GMP- Hands on Practice

Dr. R.Padmavathy

What is gmp?

- Portable Library
- High precision arithmetic operations
- Very fast operations on big numbers

How to install?

- Open Terminal
- sudo apt-get update
- sudo apt-get install libgmp3-dev

- Download GMP manual

  gmplib.org/gmp-man-6.0.0a.pdf

Basics

- Header file - #include<gmp.h>
- The mpz_t datatype:
- First declare a variable

  eg - mpz_t a,b,c;
- Initialize the variable

  eg - mpz_init(a); //single variable

  eg - mpz_inits(a,b,c,NULL); //multiple variable
- Setting the value to some unsigned integer type (ui)

  eg - mpz_set_ui(a,10000);

Input/ Output

- Taking input from users:

  gmp_scanf(“%Zd”, a);
- Printing on screen

  gmp_printf(“Value of a is : %Zd”,a);

Basic arithmetic in gmp

- **a > b** => mpz_cmp(a,b) > 0
- **a < b** => mpz_cmp(a,b) < 0
- **a == b** => mpz_cmp(a,b) == 0
- **a > 0** => mpz_cmp_ui(a,0) > 0
- **a < 0** => mpz_cmp_ui(a,0) < 0

- **swap** values of a and b -> mpz_swap(a,b);

Basic arithmetic in gmp

- **c = a + b** => mpz_add(c,a,b);
- **c = a - b** => mpz_sub(c,a,b);
- **c = a * b** => mpz_mul(c,a,b);
- **c = a / b** => mpz_fdiv_q(c,a,b);
- **c = a % b** => mpz_fdiv_r(c,a,b);
- **c = a + 5** => mpz_add_ui(c,a,5);
- **a++** => mpz_add_ui(a,a,1);
- **a--** => mpz_sub_ui(a,a,1);

Test Program : greater of two numbers

```
#include<gmp.h>
void main()
{
    mpz_t a,b,c;
    mpz_inits(a,b,c,NULL);
    gmp_printf(“\n Enter the value of a - “);
    gmp_scanf(“%Zd”,a);
    gmp_printf(“\n Enter the value of b - “);
    gmp_scanf(“%Zd”,b);
    if(mpz_cmp(a,b) > 0) printf(“a is greater”);
    else   printf(“b is greater”);
}
```

Compile and Run

- To complile -   gcc fileName.c -lgmp
- To run -   ./a.out

OR

- To complile -   gcc progName.c   -o   p1   -lgmp
- To run -   ./p1

Calculator Program

```
#include<gmp.h>
void main()
{
    mpz_t a,b,c;
    mpz_inits(a,b,c,NULL);
    gmp_printf(“\n Enter  a - “);
    gmp_scanf(“%Zd”,a);
    gmp_printf(“\n Enter b - “);
    gmp_scanf(“%Zd”,b);
    mpz_add(c,a,b);
    gmp_printf(“a + b = %Zd”,c);
    mpz_sub(c,a,b);
    gmp_printf(“a - b = %Zd”,c);
    mpz_mul(c,a,b);
    gmp_printf(“a * b = %Zd”,c);
    mpz_fdiv_q(c,a,b);
    gmp_printf(“a / b = %Zd”,c);
    mpz_mod(c,a,b);
    gmp_printf(“a % b = %Zd”,c);
}
```

Taking random inputs

- gmp_randstate_t state;
- gmp_randinit_mt(state);
- unsigned long seed;
- seed = time(NULL);
- gmp_randseed_ui(state,seed);

Method 1:

- mpz_set_ui(max,100000);
- mpz_rrandomm(a,state,max);

Method 2:

- int bits = 10;
- mpz_rrandomb(a,state,bits);

Euclidean Algorithm

- Input two numbers a and b
- while(mpz_cmp_ui(b,0) != 0)

```
 mpz_set(t,b);
 mpz_mod(b,a,b);
 mpz_set(a,t);
```

- gcd is 'a'

- Eg ->

1. a = 180 b = 48

   rem = 36
2. a = 48 b = 36

   rem = 12
3. a = 36 b = 12

   rem = 0
4. a = 12 b = 0

   gcd = 12

Assignment 1

- Write a program (in gmp) to find the gcd of two numbers using euclidean algorithm.

Extended Euclidean Algorithm

- The extended Euclidean algorithm is an extension to the Euclidean algorithm, it computes,
- the greatest common divisor of integers a and b, and
- the coefficients x and y such that
- $ax + by = \gcd(a,b)$

- s(i) = s(i-2) - q(i)*s(i-1)
- t(i) = t(i-2) - q(i)*t(i-1)

The following table shows how the extended Euclidean algorithm proceeds with input 240 and 46. The greatest common divisor is the last non zero entry, 2 in the column "remainder". The computation stops at row 6, because the remainder in it is 0. Bézout coefficients appear in the last two entries of the second-to-last row. In fact, it is easy to verify that −9 × 240 + 47 × 46 = 2. Finally the last two entries 23 and −120 of the last row are, up to the sign, the quotients of the input 46 and 240 by the greatest common divisor 2.

index i	quotient q_{i-1}	Remainder r_i	s_i	t_i
0		240	1	0
1		46	0	1
2	240 ÷ 46 = 5	240 − 5 × 46 = 10	1 − 5 × 0 = 1	0 − 5 × 1 = −5
3	46 ÷ 10 = 4	46 − 4 × 10 = 6	0 − 4 × 1 = −4	1 − 4 × −5 = 21
4	10 ÷ 6 = 1	10 − 1 × 6 = 4	1 − 1 × −4 = 5	−5 − 1 × 21 = −26
5	6 ÷ 4 = 1	6 − 1 × 4 = 2	−4 − 1 × 5 = −9	21 − 1 × −26 = 47
6	4 ÷ 2 = 2	4 − 2 × 2 = 0	5 − 2 × −9 = 23	−26 − 2 × 47 = −120

Multiplicative Inverse

- Modulo Arithmetics:

5 % 3 = 2 13 % 5 = 3 17 % 7 = 3 24 % 12 = 0

17 % 11 = ? 29 % 6 = ? 11 % 2 = ? 100 % 70 = ?

- Inverse in modulo arithmetics

  If a*b(% m) = 1, then a is the inverse of b in modulo m, and vice versa.

- Examples:
  - inv of 3 in (mod 5) = 2, [3 * 2 = 6 (%5) = 1]
  - inv of 9 in (mod 11) = 5, [9 * 5 = 45(%11) = 1]
  - inv of 2 in (mod 13) = 7, [2 * 7 = 14(%11) = 1]
- GMP Function : mpz_inv (inv, num, mod);

Inverse using Extended Euclidean

- Goal is to find the inverse of **a** in **(mod m)**.
- Use the equation of ex_eucliden

  ax + my = d, **d** is the gcd of **a** and **b**
- If **a** and **b** are co-primes, then gcd, **d = 1** .
- Taking **(mod m)** on both sides

  (ax + my) (mod m) = 1 (mod m)

  ax (mod m) = 1 (mod m)
- **x** is the inverse of **a** in mod **m**.

Primarility Test

- Fermat's Theorem : Let p be a prime. If gcd(a,p) = 1, then

$$a^{p-1} = 1(mod)p$$

Algorithm Fermat primality test

FERMAT(n,t)
INPUT: an odd integer $n \geq 3$ and security parameter $t \geq 1$.
OUTPUT: an answer "prime" or "composite" to the question: "Is n prime?"

1. For i from 1 to t do the following:
   - 1.1 Choose a random integer a, $2 \leq a \leq n-2$.
   - 1.2 Compute $r = a^{n-1} \bmod n$ using Algorithm 2.143.
   - 1.3 If $r \neq 1$ then return("composite").
2. Return("prime").

Assignment 2

int mpz_probab_prime_p (const mpz t n, int reps)

- Determine whether n is prime. Return 2 if n is definitely prime, return 1 if n is probably prime (without being certain), or return 0 if n is definitely composite.

void mpz_nextprime (mpz t rop, const mpz t op)

- Set rop to the next prime greater than op.

Write a program in GMP to generate 1024 bit prime number.

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

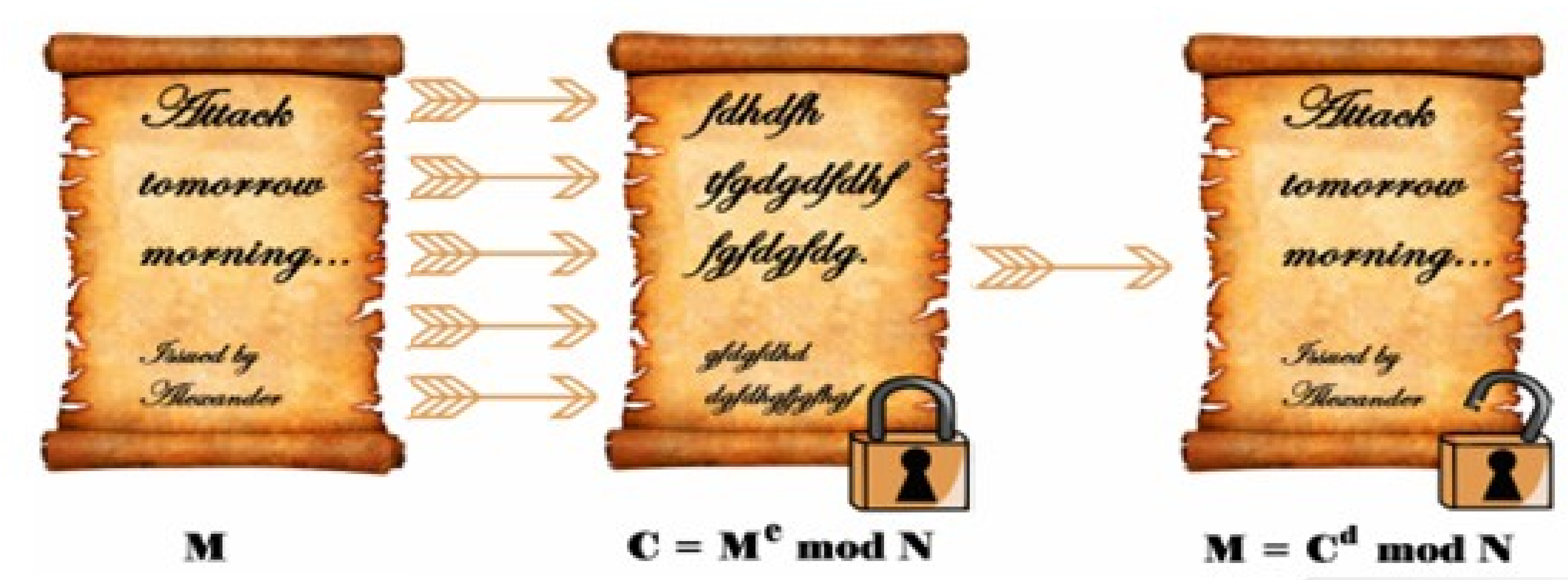
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption

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

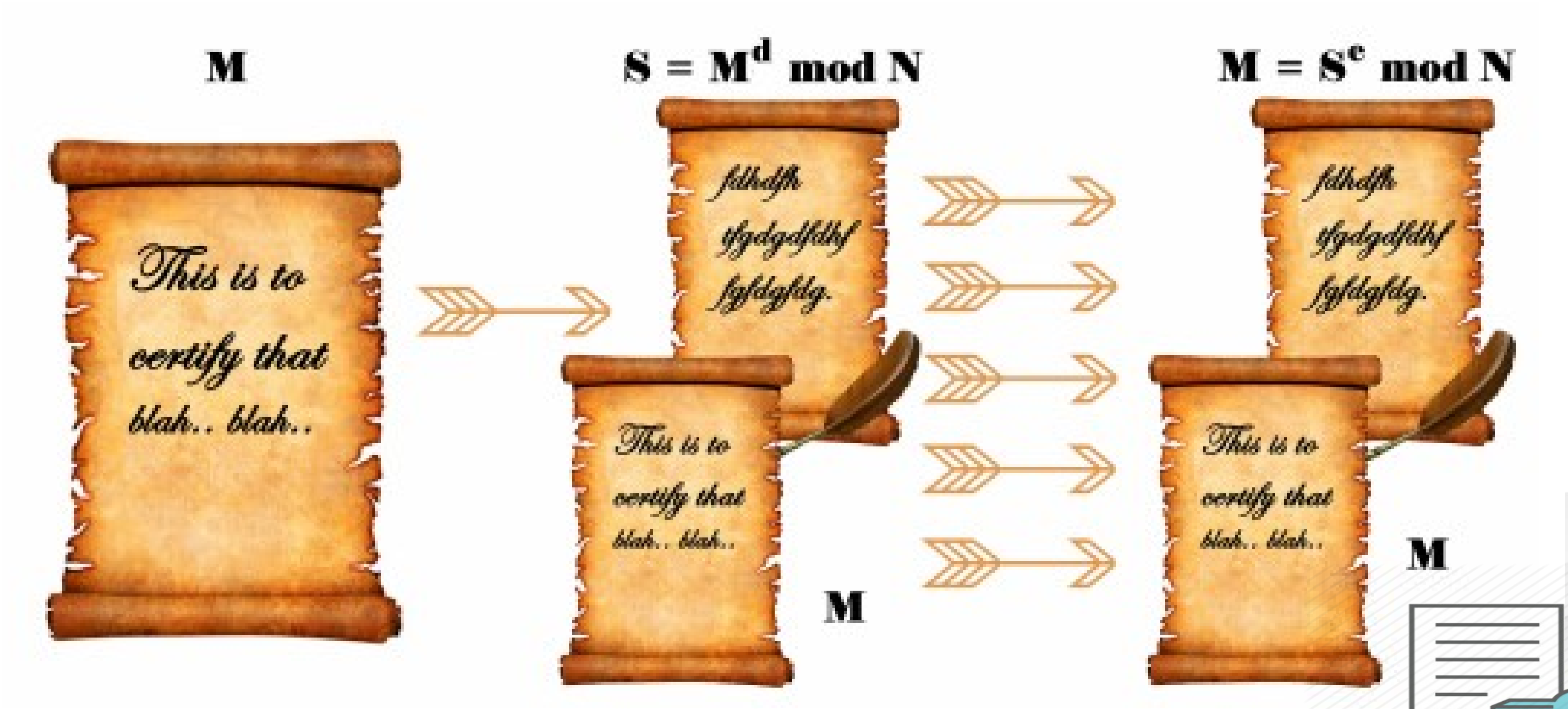
RSA Algorithm (Encryption)

Simplified model of RSA Cryptosystem



RSA Algorithm (Signature)

Simplified model of RSA Signature



RSA Algorithm Example

Step 1: Let p = 47 and q = 59. Thus n = 47 x 59 = 2773

Step 2: Select e = 17

Step 3: Publish (n,e) = (2773, 17)

Step 4: (p-1) x (q-1) = 46 x 58 = 2668

Use the Euclidean Algorithm to compute the modular inverse of 17 modulo 2668. The result is d = 157

<< Check: 17 x 157 = 2669 = 1(mod 2668) >>

Public key is (2773,17)

Private key is 157

RSA Algorithm Example Cont.

- Public key is (**2773**,**17**)
- Private key is **157**

- Plaintext block represented as a number: $M = 31$
- Encryption using Public Key: $C = 31^{17} \pmod{\mathbf{2773}}$

  $= \mathbf{587}$
- Decryption using Private Key: $M = \mathbf{587}^{157} \pmod{\mathbf{2773}}$

  $= 31$

Assignment 3

- Design RSA cryptosystem for secure communication.
- Design RSA for encrypting and decrypting string messages.

Generator of a field

- A generator g of a finite field F of order p (contains p elements) is an element whose first p - 1 powers generate all the nonzero elements of F. That is, the elements of F consist of 0, g^0, g^1 , ... , g^{p-1}.
- Example : Z_7 ={ 0,1,.....,6}
  - $g^1 = 3$, $g^2 = 2$, $g^3 = 6$, $g^4 = 4$, $g^5 = 5$, $g^6 = 1$
  - 3 generates all the elements of Z_7. Hence, it is the generator.

Algorithm Finding a generator of a cyclic group

INPUT: a cyclic group G of order n, and the prime factorization $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$.
OUTPUT: a generator α of G.

1. Choose a random element α in G.
2. For i from 1 to k do the following:
   - 2.1 Compute $b \leftarrow \alpha^{n/p_i}$.
   - 2.2 If $b = 1$ then go to step 1.
3. Return(α).

A simple trick

- Generate p from known prime factors.
- For eg : Choose p such that (p - 1 = q * r) , where p,q,r are all prime.
- To find out the generator

while(flag = 0){

generate α randomly from [2, p-1]

if(α^{q} *mod p !=1 && α^{r} mod p != 1)*

α is your generator;

flag = 1;

}

Diffie-Hellman Key Exchange Algorithm

Global Public Elements

q = prime number(300 decimal, i.e. 1024 bits)

α = generator of field F_q

User A key Generation

Select private X_a , $X_a < q$

Calculate public Y_a , $Y_a = \alpha^{X_a} \mod q$

User B Key Generation

Select private X_b , $X_b < q$

Calculate public Y_b , $Y_b = \alpha^{X_b} \mod q$

Diffie-Hellman Key Exchange Algorithm

Generation of secret key by user A

$$K=(Y_b)^{X_a} \bmod q$$

Generation of secret key by user B

$$K=(Y_a)^{X_b} \bmod q$$

Key Exchange Example

- users Alice & Bob who wish to swap keys:
- agree on prime $q=353$ and $\alpha=3$
- select random secret keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- compute respective public keys:
  - $y_A=397$ mod 353 = 40 (Alice)
  - $y_B=3233$ mod 353 = 248 (Bob)
- compute shared session key as:
  - $K_{AB}= y_B{}^{x_A}$ mod 353 = 24897 = 160 (Alice)
  - $K_{AB}= y_A{}^{x_B}$ mod 353 = 40233 = 160 (Bob)

Diffie-Hellman Key Exchange

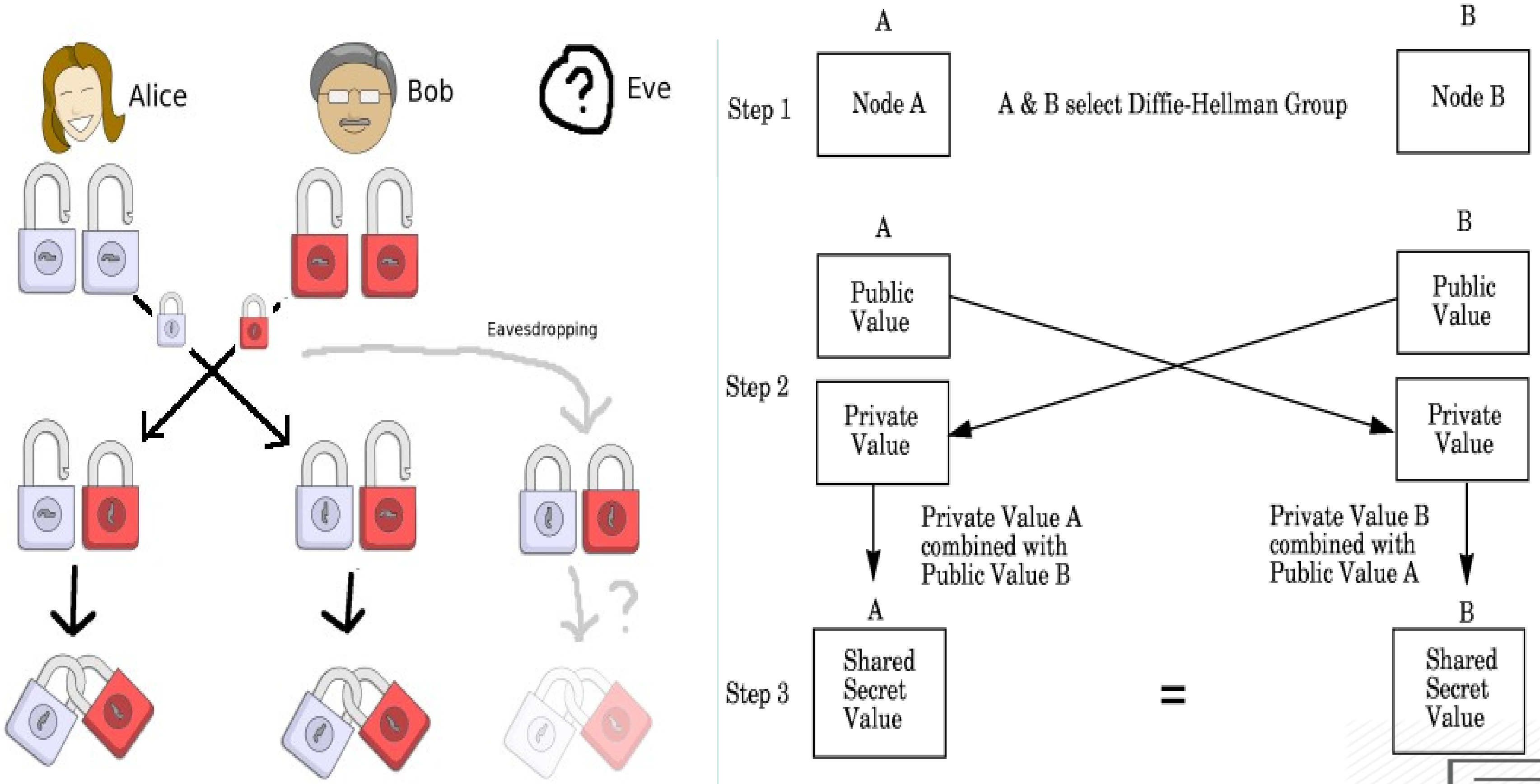


Fig-1 : Diffie-Hellman Key-Exchange Procedure

Assignment 4

- Design Diffie-Hellman key exchange algorithm in gmp.

ElGamal Algorithm

Let p be a large prime

By “large” we mean here a prime rather typical in length to that of an RSA modulus .

Select a special number **g**

The number g must be a **primitive element** modulo **p**.

Choose a private key **x**

This can be any number bigger than 1 and smaller than **p**-1

Compute public key **y** from **x**, **p** and **g**

The public key **y** is **g** raised to the power of the private key **x** modulo **p**. In other words:

$$y = g^{x} \bmod p$$

ElGamal Algorithm : Example

Step 1: Let p = 23

Step 2: Select a primitive element g = 11

Step 3: Choose a private key x = 6

Step 4: Compute $y = 11^6 \pmod{23}$

$= 9$

Public key is 9

Private key is 6

ElGamal Encryption

The first job is to represent the plaintext as a series of numbers modulo p. Then:

1. Generate a random number k
2. Compute two values C_1 and C_2, where

$$C_1 = g^k \bmod p \quad \text{and} \quad C_2 = M \cdot y^k \bmod p$$

3. Send the ciphertext C, which consists of the two separate values C_1 and C_2.

ElGamal Encryption: Example

To encrypt M = 10 using Public key **9**

1 - Generate a random number k = **3**

2 - Compute $C_1 = \mathbf{11}^3 \bmod \mathbf{23} = \mathbf{20}$

$C_2 = 10 \times \mathbf{9}^3 \bmod \mathbf{23}$

$= 10 \times 16 = 160 \bmod \mathbf{23} = \mathbf{22}$

3 - Ciphertext C = (**20** , **22**)

ElGamal Decryption

$$\boldsymbol{C_1 = g^{k} \bmod p \qquad C_2 = M \,.\, y^{k} \bmod p}$$

1 - The receiver begins by using their private key **x** to transform **C_1** into something more useful:

$$C_1^{x} = (g^{k})^{x} \bmod p$$

NOTE: $\boldsymbol{C_1^{x} = (g^{k})^{x} = (g^{x})^{k} = (y)^{k} = y^{k} \bmod p}$

2 - This is a very useful quantity because if you divide **C_2** by it you get **M**. In other words:

$$C_2 / y^{k} = (M \,.\, y^{k}) / y^{k} = M \bmod p$$

ElGamal Encryption : Example

To decrypt C = (20 , 22)

1 - Compute $20^6 = 16 \bmod 23$

2 - Compute $22 / 16 = 10 \bmod 23$

3 - Plaintext = 10

Assignment 5

- Design Elgamal cryptosystem using gmp.

Thank You…