Practical Assignment 5

Name: Kshitij Sharma
Roll no: 18075030
Department: CSE(B.Tech)

Github Link: Here

El Gamal Algorithm

It is an asymmetric encryption algorithm where there are two keys (public and private), public key is used for encryption and the private key is used for decryption

Suppose Bob and Alice want to communicate, Bob wants to send a message to Alice.

The following parameters would be randomly chosen when the session is initiated (the parameters are public)

- p: A large enough prime number (For this illustration we have used a 64 byte integer)
- g: A generator in the range [1, p)

Now, both Alice and Bob would generate their private and public keys as follows:

	Private Key	**Public Key**
Alice	A randomly chosen number in the range [0, p-1] Suppose the chosen number is a	$g^a \mod p$
Bob	A randomly chosen number in the range [0, p-1] Suppose the chosen number is b	$g^b \mod p$

Now, if Alice wants to send Bob a message, first Bob would share its public key with Alice, let's suppose it is X, and let's suppose that the message is M

Alice would generate the following encrypted message:

$$M \cdot X^a \mod p$$

Note that this is equivalent to:

$$M \cdot g^{ab} \mod p$$

Alice would send the above encrypted message to Bob along with his own public key (i.e., $g^a \mod p$, let's suppose it is Y)

Bob now would want to decrypt the message, to do so he will calculate the following:

$$\text{encryptedMessage} \cdot Y^{-b} \mod p$$

Note that it is equivalent to:

$$M \cdot g^{ab} \cdot Y^{-b} \mod p$$
$$M \cdot g^{ab} \cdot g^{-ab} \mod p$$
$$M \mod p$$

Hence, Bob would be able to decrypt the message.

This works because even if someone know the public keys ($g^a \mod p$ and $g^b \mod p$) it is extremely difficult to find $g^{ab} \mod p$

Snippet for Send Message Function:

```
def send_message(self, message, recepient):
    recepient_public_key = recepient.get_public_key()
    key = calculate_power_modulo(recepient_public_key, self.private_key, p)
    encrypted_message = message * key
    return encrypted_message
```

Snippet for Receive Message Function:

```
def receive_message(self, encrypted_message, sender):
    encrypted_message %= p
    sender_public_key = sender.get_public_key()
    temp = calculate_power_modulo(sender_public_key, self.private_key, p)
    temp = calculate_power_modulo(temp, p - 2, p)
    temp = temp * (encrypted_message % p)
    message = temp % p
    return message
```

Code Output:

```
PS C:\Users\kshit\Desktop\sem8\Network Security\Assignments\Practical Assignment 0> & C:/Users/kshit/Anacon
thon.exe "c:/Users/kshit/Desktop/sem8/Network Security/Assignments/Practical Asisgnment 4/code.py"
creating credentials for person1
generated private_key =  130914920825384686825190053 89
generated public_key =  13068568288168357246399908 1778
creating credentials for person2
generated private_key =  171815711751992595535016031978
generated public_key =  143095198890533789082155248841
random message =  174606852099295653525985973978
sending random_message from person1 to person2...
encrypted message:  495440970553192656943656335059735862257135935463617791354
decrypting message received from person1 to person2...
decrypted_message:  174606852099295653525985973978
decrypted_message and message sent matched!
PS C:\Users\kshit\Desktop\sem8\Network Security\Assignments\Practical Assignment 0>
```