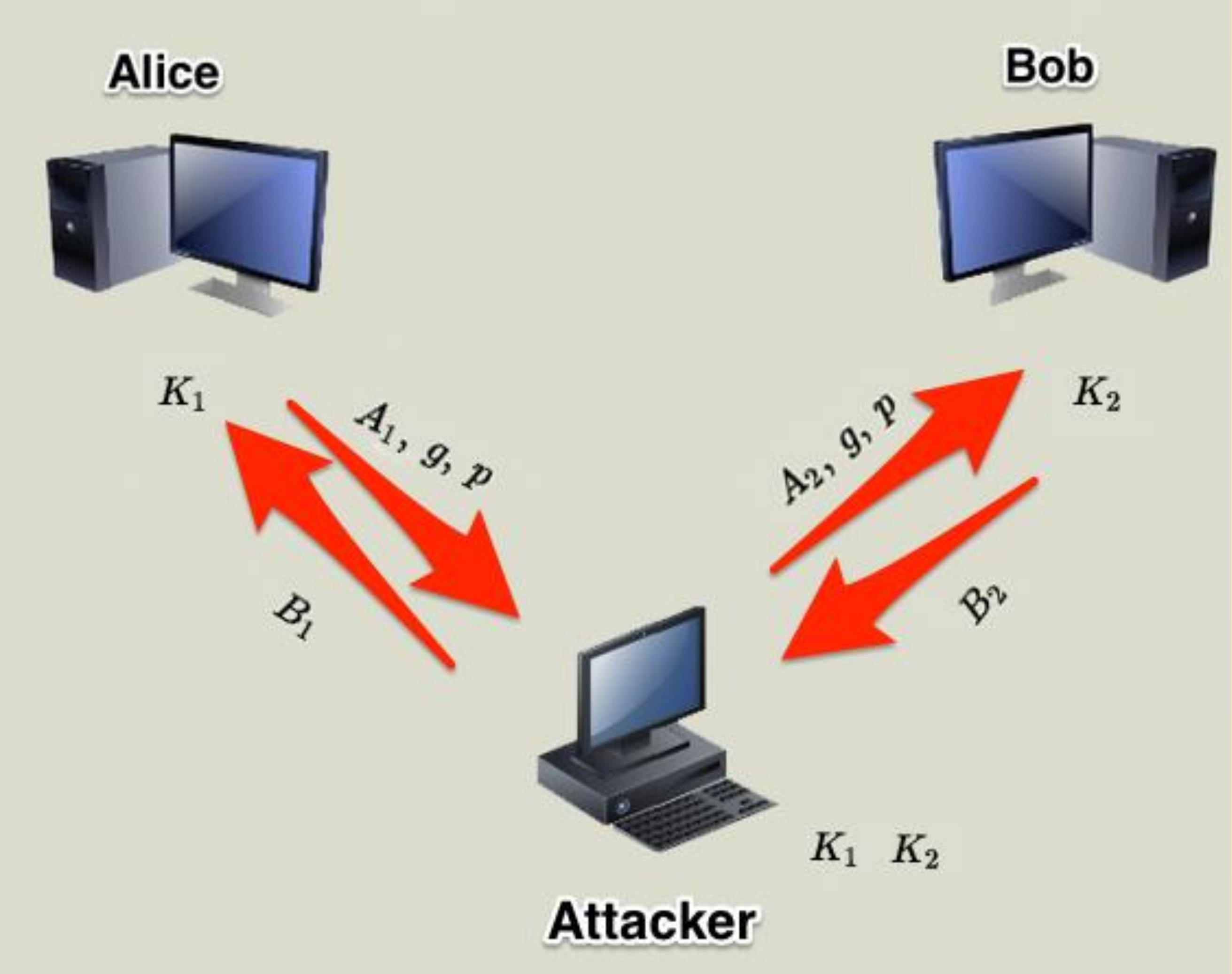
1. Alice sends an encrypted message M: E(K2, M).
2. Darth intercepts the encrypted message and decrypts it, to recover M.
3. Darth sends Bob E(K1, M) or E(K1, M'), where M' is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

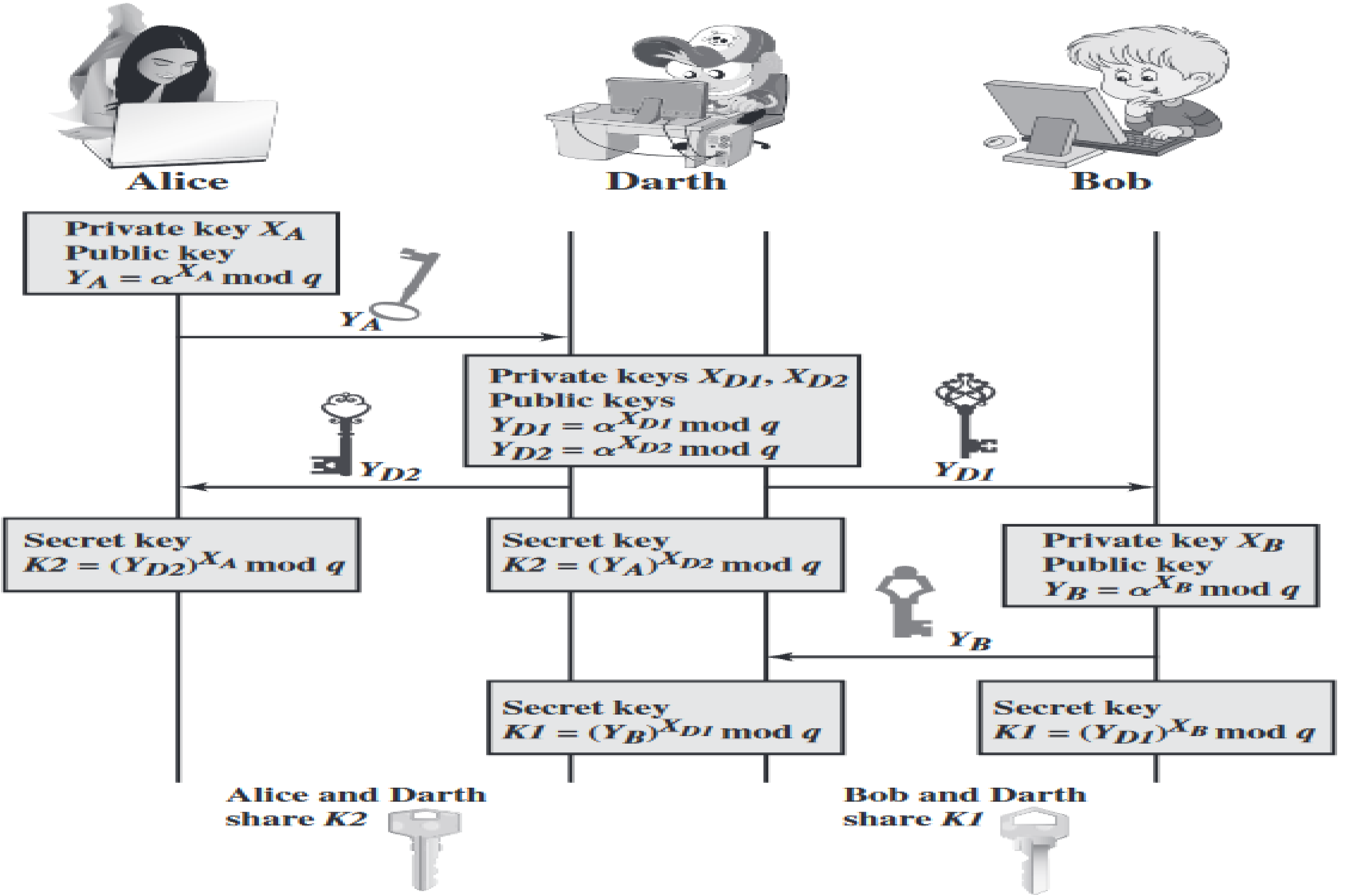
The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates.

This is how does Diffie-Hellman work:


Alice
Bob
a, g, p
A, g, p
b
$A = g^a \text{ mood } p$
$B = g^b \text{ mood } p$
B
$K = B^a \text{ mood } p$
$K = A^b \text{ mood } p$

And this is how does the man-in-the-middle attack work in Diffie-Hellman:





Alice

Darth

Bob

Private key X_A
Public key
$Y_A = \alpha^{X_A} \bmod q$

Y_A

Private keys X_{D1}, X_{D2}
Public keys
$Y_{D1} = \alpha^{X_{D1}} \bmod q$
$Y_{D2} = \alpha^{X_{D2}} \bmod q$

Y_{D2}

Y_{D1}

Secret key
$K2 = (Y_{D2})^{X_A} \bmod q$

Secret key
$K2 = (Y_A)^{X_{D2}} \bmod q$

Private key X_B
Public key
$Y_B = \alpha^{X_B} \bmod q$

Y_B

Secret key
$K1 = (Y_B)^{X_{D1}} \bmod q$

Secret key
$K1 = (Y_{D1})^{X_B} \bmod q$

Alice and Darth
share *K2*

Bob and Darth
share *K1*

Elliptic Curve Cryptography

- majority of public-key crypto (RSA, D-H) use either integer or polynomial arithmetic with very large numbers/polynomials
- imposes a significant load in storing and processing keys and messages
- Quantum computing will make RSA obsolete overnight
- an alternative is to use elliptic curves
- offers same security with smaller bit sizes

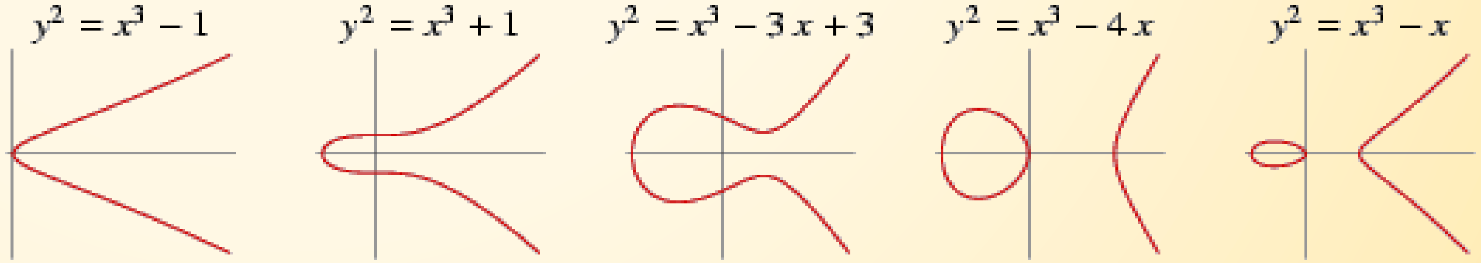
An *elliptic curve* is a curve of the form

$$y^2 = x^3 + ax + b$$

where $4a^3 + 27b^2 \neq 0$

Plus a point O at "infinity". It is at the end of all vertical lines.

Examples



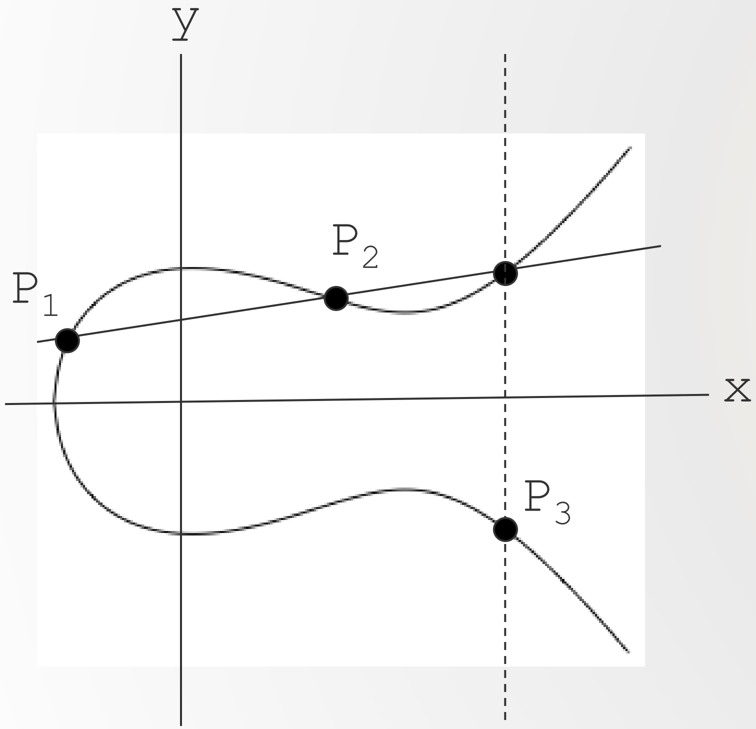
- Consider elliptic curve

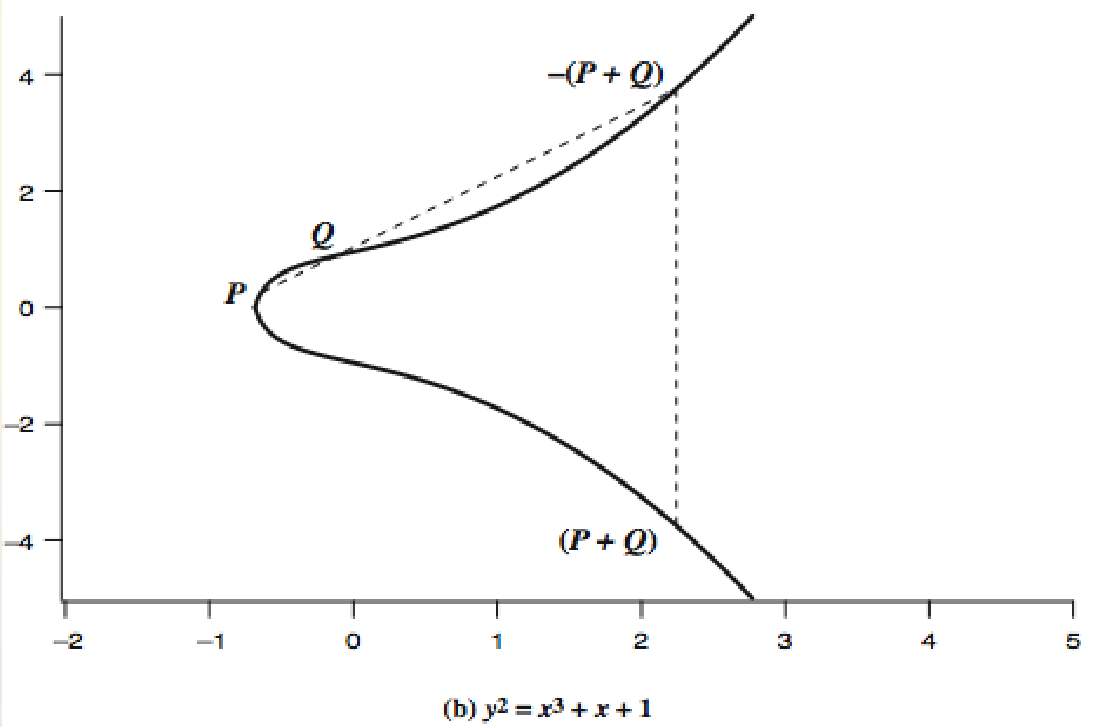
  $E: y^2 = x^3 - x + 1$

- If P_1 and P_2 are on E, we can define

  $\mathbf{P_3 = P_1 + P_2}$ as shown in picture

- Addition is all we need.
- geometrically sum of P+Q is reflection of the intersection R





(b) $y^2 = x^3 + x + 1$

- Elliptic curve cryptography uses curves whose variables & coefficients are finite
- have two families commonly used:
  - prime curves $E_p(a,b)$ defined over Z_p
    - use integers modulo a prime
    - best in software
  - binary curves $E_{2m}(a,b)$ defined over $GF(2^n)$
    - use polynomials with binary coefficients
    - best in hardware

Sum of two points

Define for two points ***P*** ***(x_1,y_1)*** and ***Q*** ***(x_2,y_2)*** in the Elliptic curve

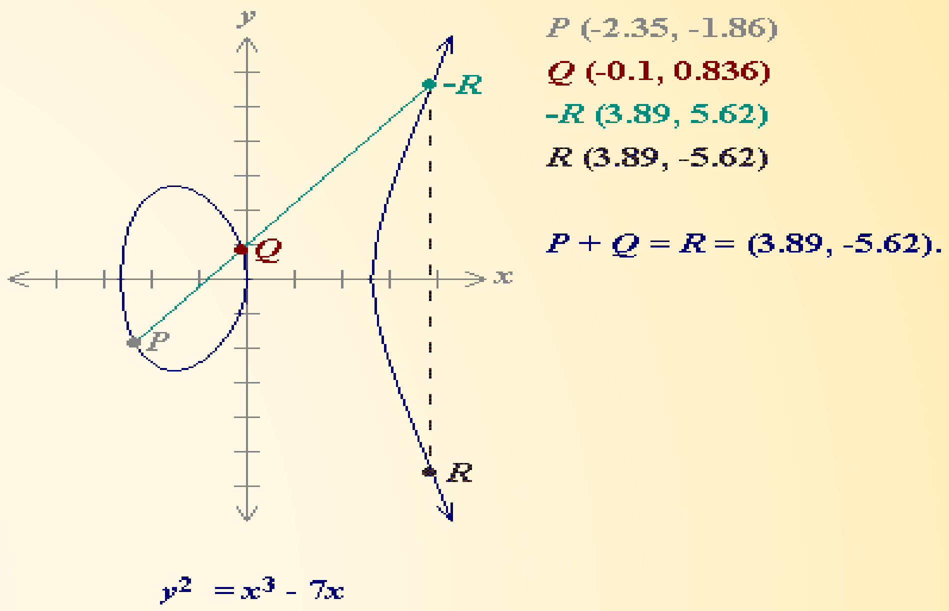
$$P = (x_1, y_1), Q = (x_2, y_2)$$

$$R = (P + Q) = (x_3, y_3)$$

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & for\_x_1 \neq x_2 \\ \dfrac{3x_1{}^2 + a}{2y_1} & for\_x_1 = x_2 \end{cases}$$

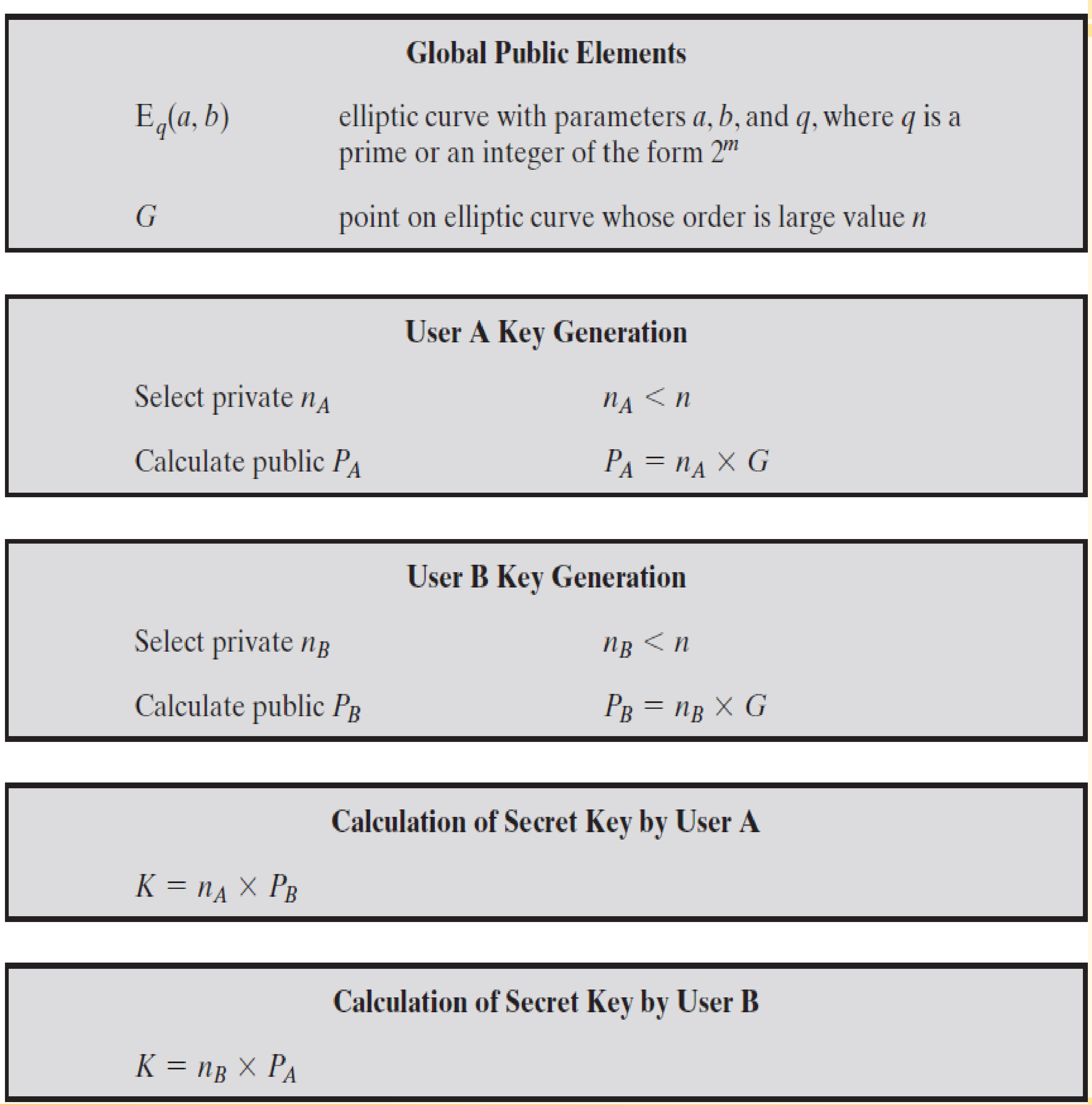
Then ***P+Q*** is given by ***R(x_3,y_3)*** :

$$\boxed{\begin{aligned} x_3 &= \lambda - x_1 - x_2 \\ y_3 &= \lambda(x_3 - x_1) + y_1 \end{aligned}}$$



Defining *mP*

- 2P = P + P
- 3P = P + P + P
- *m*P = P + P + … + P
- No matter how big *m* is, there is an efficient (quick) way to calculate *m*P.



Global Public Elements

$E_q(a, b)$	elliptic curve with parameters a, b, and q, where q is a prime or an integer of the form 2^m
G	point on elliptic curve whose order is large value n

User A Key Generation

Select private n_A	$n_A < n$
Calculate public P_A	$P_A = n_A \times G$

User B Key Generation

Select private n_B	$n_B < n$
Calculate public P_B	$P_B = n_B \times G$

Calculation of Secret Key by User A

$K = n_A \times P_B$

Calculation of Secret Key by User B

$K = n_B \times P_A$

- Begin with an elliptic curve mod p, let P be a point and let Q be a multiple of P. The **ECDLP** is to find the value of m such that Q = mP.

- We can simply calculate 2P, 3P, 4P, etc. But if p and m are large numbers, this could take trillions of years.

- Basically, we do not know of a fast way to solve ECDLP.

Elliptic Curve Encryption/Decryption

As with the key exchange system, an encryption/decryption system requires a point G and an elliptic group $E_q(a, b)$ as parameters. Each user A selects a private key n_A and generates a public key $P_A = n_A * G$.

To encrypt and send a message P_m to B, A chooses a random positive integer k and produces the ciphertext C_m consisting of the pair of points:

$$C_m = \{kG, P_m + kP_B\}$$

Note that A has used B's public key P_B. To decrypt the ciphertext, B multiplies the first point in the pair by B's private key and subtracts the result from the second point:

$$P_m + kP_B - n_B(kG) = P_m + k(n_BG) - n_B(kG) = P_m$$

Comparable Key Sizes for Equivalent Security

Symmetric scheme (key size in bits)	ECC-based scheme (size of *n* in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360