Hand-in 1

antni

October 2022

1 Part 1

For the first part of the assignment I have created four functions in Python that implements the basic operations I need for the assymetric public key method El Gamal.

1.1 Generating the Keys

The function *gen_keypair* is where I choose a random secret key $sk \in \mathbb{Z}_q^*$. The generic group is integers with times as the group operator. The order of the group and the generator have been given in the assignment as p and g respectively.

The function returns the public key and the secret key as a pair. The public key is generated with a call to the function *gen_pubkey*, which computes the public key by taking the generator g to the power of the secret key sk.

$$g^{sk}$$

I will need to apply modulo with the order of the group to all of the computations to make sure that we stay inside the group. If we do not do this, the generator breaks. So for this it would be:

$$g^{sk} \pmod p$$

1.2 Encryption

The encryption of messages is handled by the function *encrypt*. Here I take the public key of the recipient to the power of the senders secret key. I then apply the result to the message with the group operator which is times in this case.

$$pk^{sk} \cdot m$$

1.3 Decryption

The decryption of messages is handled by the function *decrypt*. To get the plaintext message from the ciphertext we need to take the inverse of the encryption.

$$\frac{c_2}{g^{sk^{pk}}} = \frac{g^{sk^{pk}} \cdot m}{g^{sk^{pk}}} = m$$

In this case we need the modular multiplicative inverse which we can get by using *Fermat's little theorem.*

$$a^{p-1} \equiv a \pmod{p}$$

By multiplying both sides with a^{-1}, this can be rewritten as:

$$a^{-1} \equiv a^{p-2} \pmod{p}$$

We can do this because we know that the order p is a prime and our group is the group of integers.

1.4 Result

```python
from random import randint

# Our predefined values
g = 666
p = 6661
bob_pk = 2227
message = 2000


def gen_pubkey(base, sk, prime):
    return (base ** sk) % prime

def gen_keypair(base, prime):
    sk = randint(1, prime)
    return (gen_pubkey(base, sk, prime), sk)

def encrypt(message, receiver_pk, sk, prime):
    return receiver_pk**sk % prime * message

def decrypt(ciphertext, sender_pk, sk, prime):
    return ciphertext * sender_pk**(sk * (prime-2)) % prime

# Alice's Keypair
(alice_pk, alice_sk) = gen_keypair(g, p)

# Alice's Encrypted Message
```

```python
ciphertext = encrypt(message, bob_pk, alice_sk, p)


# Part 1
print("1:")
print("Ciphertext sent to Bob: ", ciphertext)
print("\n")
```

2 Part 2

2.1 Bruteforcing Secret Keys

By intercepting Alice's encrypted message to Bob I learn of Alice's public key and the ciphertext. I assume that we also have Bob's public key from intercepting the initial key exchange. If we know the generator and the group order we can now bruteforce Bob's secret key by generating public keys from a secret key $sk \in \mathbb{Z}_q^*$. If the generated public key matches Bob's public key, then we conclude that the secret key sk is Bob's secret key.

2.2 Result

```python

def bruteforce_sk(ct, base, prime, pk):
    for sk in range(p):
        if gen_pubkey(base, sk, prime) == pk:
            break
    return sk

```

```python

# Part 2
bob_sk = bruteforce_sk(ciphertext, g, p, bob_pk)

print("2:")
print("Ciphertext: ", ciphertext)
print("Decrypted Ciphertext: ", decrypt(ciphertext, alice_pk, bob_sk, p))
print("Bob's private key: ", bob_sk)
print("\n")

```

3 Part 3

3.1 Modifying a Ciphertext

By using El Gamal we get confidentiality, but we do not get any integrity. This means that we can change the message even if we cannot read it.

The original plain text message is '2000' and we want to change the message to '6000'. In this case I assume that we know the original plain text message. The message is an integer value, so we should be able to just multiply it by 3 to get the desired result. This works because of associativity.

$$g^{sk^{pk}} \cdot (m \cdot 3) = (g^{sk^{pk}} \cdot m) \cdot 3$$

3.2 Result

```python

# Part 3
modified_message = ciphertext * 3 % p

print("3:")
print("Ciphertext: ", ciphertext)
print("Decrypted Ciphertext", decrypt(ciphertext, alice_pk, bob_sk, p))
print("Modified Ciphertext: ", modified_message)
print("Decrypted Modified Ciphertext", decrypt(modified_message, alice_pk, bob_sk, p))

```