Information Security Homework#1

I. DH key exchange

1.

```python
import random
p=13#大素数
g=7#生成元
def mod_exp(g,x,p=13):#加密函数
    return (g**x)%p
def diffie_hellman(verbose=False,number_alice=random.randint(1,p-1),number_bob=random.randint(1,p-1))-> bool:
    message_alice = mod_exp(g,number_alice)#alice发送的信息
    message_bob=mod_exp(g,number_bob)#bob发送的消息
    key_alice=mod_exp(message_bob,number_alice)
    key_bob=mod_exp(message_alice,number_bob)
    if(verbose):#输出调试信息
        print("Prime number:" ,p)
        print("generator number:",g)
        print("number chosen by Alice:",number_alice)
        print("number chosen by Bob:",number_bob)
        print ("message sent by Alice :",message_alice)
        print("message sent by Bob :",message_bob)
        print("common key got by Alice: ",key_alice)
        print("common key got by Bob: ",key_bob)
        print("Are their key the same:",bool(key_bob==key_alice))
    return key_bob==key_alice
diffie_hellman(verbose=True,number_alice=10,number_bob=6)#模拟单次过程
simulate_number=0
simulate_times=1000#模拟多次过程看是否成功
for i in range(simulate_times):
    if(diffie_hellman()):
        simulate_number+=1
print("simulate_times:",simulate_times)
print("sucess_times:",simulate_number)
```

```
Prime number: 13
generator number: 7
number chosen by Alice: 10
number chosen by Bob: 6
message sent by Alice : 4
message sent by Bob : 12
common key got by Alice:  1
common key got by Bob:  1
Are their key the same: True
simulate_times: 1000
sucess_times: 1000
```

As shown in the image above, I wrote a piece of code to simulate the Diffie-Hellman key exchange. Alice and Bob each choose a number and use a public generator (g=7) and a large prime number(p=13) as public keys for encryption. They then send the encrypted results to each other. (mod_exp function), Alice and Bob use the information from the other party as a new generator and input their private keys to encrypt and obtain the same shared public key. To make the results more general, I randomly generated 1000 pairs of private keys for Alice and Bob, performed simulations, compared how many times the public keys were the same, and found that they were all identical.

2. (a).

Man-in-the-middle attack: A Man-in-the-middle attack occurs when an attacker secretly positions themselves between two communicating parties, intercepting and possibly altering the messages exchanged. The attacker impersonates each party to the other, making both believe they are communicating directly and securely. Take for example, the attacker sends messages to Alice that appear to come from Bob and sends messages to Bob that appear to come from Alice. In this way, each party continues the exchange under the false assumption that they are talking directly to the other.

(b).

```python
def man_in_middle_attack(verbose=False,number_alice=random.randint(1,p-1),number_bob=random.randint(1,p-1),number_mallory=random.randint(1,p-1)):
    message_alice = mod_exp(g,number_alice)#alice发送的信息
    message_bob = mod_exp(g,number_bob)#bob发送的消息
    message_mallory = mod_exp(g,number_mallory)
    key_alice = mod_exp(message_mallory,number_alice)
    key_bob = mod_exp(message_mallory,number_bob)
    key_mallory_alice = mod_exp(message_alice,number_mallory)
    key_mallory_bob = mod_exp(message_bob,number_mallory)
    if(verbose):#输出调试信息
        print("Prime number:" ,p)
        print("generator number:",g)
        print("number chosen by Alice:",number_alice)
        print("number chosen by Bob:",number_bob)
        print("number chosen by mallory:",number_mallory)
        print ("message sent by Alice :",message_alice)
        print("message sent by Bob :",message_bob)
        print ("message sent by mallory :",message_mallory)
        print("common key got by Alice: ",key_alice)
        print("common key got by Bob: ",key_bob)
        print("common key of Alice got by mallory: ",key_mallory_alice)
        print("common key of Bob got by mallory: ",key_mallory_bob)
man_in_middle_attack(verbose=True,number_alice=10,number_bob=6,number_mallory=5)
```

```
Prime number: 13
generator number: 7
number chosen by Alice: 10
number chosen by Bob: 6
number chosen by mallory: 5
message sent by Alice : 4
message sent by Bob : 12
message sent by mallory : 11
common key got by Alice:  10
common key got by Bob:  12
common key of Alice got by mallory:  10
common key of Bob got by mallory:  12
```

Similarly, I wrote a piece of code to simulate the man-in-the-middle attack, Mallory pretended to be Alice and Bob and sent a message to Bob and Alice, respectively. Mallory used the same mod_exp function to generate his message(11). Then Mallory used the information from Alice and his secret key to generate the shared key with Alice(10), and he used the message from Bob and his secret key to generate the shared key with Bob(12). The simulation outcome suggests that Mallory can share the same key with Alice and Bob, respectively, while the keys shared by Bob and Alice are different.

II. RSA Encryption

1.

Firstly, we choose two large prime numbers p and q, and calculate the n=pq, phi_n=(p-1)(q-1). Next, we found our secret key d and public key e, which meet the requirements of gcd(e,phi_n)=1 and (de)mod phi_n =1. All people know n and e, but only you know d. When other people want to send a message to you, he uses the mod_exp function (message, public_key e,n) to generate the ciphertext to you. When you receive ciphertext, you use the mod_exp function (ciphertext, secret_key d,n) to get the message.

2.

```python
def egcd(a, b):
    if a == 0:
        return b, 0, 1
    gcd, x1, y1 = egcd(b % a, a)
    x = y1 - (b // a) * x1
    y = x1
    return gcd, x, y

def mod_inverse(a, m):
    gcd, x, _ = egcd(a, m)
    if gcd != 1:
        raise Exception('模逆元不存在')
    else:
        return x % m
prime_number_p = 11
prime_number_q = 17
secret_key = 19
message = 8  # 明文
def RSA_encryption(verbose=False):
    n=prime_number_p*prime_number_q
    phi_n = (prime_number_p - 1) * (prime_number_q - 1)
    public_key = mod_inverse(secret_key, phi_n)
    cipher_text = mod_exp(message, public_key, n)
    message_decryption = mod_exp(cipher_text, secret_key, n)
    if(verbose):…
RSA_encryption(verbose=True)
```

```
prime_number_p = 11
prime_number_q = 17
n = 187
φ(n) = 160
secret_key = 19
public_key = 59
message= 8
cipher_text = 172
message_decryption' = 8
Is sucessful?: True
```

(1)

I also wrote a piece of code to simulate this process, I used egcd to find the mod_inverse of the secret key under phi_i. After calculation, the public key is 59.

(2)

Using the mod_exp function(message=8, public key=59, n=187), we can get the ciphertext c=172. Then we use the mod_exp function (ciphertext=172, secret key=19, n=187) to decryption. We get the same 8 with the massage 8.