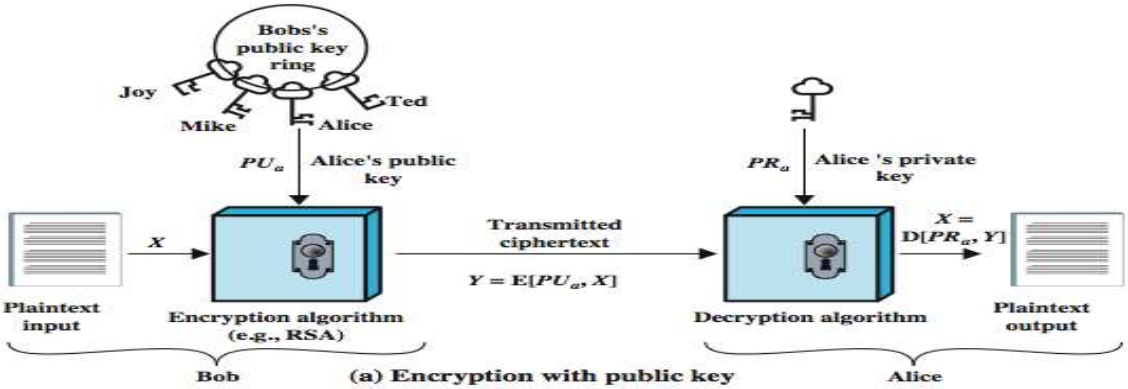
Module-3

Private-Key Cryptography

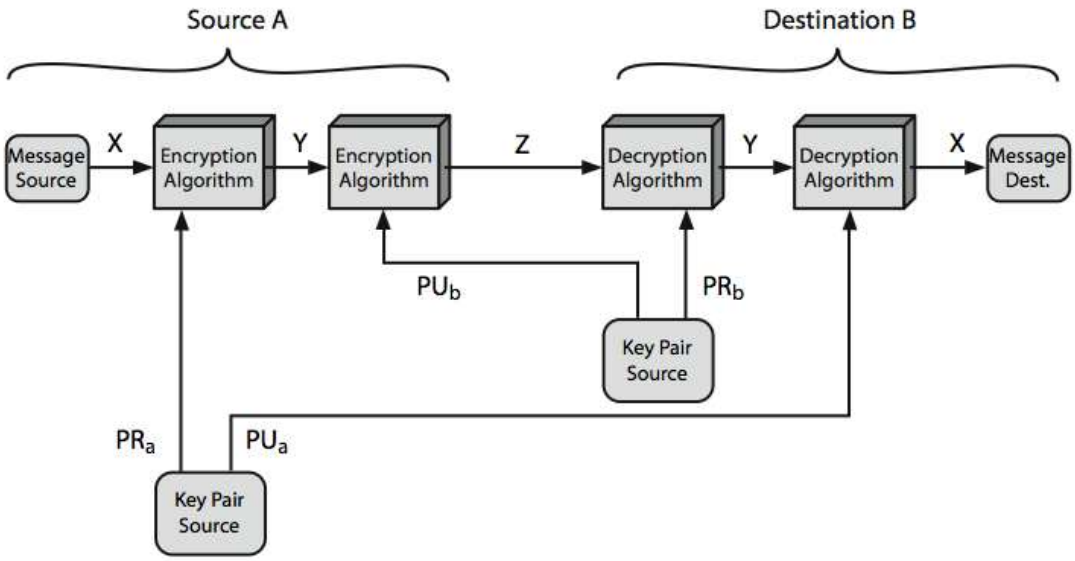
- traditional **private/secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- if this key is disclosed communications are compromised
- also is **symmetric**, parties are equal
- hence does not protect sender from receiver forging a message & claiming is sent by sender
- probably most significant advance in the 3000 year history of cryptography
- uses **two** keys – a public & a private key
- **asymmetric** since parties are **not** equal
- uses clever application of number theoretic concepts to function
- complements **rather than** replaces private key crypto



Symmetric vs 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.

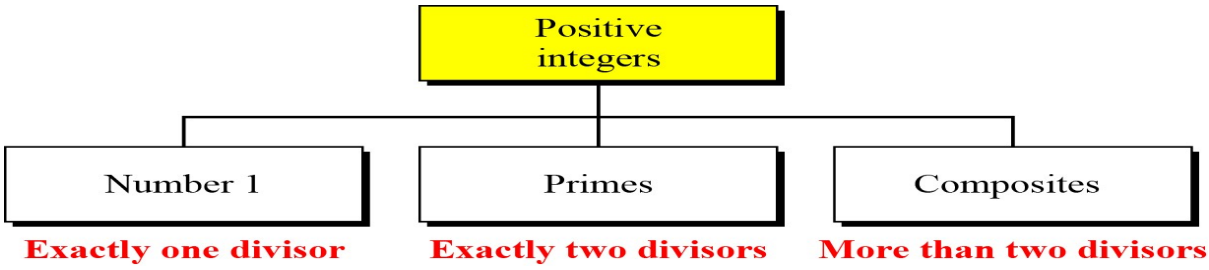
Public-Key Cryptosystems



Public-Key Applications

- can classify uses into 3 categories:
  - encryption/decryption (provide secrecy)
  - digital signatures (provide authentication)
  - key exchange (of session keys)

PRIMES



Euler's Theorem

In theory, Let $\phi(n)$ be Euler's totient function. If n is a positive integer, $\phi(n)$ is the number of integers in the range $\{1, 2, 3 \cdots, n\}$ which are relatively prime to n. If a is an integer and m is a positive integer relatively prime to a,

Then $a^{\phi(m)} \equiv 1 \pmod{m}$.

Proof

Consider the set of numbers $A = \{n_1, n_2, ...n_{\phi(m)}\} \pmod m$ such that the elements of the set are the numbers relatively prime to m. It will now be proved that this set is the same as the set $B = \{an_1, an_2, ...an_{\phi(m)}\} \pmod m$ where $(a, m) = 1$. All elements of B are relatively prime to m so if all elements of B are distinct, then B has the same elements as A. In other words, each element of B is congruent to one of A.This means that $n_1n_2...n_{\phi(m)} \equiv an_1 \cdot an_2...an_{\phi(m)} \pmod m$ $\rightarrow a^{\phi(m)} \cdot (n_1n_2...n_{\phi(m)}) \equiv n_1n_2...n_{\phi(m)} \pmod m \rightarrow a^{\phi(m)} \equiv 1 \pmod m$ as desired. Note that dividing by $n_1n_2...n_{\phi(m)}$ is allowed since it is relatively prime to m.

Euler's Phi-Function

Euler's phi-function, Φ (n), which is sometimes called the Euler's totient function plays a very important role in cryptography.

1. $\phi(1) = 0$.
2. $\phi(p) = p - 1$ if p is a prime.
3. $\phi(m \times n) = \phi(m) \times \phi(n)$ if m and n are relatively prime.
4. $\phi(p^e) = p^e - p^{e-1}$ if p is a prime.

We can combine the above four rules to find the value of f(n). For example, if n can be factored as

$$n = p_1^{e1} \times p_2^{e2} \times \ldots \times p_k^{ek}$$

then we combine the third and the fourth rule to find

$$\phi(n) = (p_1^{e_1} - p_1^{e_1-1}) \times (p_2^{e_2} - p_2^{e_2-1}) \times \cdots \times (p_k^{e_k} - p_k^{e_k-1})$$

Fermat's Little Theorem

Let *p* be a prime which does not divide the integer *a*, then $a^{p-1} \equiv 1 \pmod p$.

Proof.

Start by listing the first p-1 positive multiples of a:

$a, 2a, 3a, \ldots (p-1)a$

Suppose that ra and sa are the same modulo p, then we have $r = s \pmod p$, so the p-1 multiples of a above are distinct and nonzero; that is, they must be congruent to 1, 2, 3, ..., p-1 in some order. Multiply all these congruences together and we find

$a\,(2a)\,(3a) \ldots ((p-1)a) \equiv 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (p-1) \pmod p$

which is, $a^{(p-1)}(p-1)! \equiv (p-1)! \pmod p$. Divide both side by $(p-1)!$ to complete the proof.

Sometimes Fermat's Little Theorem is presented in the following form:

Corollary.

Let p be a prime and a any integer, then $a^p \equiv a \pmod{p}$.

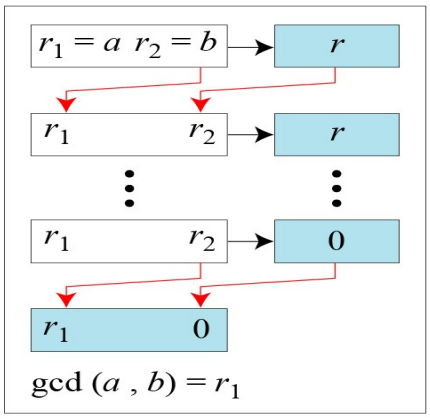
Proof.

The result is trival (both sides are zero) if p divides a. If p does not divide a, then we need only multiply the congruence in Fermat's Little Theorem by a to complete the proof.

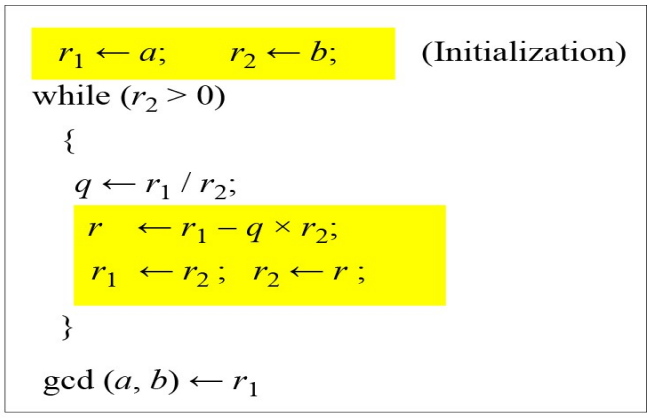
Greatest Common Divisor

The greatest common divisor of two positive integers is the largest integer that can divide both integers. When gcd (a, b) = 1, we say that a and b are relatively prime.

Euclidean Algorithm



a. Process



b. Algorithm

Find the greatest common divisor of 2740 and 1760

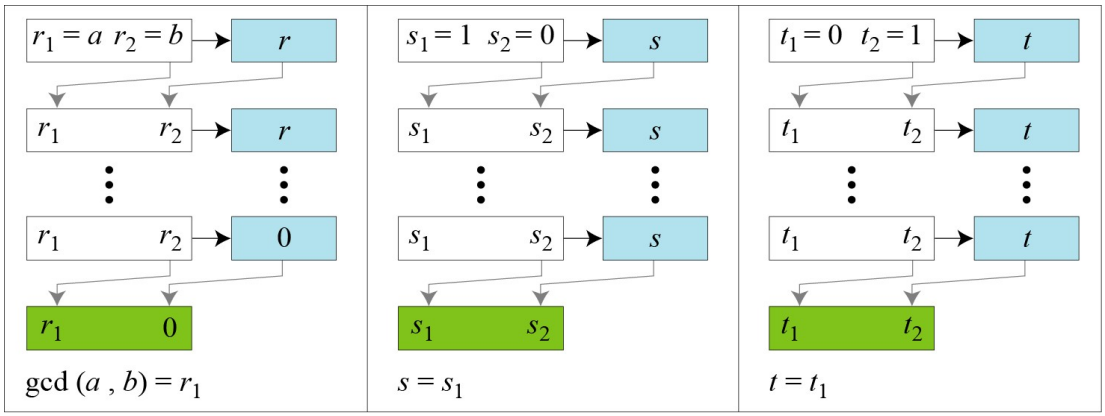
q	r_1	r_2	r
1	2740	1760	980
1	1760	980	780
1	980	780	200
3	780	200	180
1	200	180	20
9	180	20	0
	20	0	

Extended Euclidean Algorithm

Given two integers a and b, we often need to find other two integers, s and t, such that

$$s \times a + t \times b = \gcd(a, b)$$

The extended Euclidean algorithm can calculate the gcd (a, b) and at the same time calculate the value of s and t.



a. Process

$r_1 \leftarrow a;$ $r_2 \leftarrow b;$
$s_1 \leftarrow 1;$ $s_2 \leftarrow 0;$ (Initialization)
$t_1 \leftarrow 0;$ $t_2 \leftarrow 1;$

while $(r_2 > 0)$

{

$q \leftarrow r_1 / r_2;$

$r \leftarrow r_1 - q \times r_2;$
$r_1 \leftarrow r_2;$ $r_2 \leftarrow r;$ (Updating r's)

$s \leftarrow s_1 - q \times s_2;$
$s_1 \leftarrow s_2;$ $s_2 \leftarrow s;$ (Updating s's)

$t \leftarrow t_1 - q \times t_2;$
$t_1 \leftarrow t_2;$ $t_2 \leftarrow t;$ (Updating t's)

}

gcd $(a, b) \leftarrow r_1;$ $s \leftarrow s_1;$ $t \leftarrow t_1$

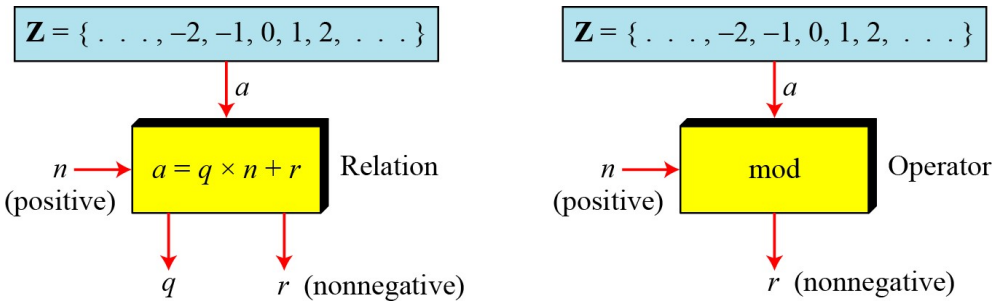
b. Algorithm

Eg. Given a = 161 and b = 28, find gcd (a, b) and the values of s and t

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
5	161	28	21	1	0	1	0	1	−5
1	28	21	7	0	1	−1	1	−5	6
3	21	7	0	1	−1	4	−5	6	−23
	7	0		**−1**	4		**6**	−23	

Modulo Operator

The modulo operator is shown as mod. The second input(n) is called the modulus. The output r is called the residue.



Find the result of the following operations:

a. 27 mod 5

b. 36 mod 12

c. −18 mod 14

d. −7 mod 10

Solution:

a. Dividing 27 by 5 results in r = 2

b. Dividing 36 by 12 results in r = 0.

c. Dividing −18 by 14 results in r = −4. After adding the modulus r = 10

d. Dividing −7 by 10 results in r = −7. After adding the modulus to −7, r = 3.

Properties:

$$\text{First Property:} \quad (a+b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$$

$$\text{Second Property:} \quad (a-b) \bmod n = [(a \bmod n) - (b \bmod n)] \bmod n$$

$$\text{Third Property:} \quad (a \times b) \bmod n = [(a \bmod n) \times (b \bmod n)] \bmod n$$

Inverses

$$a + b \equiv 0 \pmod{n}$$

In modular arithmetic, each integer has an additive inverse. The sum of an integer and its additive inverse is congruent to 0 modulo n.

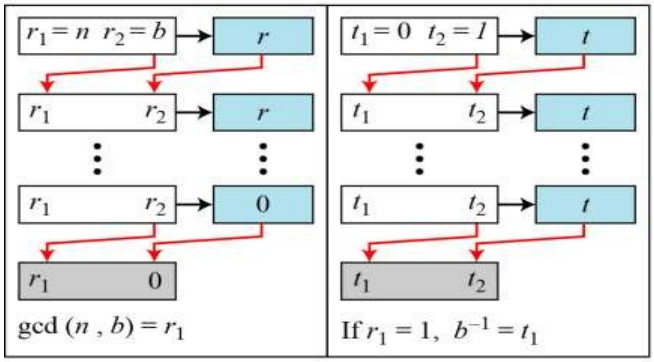
Multiplicative Inverse

$$a \times b \equiv 1 \pmod{n}$$

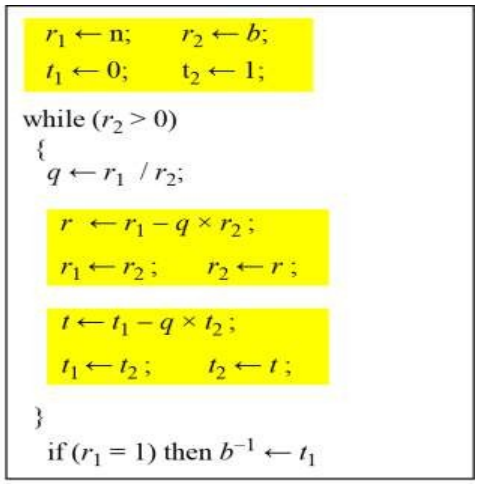
In modular arithmetic, an integer may or may not have a multiplicative inverse.When it does, the product of the integer and its multiplicative inverse is congruent to 1 modulo n.

The extended Euclidean algorithm finds the multiplicative inverses of b in Z_n when n and b are given and gcd (n, b) = 1.The multiplicative inverse of b is the value of t after being mapped to Z_n.

Using extended Euclidean algorithm tofind multiplicative inverse



a. Process



b. Algorithm

Find the multiplicative inverse of 11mod26 (ie. 11^{-1}mod26)

q	r_1	r_2	r	t_1	t_2	t
2	26	11	4	0	1	−2
2	11	4	3	1	−2	5
1	4	3	1	−2	5	−7
3	3	1	0	5	−7	26
	1	0		−7	26	

The gcd (26, 11) is 1; the inverse of 11 is -7 or 19.

Find the inverse of 12 in Z_{26}.

q	r_1	r_2	r	t_1	t_2	t
2	26	12	2	0	1	−2
6	12	2	0	1	−2	13
	2	0		−2	13	

The gcd (26, 12) is 2; the inverse does not exist**.**

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
- 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: $M = C^d \bmod n$
- 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 $\phi(n)=(p-1)(q-1)$
- selecting at random the encryption key e
    - where $1<e<\phi(n)$, $\gcd(e,\phi(n))=1$
- solve following equation to find decryption key d
    - $e.d=1 \bmod \phi(n)$ and $0 \le d \le n$
- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,n}

Key Generation Algorithm

Key Generation	
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	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

RSA Example

1. Select primes: p=17 &q=11
2. Calculate $n = pq$=17 x 11=187
3. Calculate ø(n)=(p–1)(q-1)=16x10=160
4. Select e:gcd(e,160)=1; choose e=7
5. Determine d:de=1 mod 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}

RSA Example - En/Decryption

- sample RSA encryption/decryption is:
- given message M = 88 (nb.88<187)
- encryption:

  C = 88^7 mod 187 = 11
- decryption:

  M = 11^{23} mod 187 = 88

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

Diffie-Hellman Key Exchange

- first public-key type scheme proposed
  - For key distribution only
- by Diffie& Hellman in 1976 along with the exposition of public key concepts
  - is a practical method for public exchange of a secret key
  - used in a number of commercial products
  - a public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants
  - value of key depends on the participants (and their private and public key information)
  - based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy
  - Security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard.
- all users agree on global parameters:
  - large prime integer or polynomial q
  - α a primitive root mod q
  - each user (eg. A) generates their key
  - chooses a secret key (number): $x_A < q$
  - compute their **public key**: $y_A = \alpha^{xA} \bmod q$
  - each user makes public that key y_A
- shared session key for users A & B is K:
  - $K = y_A^{xB} \bmod q$ (which **B** can compute)
  - $K = y_B^{xA} \bmod q$ (which **A** can compute)

- 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

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

$K = n_B \times P_A$

Diffie-Hellman Example

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

- compute shared session key as:
  - $K_{AB} = y_B^{xA} \bmod 353 = 248^{97} = 160$ (Alice)
  - $K_{AB} = y_A^{xB} \bmod 353 = 40^{233} = 160$ (Bob)

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
- an alternative is to use elliptic curves
- offers same security with smaller bit sizes

Real Elliptic Curves

- 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
  - also define zero point O

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
$K = n_B \times P_A$

- can do key exchange similar to D-H
- users select a suitable curve $E_p(a,b)$
  - Either a prime curve, or a binary curve
- select base point $G=(x_1,y_1)$ with large order n ($nG=O$)
- A & B select private keys $n_A<n$, $n_B<n$
- compute public keys: $P_A=n_A\times G$, $P_B=n_B\times G$
- compute shared key: $K=n_A\times P_B$, $K=n_B\times P_A$
  - same since $K=n_A\times n_B\times G$

ECC Encryption/Decryption

- select suitable curve & point G as in D-H
- encode any message M as a point on the elliptic curve $P_m=(x,y)$
- each user chooses private key $n_A<n$
- and computes public key $P_A=n_A\times G$
- to encrypt pick random k: $C_m=\{kG, P_m+k P_b\}$,
- decrypt C_m compute:
  - $P_m+kP_b-n_B(kG) = P_m+k(n_BG)-n_B(kG) = P_m$