Network Security
Programming Exercise-3

Algorithm to Pick the Project

k=(A1+A2)%2
k=(1019+1039)%2=0

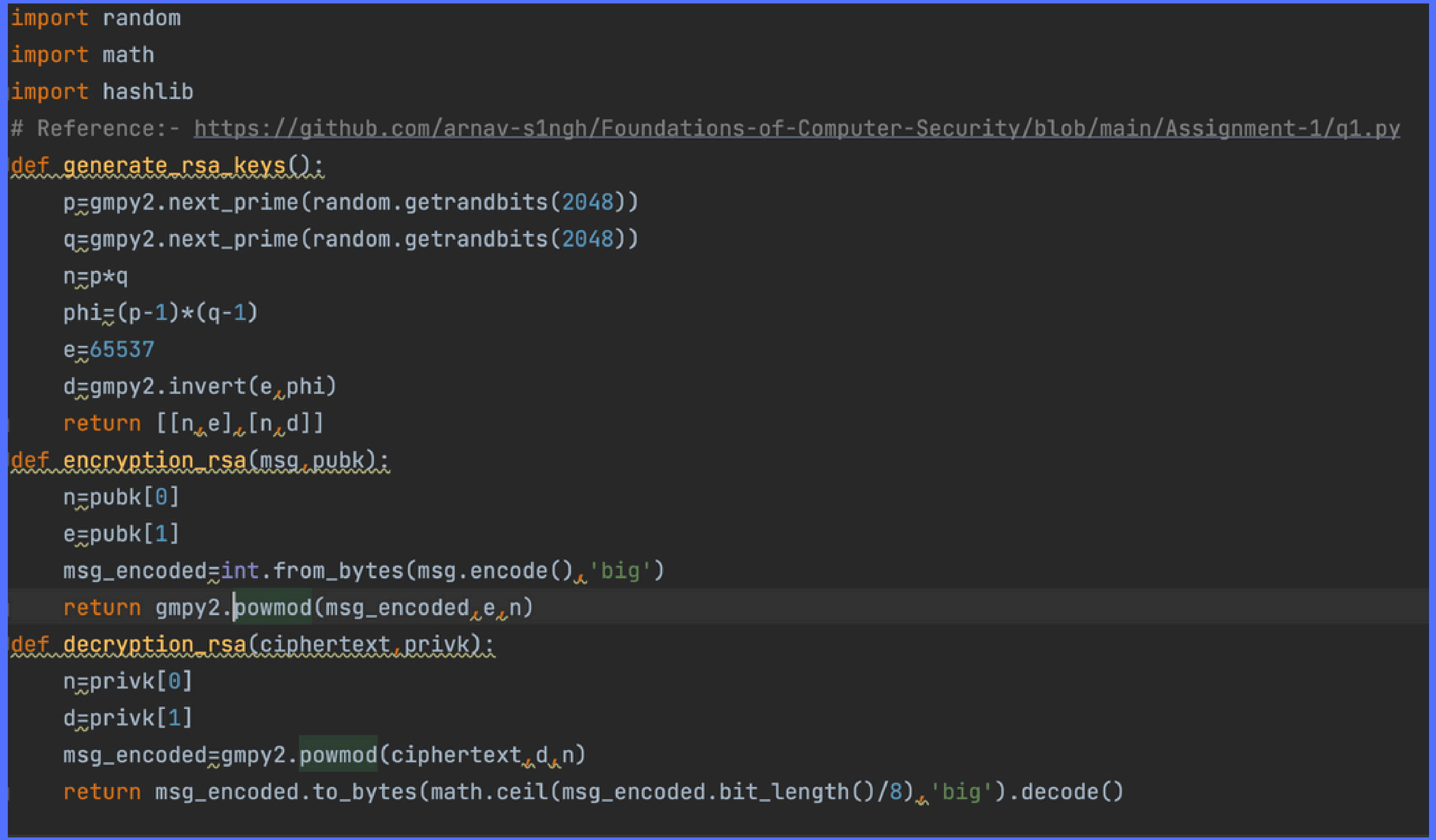
Assigned Project:- RSA-based Public-key CA

Arnav Singh (2021019)   Dharani Kumar S. (2021039)

Objectives

- Issue and verify RSA-based digital certificates.
- Allow clients to securely exchange messages.
- Ensure confidentiality and authenticity.

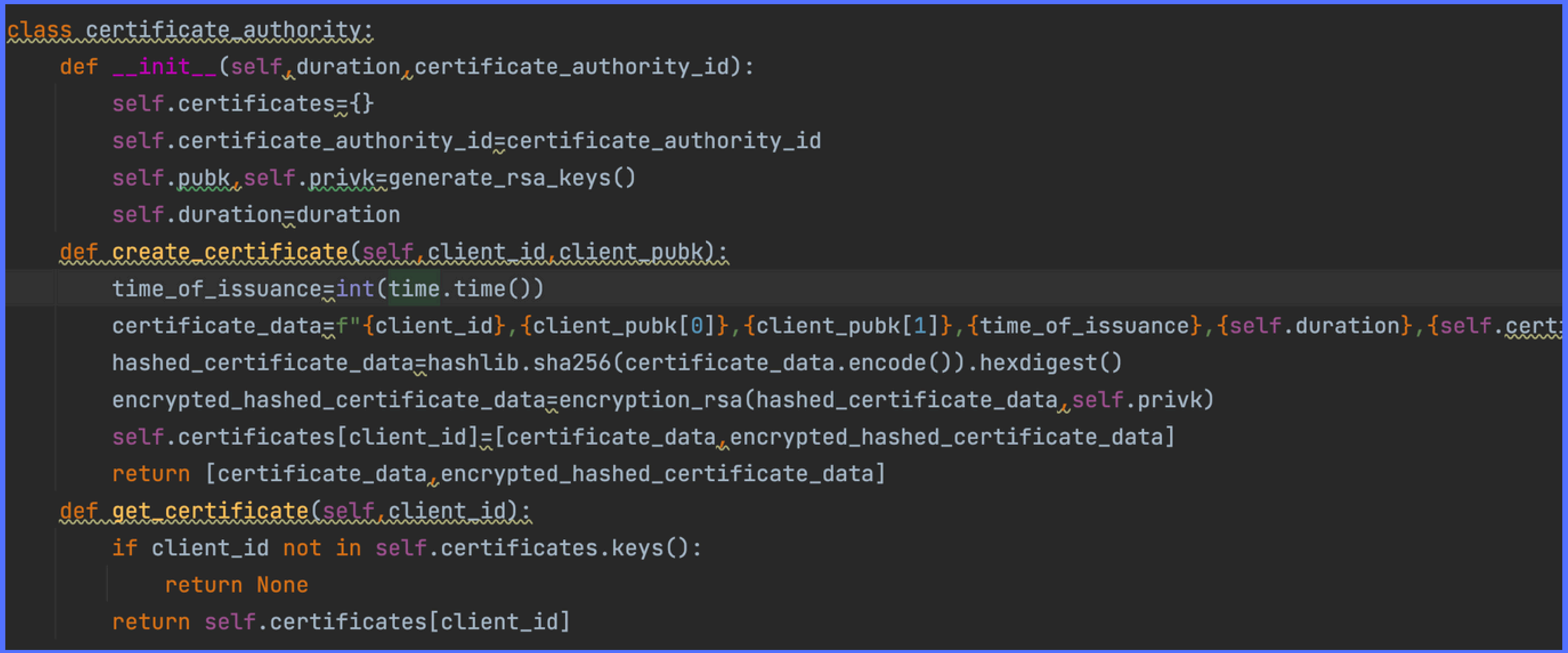
RSA



```python
import random
import math
import hashlib
# Reference:- https://github.com/arnav-s1ngh/Foundations-of-Computer-Security/blob/main/Assignment-1/q1.py
def generate_rsa_keys():
    p=gmpy2.next_prime(random.getrandbits(2048))
    q=gmpy2.next_prime(random.getrandbits(2048))
    n=p*q
    phi=(p-1)*(q-1)
    e=65537
    d=gmpy2.invert(e,phi)
    return [[n,e],[n,d]]
def encryption_rsa(msg,pubk):
    n=pubk[0]
    e=pubk[1]
    msg_encoded=int.from_bytes(msg.encode(),'big')
    return gmpy2.powmod(msg_encoded,e,n)
def decryption_rsa(ciphertext,privk):
    n=privk[0]
    d=privk[1]
    msg_encoded=gmpy2.powmod(ciphertext,d,n)
    return msg_encoded.to_bytes(math.ceil(msg_encoded.bit_length()/8),'big').decode()
```

- select two large prime numbers
- generate n, phi, d, and e.
- create public key (n,e) and private key (n,d)
- encrypt messages by converting their bytestreams to integers, and performing a powmod operation with e and n. convert the ciphertext back by performing a powmod operation with d and n, and then converting the integer back into a string

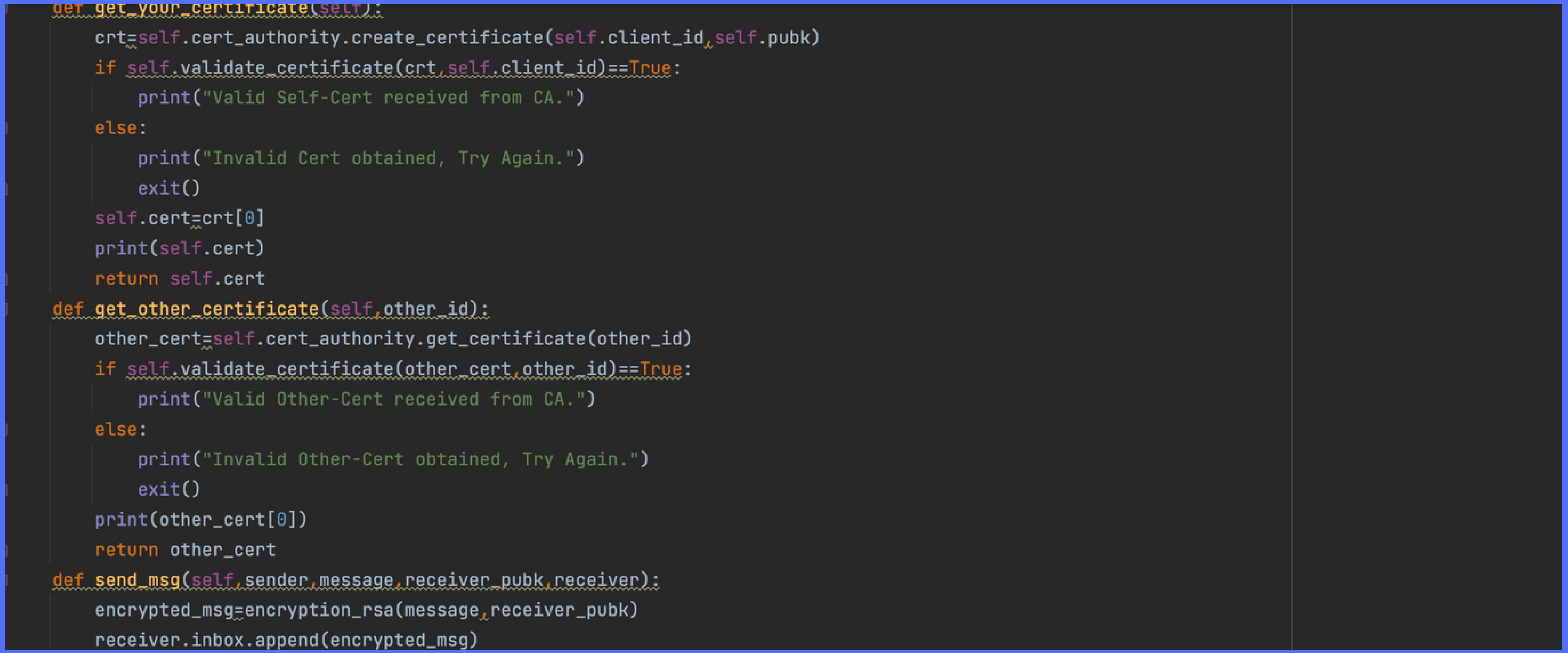
Cert. Auth.



```python
class certificate_authority:
    def __init__(self,duration,certificate_authority_id):
        self.certificates={}
        self.certificate_authority_id=certificate_authority_id
        self.pubk,self.privk=generate_rsa_keys()
        self.duration=duration
    def create_certificate(self,client_id,client_pubk):
        time_of_issuance=int(time.time())
        certificate_data=f"{client_id},{client_pubk[0]},{client_pubk[1]},{time_of_issuance},{self.duration},{self.cert
        hashed_certificate_data=hashlib.sha256(certificate_data.encode()).hexdigest()
        encrypted_hashed_certificate_data=encryption_rsa(hashed_certificate_data,self.privk)
        self.certificates[client_id]=[certificate_data,encrypted_hashed_certificate_data]
        return [certificate_data,encrypted_hashed_certificate_data]
    def get_certificate(self,client_id):
        if client_id not in self.certificates.keys():
            return None
        return self.certificates[client_id]
```

- stores all the generating certs in a dict
- certificates are transferred along with a hash to ensure integrity
- the certificate contains client id, client public key, time of issuance, duration, and an encrypted (with ca’s privk) hash of a concatenated string containing the prev mentioned info.

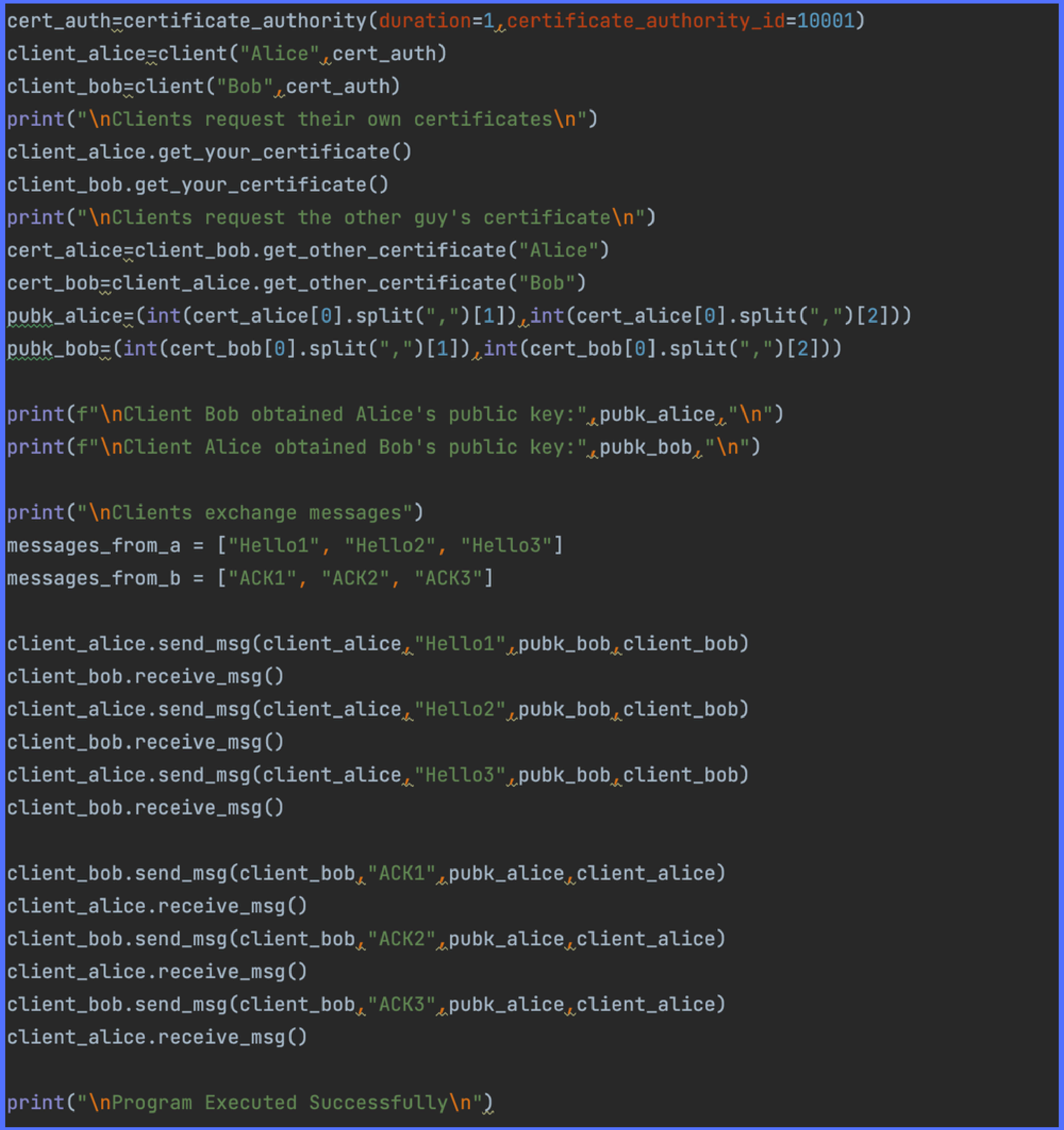
Client



```
    def get_your_certificate(self):
        crt=self.cert_authority.create_certificate(self.client_id,self.pubk)
        if self.validate_certificate(crt,self.client_id)==True:
            print("Valid Self-Cert received from CA.")
        else:
            print("Invalid Cert obtained, Try Again.")
            exit()
        self.cert=crt[0]
        print(self.cert)
        return self.cert
    def get_other_certificate(self,other_id):
        other_cert=self.cert_authority.get_certificate(other_id)
        if self.validate_certificate(other_cert,other_id)==True:
            print("Valid Other-Cert received from CA.")
        else:
            print("Invalid Other-Cert obtained, Try Again.")
            exit()
        print(other_cert[0])
        return other_cert
    def send_msg(self,sender,message,receiver_pubk,receiver):
        encrypted_msg=encryption_rsa(message,receiver_pubk)
        receiver.inbox.append(encrypted_msg)
```

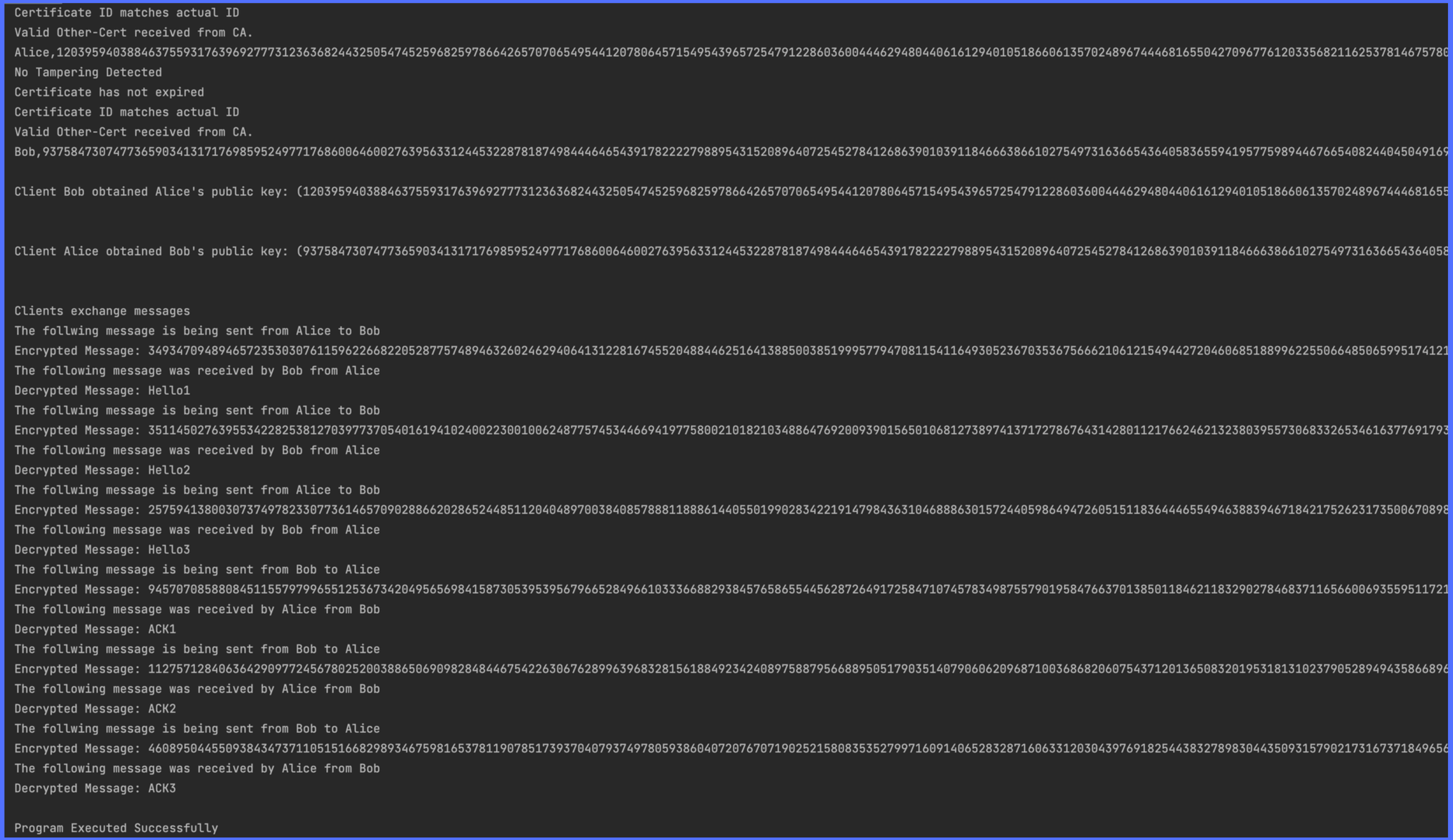
- retrieves certificates from the certificate authority
- validates retrieved certificates to ensure the cia triad is held
- sent messages are added to the receiver’s inbox along with the associated contact
- each client is supposed to connect to the same certificate authority in order to fetch the other’s certificate

Main Procedure



```python
cert_auth=certificate_authority(duration=1,certificate_authority_id=10001)
client_alice=client("Alice",cert_auth)
client_bob=client("Bob",cert_auth)
print("\nClients request their own certificates\n")
client_alice.get_your_certificate()
client_bob.get_your_certificate()
print("\nClients request the other guy's certificate\n")
cert_alice=client_bob.get_other_certificate("Alice")
cert_bob=client_alice.get_other_certificate("Bob")
pubk_alice=(int(cert_alice[0].split(",")[1]),int(cert_alice[0].split(",")[2]))
pubk_bob=(int(cert_bob[0].split(",")[1]),int(cert_bob[0].split(",")[2]))

print(f"\nClient Bob obtained Alice's public key:",pubk_alice,"\n")
print(f"\nClient Alice obtained Bob's public key:",pubk_bob,"\n")

print("\nClients exchange messages")
messages_from_a = ["Hello1", "Hello2", "Hello3"]
messages_from_b = ["ACK1", "ACK2", "ACK3"]

client_alice.send_msg(client_alice,"Hello1",pubk_bob,client_bob)
client_bob.receive_msg()
client_alice.send_msg(client_alice,"Hello2",pubk_bob,client_bob)
client_bob.receive_msg()
client_alice.send_msg(client_alice,"Hello3",pubk_bob,client_bob)
client_bob.receive_msg()

client_bob.send_msg(client_bob,"ACK1",pubk_alice,client_alice)
client_alice.receive_msg()
client_bob.send_msg(client_bob,"ACK2",pubk_alice,client_alice)
client_alice.receive_msg()
client_bob.send_msg(client_bob,"ACK3",pubk_alice,client_alice)
client_alice.receive_msg()

print("\nProgram Executed Successfully\n")
```

Output



```
Certificate ID matches actual ID
Valid Other-Cert received from CA.
Alice,120395940388463755931763969277731236368244325054745259682597866426570706549544120780645715495439657254791228603600444629480440616129401051866061357024896744468165504270967761203356821162537814675780
No Tampering Detected
Certificate has not expired
Certificate ID matches actual ID
Valid Other-Cert received from CA.
Bob,93758473074773659034131717698595249771768600646002763956331244532287818749844464654391782222798895431520896407254527841268639010391184666386610275497316366543640583655941957759894467665408244045049169

Client Bob obtained Alice's public key: (1203959403884637559317639692777312363682443250547452596825978664265707065495441207806457154954396572547912286036004446294804406161294010518660613570248967444681655


Client Alice obtained Bob's public key: (9375847307477365903413171769859524977176860064600276395633124453228781874984446465439178222279889543152089640725452784126863901039118466638661027549731636654364058


Clients exchange messages
The follwing message is being sent from Alice to Bob
Encrypted Message: 34934709489465723530307611596226682205287757489463260246294064131228167455204884462516413885003851999577947081154116493052367035367566621061215494427204606851889962255066485065995174121
The following message was received by Bob from Alice
Decrypted Message: Hello1
The follwing message is being sent from Alice to Bob
Encrypted Message: 35114502763955342282538127039773705401619410240022300100624877574534466941977580021018210348864769200939015650106812738974137172786764314280112176624621323803955730683326534616377691793
The following message was received by Bob from Alice
Decrypted Message: Hello2
The follwing message is being sent from Alice to Bob
Encrypted Message: 25759413800307374978233077361465709028866202865244851120404897003840857888118886144055019902834221914798436310468886301572440598649472605151183644465549463883946718421752623173500670898
The following message was received by Bob from Alice
Decrypted Message: Hello3
The follwing message is being sent from Bob to Alice
Encrypted Message: 94570708588084511557979965512536734204956569841587305395395679665284966103336688293845765865544562872649172584710745783498755790195847663701385011846211832902784683711656600693559511721
The following message was received by Alice from Bob
Decrypted Message: ACK1
The follwing message is being sent from Bob to Alice
Encrypted Message: 11275712840636429097724567802520038865069098284844675422630676289963968328156188492342408975887956688950517903514079060620968710036868206075437120136508320195318131023790528949435866896
The following message was received by Alice from Bob
Decrypted Message: ACK2
The follwing message is being sent from Bob to Alice
Encrypted Message: 46089504455093843473711051516682989346759816537811907851739370407937497805938604072076707190252158083535279971609140652832871606331203043976918254438327898304435093157902173167371849656
The following message was received by Alice from Bob
Decrypted Message: ACK3

Program Executed Successfully
```