MODULE III

Public key cryptosystems – The RSA Algorithm – Diffie Hellman key exchange –comparison of RSA & DES – Elliptic Curve Cryptography – Number Theory Concepts

Private-Key Cryptography

- ➢ traditional **private/secret/single key** cryptography uses **one** key
- ➢ shared by both sender and receiver
- ➢ if this key is disclosed communications are compromised
- ➢ also is **symmetric**, parties are equal
- ➢ hence does not protect sender from receiver forging a message & claiming is sent by sender

Public-Key Cryptography

The development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography. From its earliest beginnings to modern times, virtually all cryptographic systems have been based on the elementary tools of substitution and permutation, and can be classed as private/secret/single key (symmetric) systems. All classical and modern block and stream ciphers are of this form.

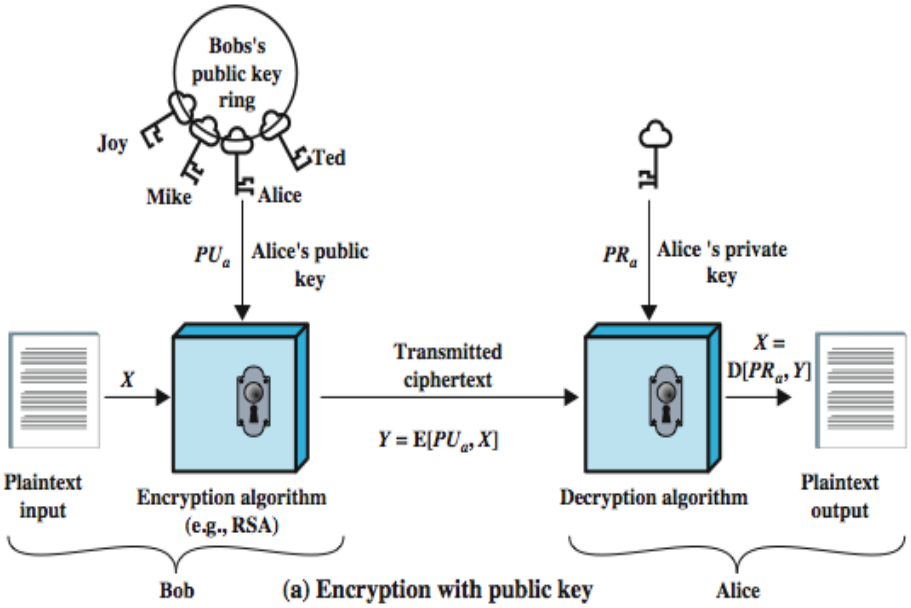
It is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, that uses only one key. Anyone knowing the public key can encrypt messages or verify signatures, but **cannot** decrypt messages or create signatures, counter-intuitive though this may seem. The use of two 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. Note that public key schemes are neither more nor less secure than private key (security depends on the key size for both), nor do they replace private key schemes (they are too slow to do so), rather they complement them. Both also have issues with key distribution, requiring the use of some suitable protocol.

- ➢ developed to address two key issues:

- **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

Public-Key Characteristics

- 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.



(a) Encryption with public key

A public-key encryption scheme has six ingredients:

• Plaintext: the readable message /data fed into the algorithm as input.

• Encryption algorithm: performs various transformations on the plaintext.

• Public and private keys: a pair of keys 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: 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: accepts the ciphertext and matching key and produces the original plaintext.

Symmetric versus Public-Key

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 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 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. 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

The public-key schemes can be used for either secrecy or authentication, or both. There is some source A that produces a message in plaintext X.

The M elements of X are letters in some finite alphabet. The message is intended for destination B.

B generates a related pair of keys: a public key, PUb, and a private key, PRb.

PRb is known only to B, whereas PUb is publicly available and therefore accessible by A.

With the message X and the encryption key PUb as input, A forms the ciphertext

$Y = E(PUb, X)$

The intended receiver, in possession of the matching private key, is able to invert the transformation:

$X = D(PRb, Y)$

An adversary, observing *Y* and having access to PUb, but not having access to PRb or X, must attempt to recover X and/or PRb. This provides confidentiality.

Can also use a public-key encryption to provide authentication:

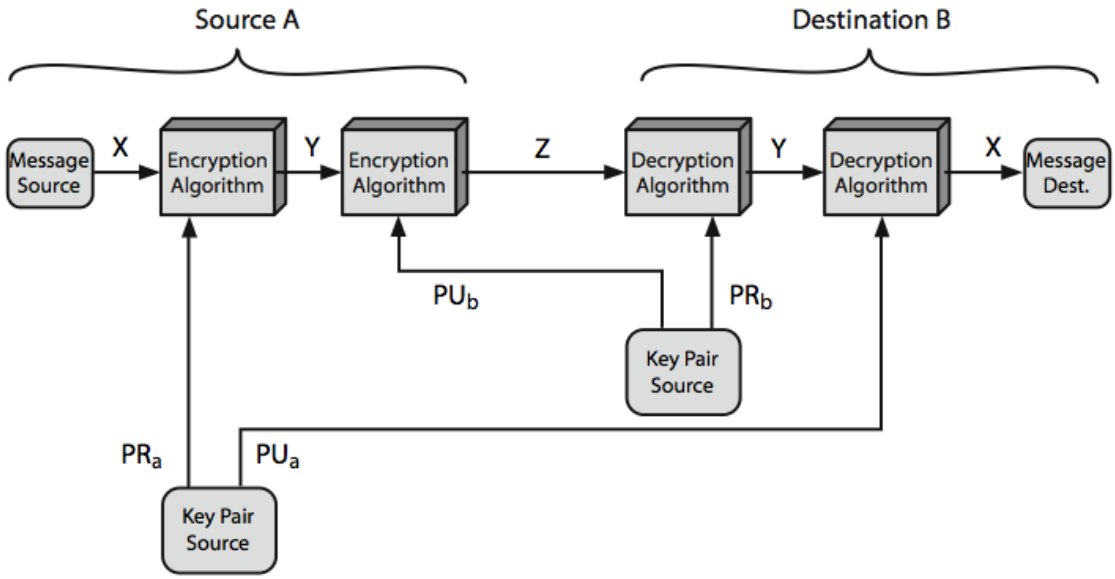
$Y = E(PRa, X); \ X = D(PUa, Y)$

To provide both the authentication function and confidentiality have a double use of the public-key scheme:

$Z = E(PUb, E(PRa, X)) \ X = D(PUa, D(PRb, Z))$

In this case, separate key pairs are used for each of these purposes. The receiver owns and creates secrecy keys, sender owns and creates authentication keys.

Authentication and Confidentiality



Public-Key Applications

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.

Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications.

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

Public-Key Requirements

1. It is computationally easy for a party B to generate a pair (public key *PUb*, private key PRb*)*.

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(PUb, 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(PRb, C) = D[PRb, E(PUb, M)

4. It is computationally infeasible for an adversary, knowing the public key, Pb, to determine the private key, PRb

5. It is computationally infeasible for an adversary, knowing the public key, Pb, and a ciphertext, C, to recover the original message, M.

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

   M = D[PU , E(PR, M)] = D[PR, E(PU, M)]

Security of Public Key Schemes

Public key schemes are no more or less secure than private key schemes - in both cases the size of the key determines the security.

- like private key schemes brute force **exhaustive search** attack is always theoretically possible
- but keys used are too large (>512bits)
- 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**
- hence is **slow** compared to private key schemes

RSA

RSA is the best known, and by far the most widely used general public key encryption algorithm, and was first published by Rivest, Shamir & Adleman of MIT in 1978. The Rivest-Shamir-Adleman (RSA) scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption. It is based on exponentiation in a finite (Galois) field over integers modulo a prime, using large integers (eg. 1024 bits). Its security is due to the cost of factoring large numbers.

RSA Encryption/decryption

The scheme makes use of an expression with exponentials. 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 $0 \leq M < n$

➢ to decrypt the ciphertext C the owner:

- uses their private key PR={d,n}

- computes: $M = C^d \bmod n$

RSA Key Setup

- The required modulus and exponent values are chosen during key setup. RSA key setup is done once (rarely) when a user establishes (or replaces) their public key, using the steps as shown. The exponent e is usually fairly small, just must be relatively prime to ø(n). Need to compute its inverse mod ø(n) to find d.
    - Each user generates a public/private key pair by:
- selecting two large primes at random: p, q
- computing their system modulus n=p.q
    - note ø(n)=(p-1)(q-1)
- selecting at random the encryption key e
    - where 1<e<ø(n), gcd(e,ø(n))=1
- solve following equation to find decryption key d
    - e.d=1 mod ø(n) and 0≤d≤n
- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,n}
- For this algorithm to be satisfactory for public-key encryption, it must be possible to find values of e, d, n such that $M^e d \bmod n = M$ for all M < n. It needs to find a relationship of the form $M^{ed} \bmod n = M$ The preceding relationship holds if *e and d* are multiplicative inverses moduloø *(n),* where ø *(n)* is the Euler totient function. This is a direct consequence of Euler's Theorem, so that raising a number to power e then d (or vica versa) results in the original number!
- because of Euler's Theorem:
    - $a^{ø(n)} \bmod n = 1$ where gcd(a,n)=1
- in RSA have:
    - n=p.q
    - ø(n)=(p-1)(q-1)
    - carefully chose e & d to be inverses mod ø(n)
    - hence e.d=1+k.ø(n) for some k

- hence :

$$C^d = M^{e.d} = M^{1+k.\phi(n)} = M^1.(M^{\phi(n)})^k$$

  - $= M^1.(1)^k = M^1 = M \bmod n$

RSA Example - Key Setup

1. Select primes: p=17 &q=11
2. Calculate $n = pq$=17 x 11=187
3. Calculate $\phi(n)=(p-1)(q-1)$=16x10=160
4. Select e:gcd(e,160)=1; choose e=7
5. Determine d: $de=1 \bmod 160$ and $d < 160$ Value is d=23 since 23x7=161= 10x160+1
6. Publish public key PU={7,187}
7. Keep secret private key PR={23,187}

sample RSA encryption/decryption is:

➢ given message M = 88 (nb. 88<187)

encryption:

$C = 88^7 \bmod 187 = 11$

decryption:

$M = 11^{23} \bmod 187 = 88$

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 probabalisticprimality 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.

RSA Security

➢ possible approaches to attacking RSA are:

- brute force key search - infeasible given size of numbers
- mathematical attacks - based on difficulty of computing ø(n), by factoring modulus n
- timing attacks - on running of decryption
- chosen ciphertext attacks - given properties of RSA

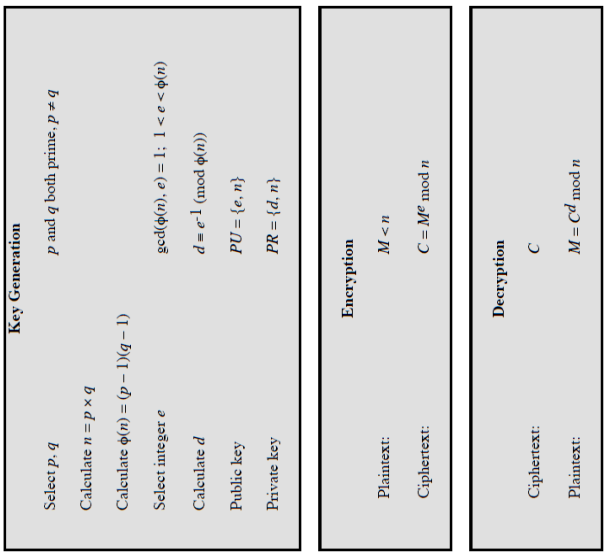


Figure 9.5 The RSA Algorithm

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. The algorithm itself is limited to the exchange of secret values. A number of commercial products employ this key exchange technique.

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. The algorithm itself is limited to the exchange of secret values, which depends on the value of the public/private keys of the participants. The Diffie-Hellman algorithm uses exponentiation in a finite (Galois) field (modulo a prime or a polynomial), and depends for its effectiveness on the difficulty of computing discrete logarithms.

In the Diffie-Hellman key exchange algorithm, there are two publicly known numbers: a prime number q and an integer α that is a primitive root of q. The prime q and primitive root α can be common to all using some instance of the D-H scheme. Note that the primitive root α is a number whose powers successively generate all the elements mod q. Users Alice and Bob choose random secrets x's, and then "protect" them using exponentiation to create their public y's. For an attacker monitoring the exchange of the y's to recover either of the x's, they'd need to solve the discrete logarithm problem, which is hard.

- ➢ all users agree on global parameters:
  - large prime integer or polynomial q
  - α being a primitive root mod q
- ➢ each user (eg. A/ B) generates their key
  - chooses a secret key (number): $x_A < q$
  - A compute their **public key**: $y_A = \alpha^{x}{}_A \bmod q$

B compute their **public key**:$Y_B = \alpha^{X}{}_B \bmod q$

– A now calculates the secret key K from his/her private key X_A and B's public key Y_B:

$K = Y_B^{X_A} \bmod q$ (1)

– B carries out a similar calculation for locally generating the shared secret key K from his/her private key X_B and A's public key Y_A:

$K = Y_A^{X_B} \bmod q$ (2)

The actual key exchange for either party consists of raising the others "public key' to power of their private key. The resulting number is used as the key for a block cipher or other private key scheme. For an attacker to obtain the same value they need at least one of the secret numbers, which means solving a discrete log, which is computationally infeasible given large enough numbers. Note that if Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys.

➢ shared session key for users A & B is K:

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

- Now two sides have exchanged a secret key.
- K is used as session key in private-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys
- attacker needs an x, must solve discrete log

Diffie-Hellman Key Exchange 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

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

Features

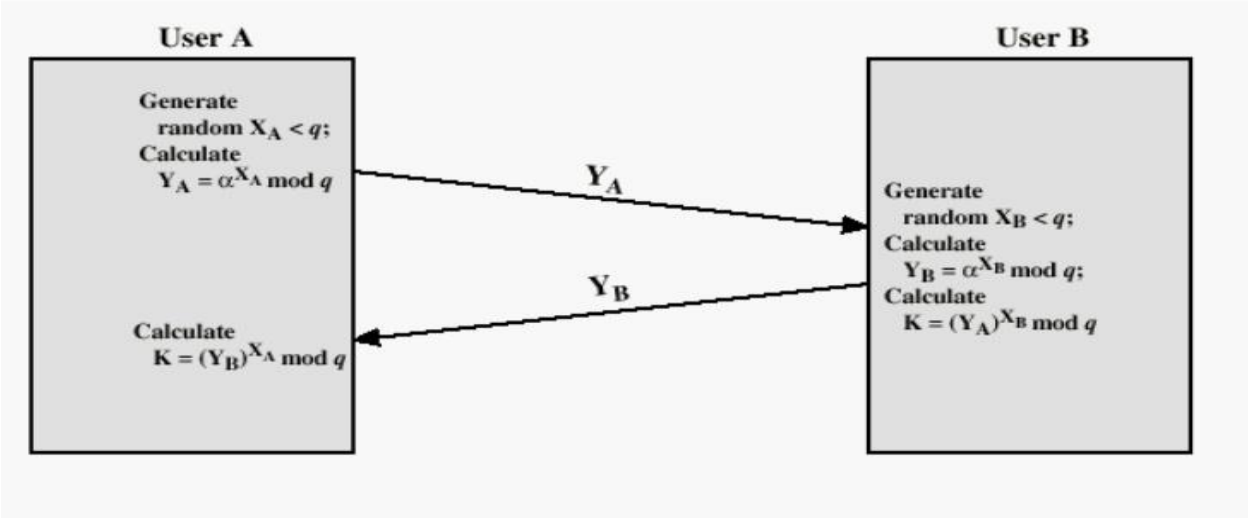
• In DH protocol an eavesdropper having access to the public keys for both A and would still not be able to figure out the secret key K.

• Another seemingly magical thing about this protocol is that it allows two parties A and B to create a shared secret K without either party having to send it directly to the other.

• The DH protocol is also referred to as the ephemeral secret keyagreement protocol because, typically, the secret key K is used only once.

- The security of the Diffie-Hellman algorithm is based on the fact that whereas it is relatively easy to compute the powers of an in teger in a finite field, it is extremely hard to compute the discrete logarithms.

Diffie-Hellman Key Exchange



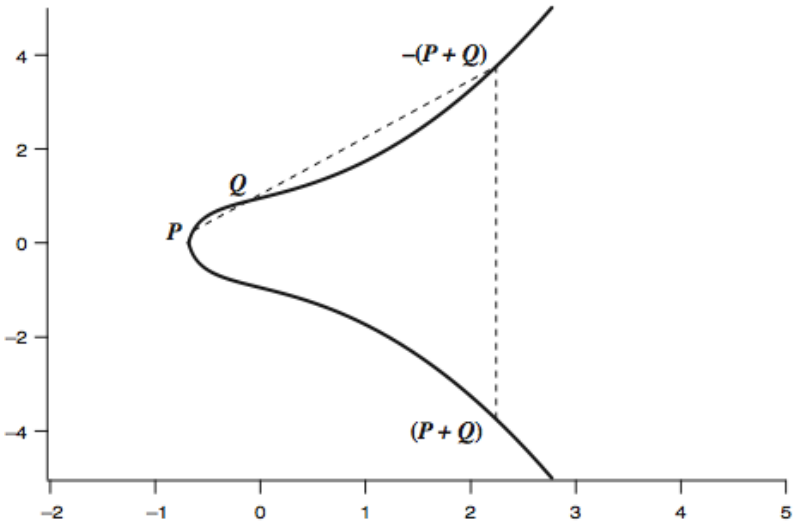
Diffie-Hellman Key Exchange

Elliptic Curve Cryptography

A major issue with the use of Public-Key Cryptography, is the size of numbers used, and hence keys being stored. Recently, an alternate approach has emerged, elliptic curve cryptography (ECC), which performs. It appears to offer equal security for a far smaller key size, thereby reducing processing overhead. The confidence level in ECC is not yet as high as that in RSA.

An elliptic curve is defined by an equation in two variables, with coefficients. For cryptography, the variables and coefficients are restricted to elements in a finite field, which results in the definition of a finite abelian group. Elliptic curve is a single element denoted *O* and called the *point at infinity* or the *zero point.* Now, consider the set of points E(*a, b*) consisting of all of the points *(x, y*) that satisfy this equation together with the element *O.* Using a different value of the pair *(a, b)* results in a different set *E(a, b).*

- ➢ an elliptic curve is defined by an equation in two variables x & y, with coefficients
- ➢ consider a cubic elliptic curve of form
  - $y^2 = x^3 + ax + b$
  - where x,y,a,b are all real numbers



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

"Example of Elliptic Curves", illustrates the geometric interpretation of elliptic curve addition, as follows: If three points on an elliptic curve lie on a straight line, their sum is *O*.

Elliptic curve cryptography makes use of elliptic curves in which the variables and coefficients are all restricted to elements of a finite field. Two families of elliptic curves are used in cryptographic applications:

prime curves $E_p(a,b)$ defined over Z_p (best for software use)

binary curves $E_{2m}(a,b)$ defined over $GF(2^n)$ (best for hardware use).

Elliptic curve Diffie-Hellman key exchange

Elliptic Curve Cryptography uses addition as an analog of modulo multiply, and repeated addition as an analog of modulo exponentiation.

- First pick a large integer *q*, which is either a prime number p or an integer of the form 2^m and elliptic curve parameters a and b
- select a suitable curve $E_q(a,b)$
- select base point $G=(x_1,y_1)$
    - with large order n s.t.nG=O
- A & B select private keys $n_A<n$, $n_B<n$
- compute public keys: $P_A=n_A*G$, $P_B=n_B*G$
- compute shared key: $K=n_A*P_B$,$K=n_B*P_A$
    - same since $K=n_A*n_B*G$
- attacker would need to find k which is not easy.

ECC Encryption/Decryption

The method is an analog of the ElGamal public-key encryption algorithm. The sender must first encode any message M as a point on the elliptic curve P_m.

- select suitable curve & point G as in D-H

- each user chooses private key $n_A<n$
- and computes public key $P_A=n_AG$
- to encrypt P_m : $C_m=\{kG, P_m+kP_b\}$, where k is a random positive integer
- decrypt C_m compute:

$P_m+kP_b–n_B(kG) = P_m+k(n_BG)–n_B(kG) = P_m$

The sender masks the message using random k, but also sends along a "clue" allowing the receiver who know the private-key to recover k and hence the message. For an attacker to recover the message, the attacker would have to compute k given G and kG, which is assumed hard.

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	$n_A < n$
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
$K = n_B \times P_A$

ECC Key Exchange

Security of ECC

- The security of ECC depends on how difficult it is to determine k given kP and P. This is referred to as the elliptic curve logarithm problem.

• ECC depends on the difficulty of the large number discrete logarithm calculation. This is referred to as the Elliptic Curve Discrete Logarithm Problem (ECDLP).

• It was shown by Menezes, Okamoto, and Vanstone (MOV) in 1993 that (for supersingular elliptic curves) the problem of solving the ECDLP problem (where the domain is the group Eq(a, b)) can be reduced to the much easier problem of finding logarithms in a finite field. There has been much work recently on extending the MOV reduction to general elliptic curves.

• When using GF(2n) finite fields, is faced by the Weil descent attack. To not be vulnerable to this attack, n must be a prime.

• Elliptic curves for which the total number of points on the curve equals the number of elements in the underlying finite field are also considered cryptographically weak.

Number Theory Concepts

Introduction to Number Theory

Prime Numbers

- prime numbers only have divisors of 1 and self
  - they cannot be written as a product of other numbers
  - note: 1 is prime, but is generally not of interest
- eg. 2,3,5,7 are prime, 4,6,8,9,10 are not
- prime numbers are central to number theory
- list of prime number less than 200 is:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199

Prime Factorization

- To factor a number *n* is to write it as a product of other numbers.
- $n = a * b * c$
- Or, 100 = 5 * 5 * 2 * 2
- Prime factorization of a number *n* is writing it as a product of prime numbers.
- 143 = 11 * 13

Relatively Prime Numbers & GCD

- two numbers a, b are relatively prime if have no common divisors apart from 1
  - eg. 8 & 15 are relatively prime since factors of 8 are 1,2,4,8 and of 15 are 1,3,5,15 and 1 is the only common factor

- conversely can determine the greatest common divisor by comparing their prime factorizations and using least powers
  - eg. $300=2^1x3^1x5^2$ $18=2^1x3^2$ hence $GCD(18,300)=2^1x3^1x5^0=6$

Fermat's Little Theorem

- If *p* is prime and *a* is an integer not divisible by *p*, then . . .
- $a^{p-1} 1 \pmod{p}$.
- And for every integer *a*
- $a^p \quad a \pmod{p}$.
- This theorem is useful in public key (RSA) and primality testing.

Euler Totient Function ø(n)

- when doing arithmetic modulo n
- complete set of residues is: 0..n-1
- reduced set of residues is those numbers (residues) which are relatively prime to n
  - eg for n=10,
  - complete set of residues is {0,1,2,3,4,5,6,7,8,9}
  - reduced set of residues is {1,3,7,9}
- number of elements in reduced set of residues is called the Euler Totient Function ø(n)
- to compute ø(n) need to count number of residues to be excluded
- in general need prime factorization, but
  - for p (p prime) ø(p) = p-1
  - for p.q (p,q prime) ø(pq) =(p-1)x(q-1)
- eg.

ø(37) = 36
ø(21) = (3–1)x(7–1) = 2x6 = 12

Euler's Theorem

- a generalisation of Fermat's Theorem
- $a^{ø(n)} = 1 \pmod{n}$
  - for any a,n where gcd(a,n)=1
- eg.

a=3;n=10; ø(10)=4;

hence $3^4 = 81 = 1 \bmod 10$

a=2;n=11; ø(11)=10;

hence $2^{10} = 1024 = 1 \bmod 11$