|  |  |
| --- | --- |
| Roll. No. A016 | Name: Varun Khadayate |
| Class B.Tech CsBs | Batch: 1 |
| Date of Experiment: 17-09-2022 | Subject: Cryptology |

# Aim

To implement ElGamel Algorithm.

# Theory

In 1984, T. Elgamal announced a public-key scheme based on discrete logarithms, closely related to the Diffie-Hellman technique [ELGA84, ELGA85]. The Elgamal2 cryptosystem is used in some form in a number of standards including the digital signature standard (DSS), which is covered in Chapter 13, and the S/MIME e-mail standard (Chapter 19).

As with Diffie-Hellman, the global elements of Elgamal are a prime number *q* and , which is a primitive root of *q*. User A generates a private/public key pair as follows:

1. Generate a random integer *XA*, such that 1 6 *XA* 6 *q* - 1.
2. Compute *YA* = a*XA* mod *q*.
3. A’s private key is *XA* and A’s public key is {*q*, a, *YA*}.

Any user B that has access to A’s public key can encrypt a message as follows:

1. Represent the message as an integer *M* in the range 0 … *M* … *q* - 1. Longer messages are sent as a sequence of blocks, with each block being an integer less than *q*.
2. Choose a random integer *k* such that 1 … *k* … *q* - 1.
3. Compute a one-time key *K* = (*YA*)*k* mod *q*.
4. Encrypt *M* as the pair of integers (*C*1, *C*2) where

*C*1 = a*k* mod *q*; *C*2 = *KM* mod *q*

User A recovers the plaintext as follows:

1. Recover the key by computing *K* = (*C*1)*XA* mod *q*.
2. Compute *M* = (*C*2*K*-1) mod *q*.

These steps are summarized in Figure 10.3. It corresponds to Figure 9.1a: Alice generates a public/private key pair; Bob encrypts using Alice’s public key; and Alice decrypts using her private key.

Let us demonstrate why the Elgamal scheme works. First, we show how *K* is recovered by the decryption process:

*K* = (*YA*)*k* mod *q K* is defined during the encryption process

*K* = (a*XA* mod *q*)*k* mod *q* substitute using *YA* = a*XA* mod *q K* = a*kXA* mod *q* by the rules of modular arithmetic

*K* = (*C*1)*XA* mod *q* substitute using *C*1 = a*k* mod *q*

Next, using *K*, we recover the plaintext as

*C*2 = *KM* mod *q*

(*C*2*K*-1) mod *q* = *KMK*-1 mod *q* = *M* mod *q* = *M*

Select private *XA*

Calculate *YA*

Public key Private key

**Key Generation by Alice**

*XA* 6 *q* - 1

*YA* = a*XA* mod *q*

{*q*, a, *YA*} *XA*

**Decryption by Alice with Alice’s Private Key**

Ciphertext: (*C*1, *C*2)

Calculate *K K* = (*C*1)*XA* mod *q*

Plaintext: *M* = (*C*2*K*-1) mod *q*

**Encryption by Bob with Alice’s Public Key**

Plaintext: *M* 6 *q*

Select random integer *k k* 6 *q*

Calculate *K K* = (*YA*)*k* mod *q*

Calculate *C*1 *C*1 = a*k* mod *q*

Calculate *C*2 *C*2 = *KM* mod *q*

Ciphertext: (*C*1, *C*2)

*q*

a

**Global Public Elements**

prime number

a 6 *q* and a a primitive root of *q*

1. Bob generates a random integer *k*.
2. Bob generates a one-time key *K* using Alice’s public-key components *YA*, *q*, and *k*.
3. Bob encrypts *k* using the public-key component a, yielding *C*1. *C*1 provides sufficient information for Alice to recover *K*.
4. Bob encrypts the plaintext message *M* using *K*.
5. Alice recovers *K* from *C*1 using her private key.
6. Alice uses *K*-1 to recover the plaintext message from *C*2.

Thus, *K* functions as a one-time key, used to encrypt and decrypt the message.

For example, let us start with the prime field GF(19); that is, *q* 19. It has primitive roots {2, 3, 10, 13, 14, 15}, as shown in Table 8.3. We choose a = 10.

Alice generates a key pair as follows:

1. Alice chooses *XA* = 5.
2. Then *YA* = a*XA* mod *q* a5 mod 19 3 (see Table 8.3).
3. Alice’s private key is 5 and Alice’s public key is {*q*, a, *YA*} = {19, 10, 3}. Suppose Bob wants to send the message with the value *M* = 17. Then:
4. Bob chooses *k* 6.
5. Then *K* = (*YA*)*k* mod *q* = 36 mod 19 729 mod 19 7.
6. So

*C*1 = a*k* mod *q* = a6 mod 19 = 11

*C*2 = KM mod *q* = 7 \* 17 mod 19 = 119 mod 19 = 5

1. Bob sends the ciphertext (11, 5). For decryption:

**1.** Alice calculates *K* = (*C*1)*XA* mod *q* = 115 mod 19 = 161051 mod 19 = 7.

**2.** Then *K* -1 in GF(19) is 7-1 mod 19 = 11.

**3.** Finally, *M* = (*C*2*K* -1) mod *q* = 5 \* 11 mod 19 = 55 mod 19 = 17.

If a message must be broken up into blocks and sent as a sequence of encrypted blocks, a unique value of *k* should be used for each block. If *k* is used for more than one block, knowledge of one block *M*1 of the message enables the user to compute other blocks as follows. Let

*C*1,1 = a*k* mod *q*; *C*2,1 = *KM*1 mod *q C*1,2 = a*k* mod *q*; *C*2,2 = *KM*2 mod *q*

Then,

*C*2,1

= *KM*1 mod *q* = *M*1 mod *q*

*C*2,2

*KM*2 mod *q M*2 mod *q*

If *M*1 is known, then *M*2 is easily computed as

*M*2 = (*C*2,1)-1 *C*2,2 *M*1 mod *q*

The security of Elgamal is based on the difficulty of computing discrete logarithms. To recover A’s private key, an adversary would have to compute *XA* = dloga,*q*(*YA*). Alternatively, to recover the one-time key *K*, an adversary would have to determine the random number *k*, and this would require computing the discrete logarithm *k* = dloga,*q*(*C*1). [STIN06] points out that these calculations

are regarded as infeasible if *p* is at least 300 decimal digits and *q* - 1 has at least one “large” prime factor.

# Code

print("Key Generation Process")

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

d = int(input("Enter a decryption key : "))

e1 = int(input("Enter the 2nd part of Encryption Key : "))

e2 = pow(e1,d) % p

print("The 3rd Part of Encryption Key is : ",e2)

print("The Public Key is : ",[e1,e2,p])

print("\n--------------------------------------------")

print("\nEncrption Process")

r = int(input("Enter a random integer : "))

c1 = pow(e1,r)%p

print("Computer Cipher text 1 is : ",c1)

pt = int(input("Enter the lenght of Plain Text : "))

c2 = pt \* pow(e2,r)%p

print("Computer Cipher text 2 is : ",c2)

print("The Cipher Text is : ",[c1,c2])

print("\n--------------------------------------------")

print("\nDecryption Process")

x = pow(c1,d)

i = 1

while True:

    if(i\*x % p == 1):

        D = i

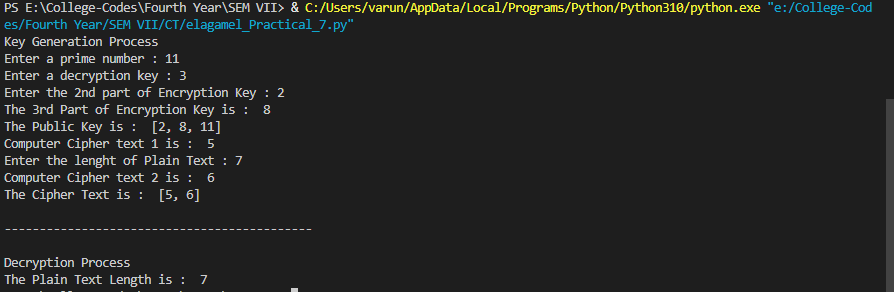
        break

    i += 1

PT = (c2\*D)%p

print("The Plain Text Length is : ",PT)

# Output



# Conclusion

Hence, we were able to perform ElGamel Algorithm.