|  |  |
| --- | --- |
| **Academic Year: 2024-25** | **Programme: BTECH-Cyber (CSE)** |
| **Year: 2nd** | **Semester: IV** |
| **Student Name: Jal Bafana** | **Batch : k1** |
| **Roll No: K005** | **Date of experiment: 13.03.2025** |
| **Faculty: Rejo Mathew** | **Signature with Date:** |

**Experiment 10: El Gamal Cryptosystem**

**Aim:** Write a program to implement El Gamal cryptosystem.

**Learning Outcomes:**

After completion of this experiment, student should be able to

1. Describe working of El Gamal cryptosystem.
2. Understand application of El Gamal along with its advantage and limitations.

**Theory:**

The **El Gamal cryptosystem** is an asymmetric encryption algorithm based on the **Discrete Logarithm Problem (DLP),** which makes it secure for encryption and digital signatures.

**Procedure:**

It consists of majorly three steps:

**Key Generation**

1. Choose a large prime p and a primitive root g (a generator).
2. Select a private key x (a random integer 1 < x < p−1).
3. Compute the public key: y = g x mod p
4. Public Key: (p, g, y), Private Key: x.

**Encryption (for message M)**

1. Choose a random integer k (ephemeral key).
2. Compute:

C1 = g k mod p

C2 = M ⋅ y k mod p

1. Ciphertext: (C1​, C2​).

**Decryption**

1. Compute the shared secret: S = C1 x mod p
2. Compute the inverse of S modulo p: M = C2 ⋅ S−1 mod p

**Example 1**

Given - Prime p = 23, Generator g = 5, Private Key x = 6

Compute Public Key

y = g x mod p = 5 6 mod 23 = 15625 mod 23 = 8

**Private Key:** x = 6

**Public Key:** (p = 23, g = 5, y = 8).

**Example 2**

Given – Public Key: (p = 23, g = 5, y = 8), Message M = 10, Random k = 3

Encryption

C1 = g k mod p = 5 3 mod 23 = 125 mod 23 = 10

C2 = M ⋅ y k mod p = 10 \* 8 3 mod 23 = 10 \* 512 mod 23 = 10 \* 6 mod 23 = 60 mod 23 = 14

Decryption

S = C1 x mod p = 10 6 mod 23 = 12

Find S−1 mod p = 12 -1 mod 23 ≡ 2 mod 23

M = C2 ⋅ S−1 mod p = 14 \* 2 mod 23 = 28 mod 23 = 10

The decrypted message and original message is 10

**Example 3**

Given – Public Key p = 29, Generator g = 7, Private Key x = 5, Message M = 11, Random k = 3

Key Generation

Compute Public Key

y = g x mod p = 75 mod 29 = 16807 mod 29 = 18

**Private Key:** x = 5

**Public Key:** (p = 29, g = 7, y = 18).

Encryption

C1 = g k mod p = 7 3 mod 29 = 343 mod 29 = 24

C2 = M ⋅ y k mod p = 11 \*18 3 mod 29 = 11 \* 5832 mod 29 = 11 \* 2 mod 29 = 22

(C1, C2) = (24,22)

Decryption

S = C1 x mod p = 24 5 mod 29 = 7

Find S−1 mod p = 7 -1 mod 29 ≡ 25 mod 29

M = C2 ⋅ S−1 mod p = 22 \* 25 mod 29 = 550 mod 29 = 11

The decrypted message and original message is 11

**Code: *type or copy your completed working code here***

*Note: Code should have proper comments*

def mod\_inverse(a, p):

# Extended Euclidean Algorithm to find the modular inverse of a mod p

t, new\_t = 0, 1

r, new\_r = p, a

while new\_r != 0:

quotient = r // new\_r

t, new\_t = new\_t, t - quotient \* new\_t

r, new\_r = new\_r, r - quotient \* new\_r

if r > 1:

return None # inverse doesn't exist

if t < 0:

t = t + p

return t

# function to check if g is a primitive root modulo p

def is\_primitive\_root(g, p):

# Check if g is a primitive root modulo p by checking powers g^i mod p for i in [1, p-1]

seen = set()

for i in range(1, p):

seen.add(pow(g, i, p))

return len(seen) == p - 1 # g is a primitive root if it generates all numbers from 1 to p-1

# Key Generation function

def key\_generation(p, g, x):

y = pow(g, x, p) # Public Key: y = g^x mod p

return y

# encryption function

def encryption(p, g, y, M, k):

C1 = pow(g, k, p) # C1 = g^k mod p

C2 = (M \* pow(y, k, p)) % p # C2 = M \* y^k mod p

return C1, C2

# Decryption function

def decryption(p, C1, C2, x):

S = pow(C1, x, p) # Shared secret S = C1^x mod p

S\_inv = mod\_inverse(S, p) # Find the modular inverse of S

if S\_inv is None:

return None # decryption not possible if inverse doesn't exist

M = (C2 \* S\_inv) % p # M = C2 \* S^-1 mod p

return M

# input

p = int(input("Enter a prime number p: "))

g = int(input("Enter a generator g: "))

# checking if g is a primitive root modulo p

if not is\_primitive\_root(g, p):

print(f"{g} is not a primitive root modulo {p}. Please choose a valid generator.")

exit()

print(f"{g} is a valid primitive root modulo {p}.")

# checking valid range 1 < x < p-1

x = int(input(f"Enter a private key x such that 1 < x < {p-1}: "))

if not (1 < x < p-1):

print(f"x must be in the range 1 < x < {p-1}. Please choose a valid private key.")

exit()

# public key generating

y = key\_generation(p, g, x)

print(f"Public Key: (p={p}, g={g}, y={y})")

print(f"Private Key: x={x}")

# input message M and random k

M = int(input("Enter a message M to encrypt: "))

k = int(input("Enter a random integer k for encryption: "))

# encrypting

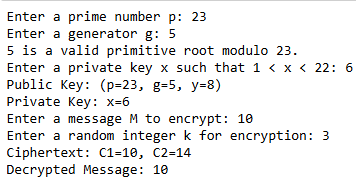
C1, C2 = encryption(p, g, y, M, k)

print(f"Ciphertext: C1={C1}, C2={C2}")

# decrypting

decrypted\_message = decryption(p, C1, C2, x)

print(f"Decrypted Message: {decrypted\_message}")



**Questions:**

1. Why is the generator g typically a small number even though p is large?

The generator ggg is small to make calculations faster. It still needs to generate all values in the group modulo ppp, which is large, but ggg can stay small for efficiency.

1. How does the selection of k affect the security of the ciphertext?

The value of kkk makes sure the same message gets different ciphertexts each time it's encrypted. Reusing kkk can weaken security and reveal patterns.

1. Can El Gamal encryption be used for **short messages**? If not, how is it typically handled?

El Gamal isn't great for short messages because it’s not efficient with small data. Instead, it’s often paired with faster symmetric encryption, like AES, to handle the message itself.

1. How does El Gamal ensure confidentiality but **not** message integrity? Which cryptographic techniques can be combined with El Gamal to ensure both??

El Gamal keeps messages secret, but it doesn’t check if they’re tampered with. To ensure both secrecy and integrity, it can be combined with digital signatures or a MAC (Message Authentication Code).

**Conclusion:** *El Gamal cryptosystem ensures confidentiality through encryption but doesn’t guarantee message integrity. To achieve both, it can be combined with techniques like digital signatures or MACs.*