Message Encryption and Transmission with DES, RSA, and PKC

Sam Pickell (Leader), Aishwarya Vissom

COMP 5610 | April 29, 2020

Abstract

With modern privacy and security concerns being what they are, one would assume that most, if not all, methods of messaging would have some form of encryption. However, a shocking amount of text on the internet is stored and transported in plaintext, leaving it vulnerable to being easily compromised. Our messaging program not only encrypts plaintext messages, but these encrypted messages are securely transported end-to-end.

Network and Security Building Blocks

The program exists in two files: a client and a server. Upon connecting to each other, the client and server will exchange public keys, generated using RSA. In this implementation, the client and server do not allow the user to use their own public keys, but this could easily be changed in a future release. Once the keys have been exchanged, the client generates a secret key (to be used as a session key) using the ANSI x9.17 standard, encrypts it using the server's public key, and sends the encrypted key to the server. The server decrypts the secret key with its private key, and both the client and server will now use this key as part of the message encryption process, all done without using prior secrets (PKC). Messages can now be securely sent, and each message that is sent is encrypted using DES using the secret key as the key used for encryption.

Accomplishments and Goals

The main goal of creating a messaging program that can encrypt messages and send them securely between two users has been accomplished. For future work, we have a couple of ideas that we could implement. The first is unifying the program. It currently exists as a "client" program and a "server" program. Ideally, we would want to make a single program like Facebook Messenger where the user does not need to decide to be the server or client, and instead can just use the program and connect to another user using the program. The other area for improvement would be in the UI. Right now, the program is a simple terminal program and is functionally just fine. But Python does have plenty of libraries that allow for a proper GUI, and a better looking program would be a lot more intuitive for the average person to use.

System Framework and Programming Platforms

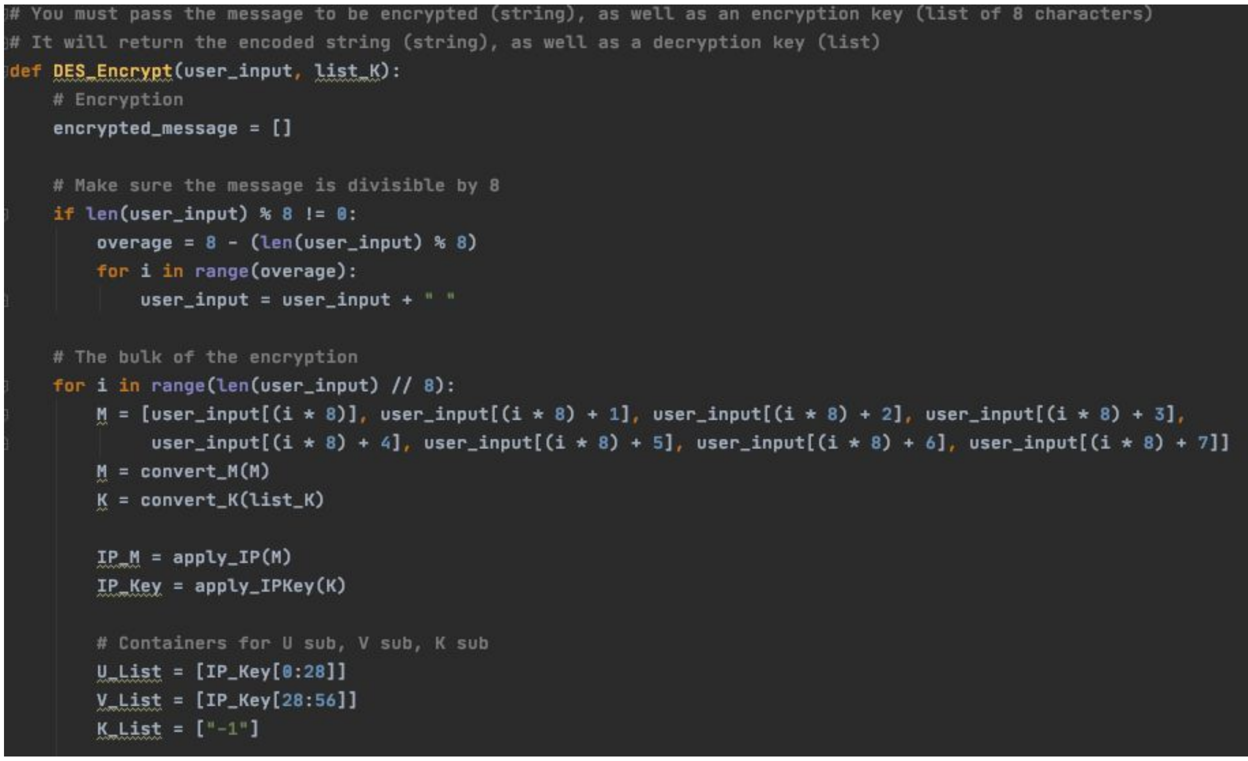
Our program is written in Python 3. The program has been written from scratch, using the algorithms and other information provided in our textbook. Certain libraries have been imported (such as the Python Cryptodome library for RSA or the Python socket library), but apart from those, no open-source code was used.

Member Contributions

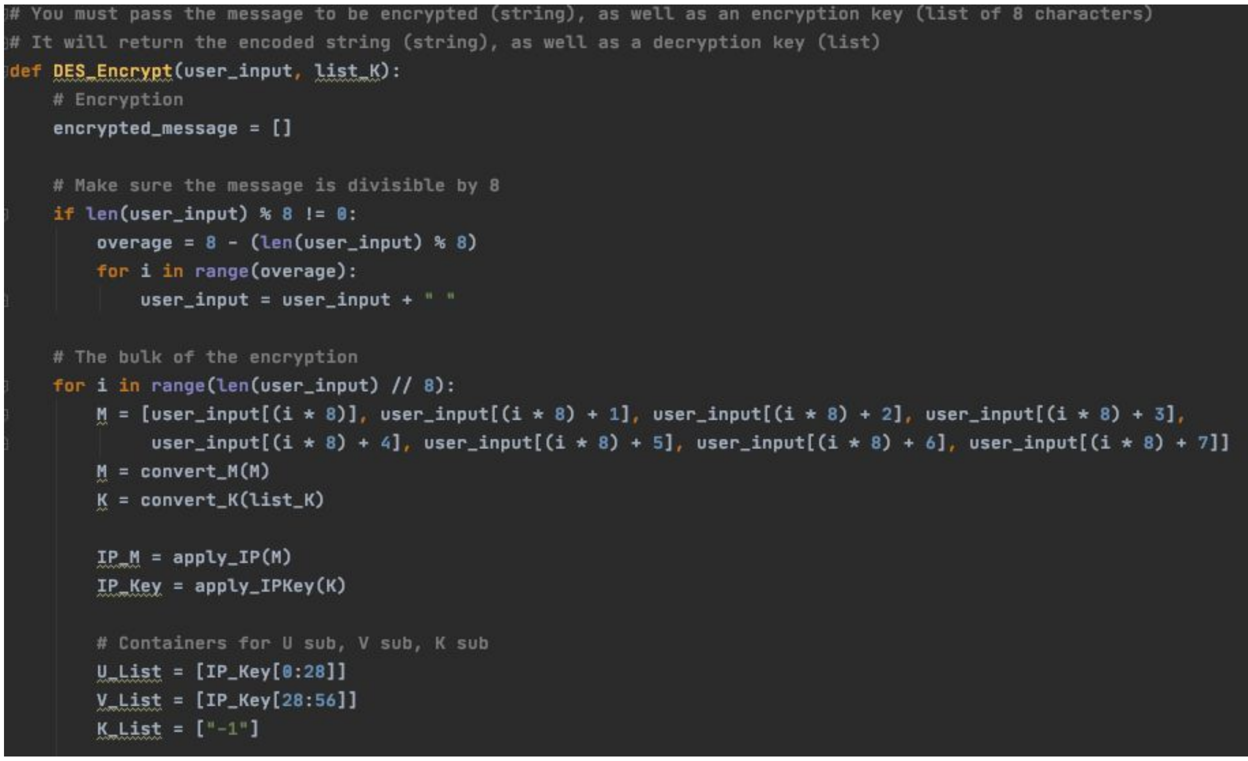
The workload of the project was divided up evenly between the two of us. Aishwarya implemented the RSA (from Crypto), PKC, and client/server (from socket) components, and Sam programmed the ANSI x9.17 standard and DES components. We both agree that we each contributed 50% of the work to complete this project.

Sample Code

- DES Encrypt

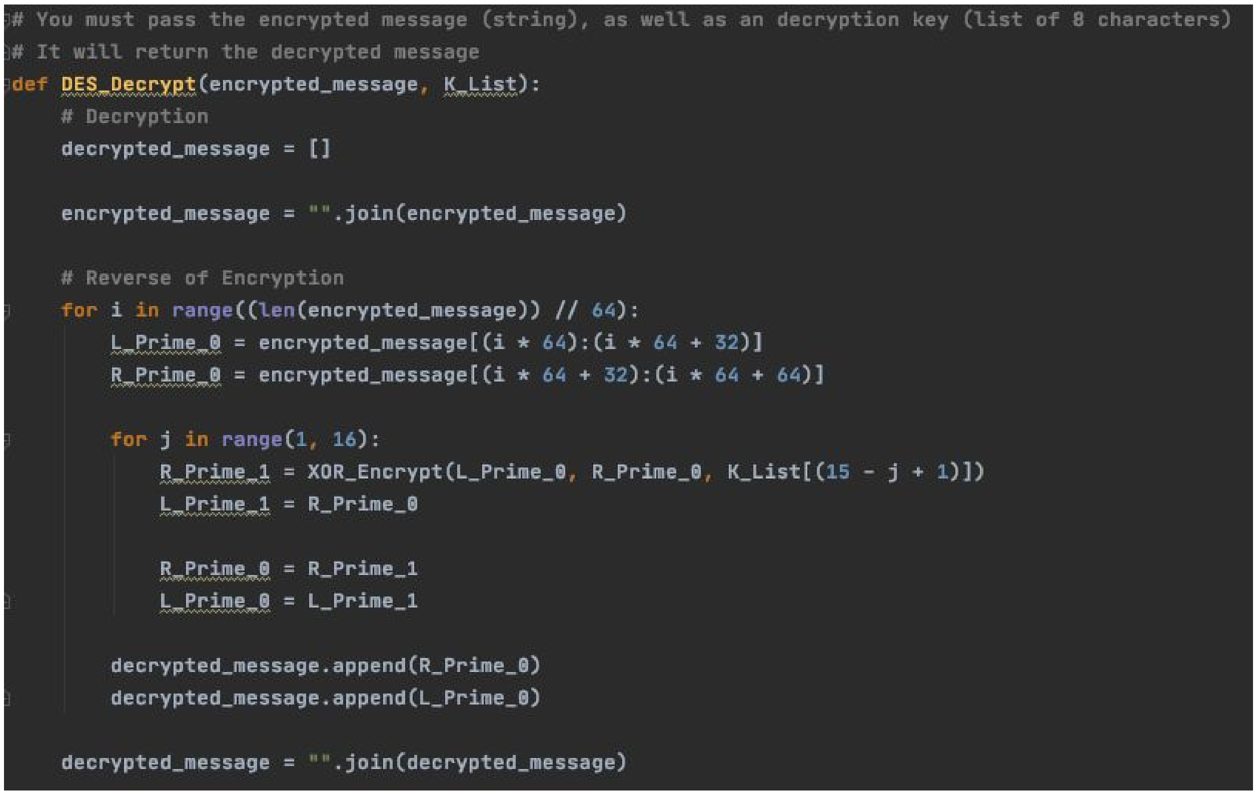


```python
# You must pass the message to be encrypted (string), as well as an encryption key (list of 8 characters)
# It will return the encoded string (string), as well as a decryption key (list)
def DES_Encrypt(user_input, list_K):
    # Encryption
    encrypted_message = []

    # Make sure the message is divisible by 8
    if len(user_input) % 8 != 0:
        overage = 8 - (len(user_input) % 8)
        for i in range(overage):
            user_input = user_input + " "

    # The bulk of the encryption
    for i in range(len(user_input) // 8):
        M = [user_input[(i * 8)], user_input[(i * 8) + 1], user_input[(i * 8) + 2], user_input[(i * 8) + 3],
             user_input[(i * 8) + 4], user_input[(i * 8) + 5], user_input[(i * 8) + 6], user_input[(i * 8) + 7]]
        M = convert_M(M)
        K = convert_K(list_K)

        IP_M = apply_IP(M)
        IP_Key = apply_IPKey(K)

        # Containers for U sub, V sub, K sub
        U_List = [IP_Key[0:28]]
        V_List = [IP_Key[28:56]]
        K_List = ["-1"]
```

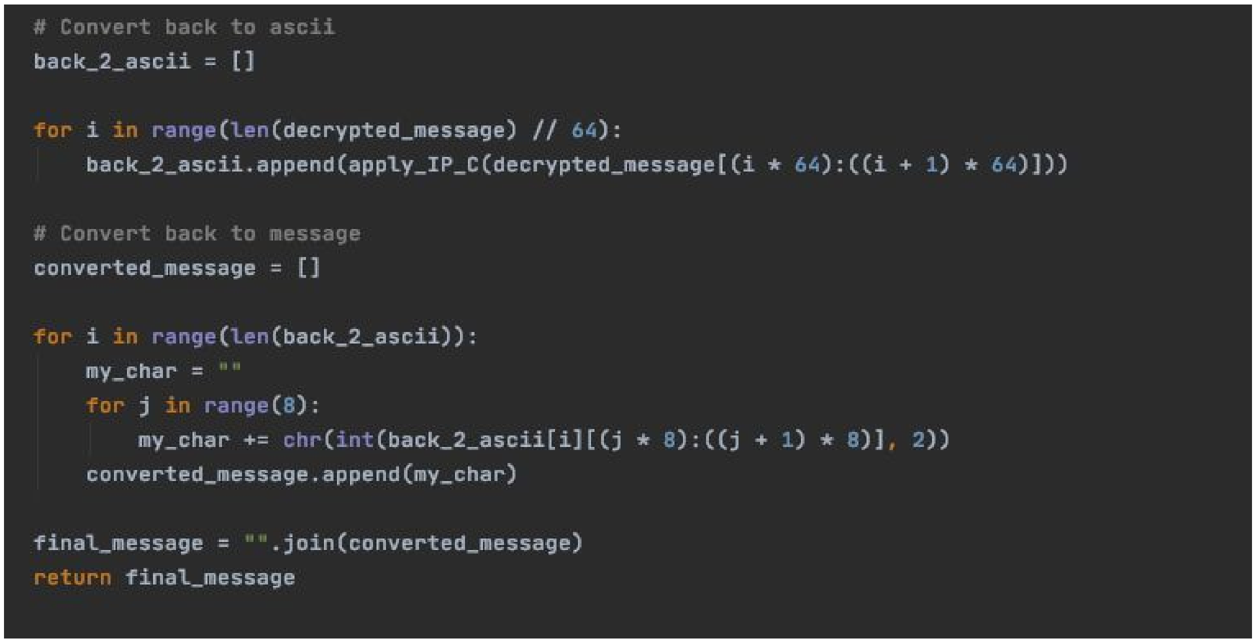


```python
# You must pass the message to be encrypted (string), as well as an encryption key (list of 8 characters)
# It will return the encoded string (string), as well as a decryption key (list)
def DES_Encrypt(user_input, list_K):
    # Encryption
    encrypted_message = []

    # Make sure the message is divisible by 8
    if len(user_input) % 8 != 0:
        overage = 8 - (len(user_input) % 8)
        for i in range(overage):
            user_input = user_input + " "

    # The bulk of the encryption
    for i in range(len(user_input) // 8):
        M = [user_input[(i * 8)], user_input[(i * 8) + 1], user_input[(i * 8) + 2], user_input[(i * 8) + 3],
             user_input[(i * 8) + 4], user_input[(i * 8) + 5], user_input[(i * 8) + 6], user_input[(i * 8) + 7]]
        M = convert_M(M)
        K = convert_K(list_K)

        IP_M = apply_IP(M)
        IP_Key = apply_IPKey(K)

        # Containers for U sub, V sub, K sub
        U_List = [IP_Key[0:28]]
        V_List = [IP_Key[28:56]]
        K_List = ["-1"]
```

- DES Decrypt

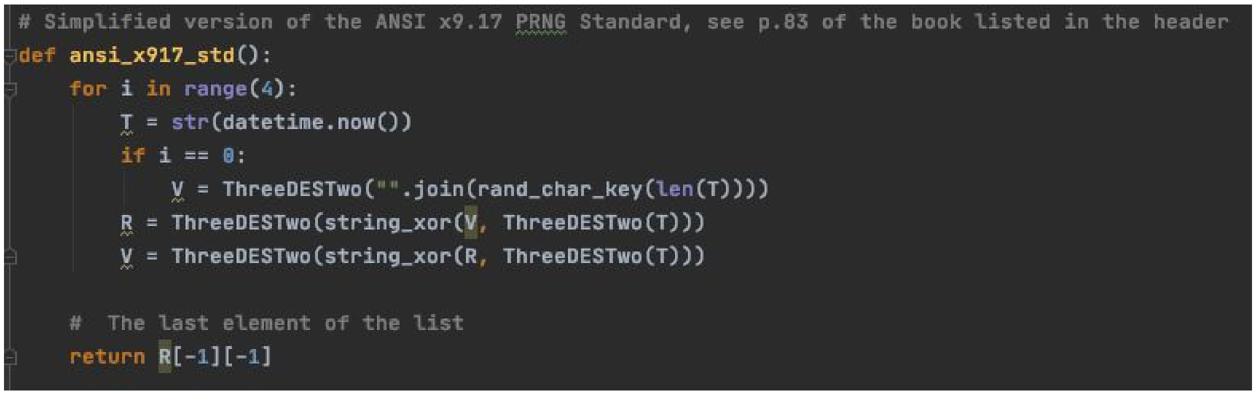


```python
# You must pass the encrypted message (string), as well as an decryption key (list of 8 characters)
# It will return the decrypted message
def DES_Decrypt(encrypted_message, K_List):
    # Decryption
    decrypted_message = []

    encrypted_message = "".join(encrypted_message)

    # Reverse of Encryption
    for i in range((len(encrypted_message)) // 64):
        L_Prime_0 = encrypted_message[(i * 64):(i * 64 + 32)]
        R_Prime_0 = encrypted_message[(i * 64 + 32):(i * 64 + 64)]

        for j in range(1, 16):
            R_Prime_1 = XOR_Encrypt(L_Prime_0, R_Prime_0, K_List[(15 - j + 1)])
            L_Prime_1 = R_Prime_0

            R_Prime_0 = R_Prime_1
            L_Prime_0 = L_Prime_1

        decrypted_message.append(R_Prime_0)
        decrypted_message.append(L_Prime_0)

    decrypted_message = "".join(decrypted_message)
```



```python
# Convert back to ascii
back_2_ascii = []

for i in range(len(decrypted_message) // 64):
    back_2_ascii.append(apply_IP_C(decrypted_message[(i * 64):((i + 1) * 64)]))

# Convert back to message
converted_message = []

for i in range(len(back_2_ascii)):
    my_char = ""
    for j in range(8):
        my_char += chr(int(back_2_ascii[i][(j * 8):((j + 1) * 8)], 2))
    converted_message.append(my_char)

final_message = "".join(converted_message)
return final_message
```

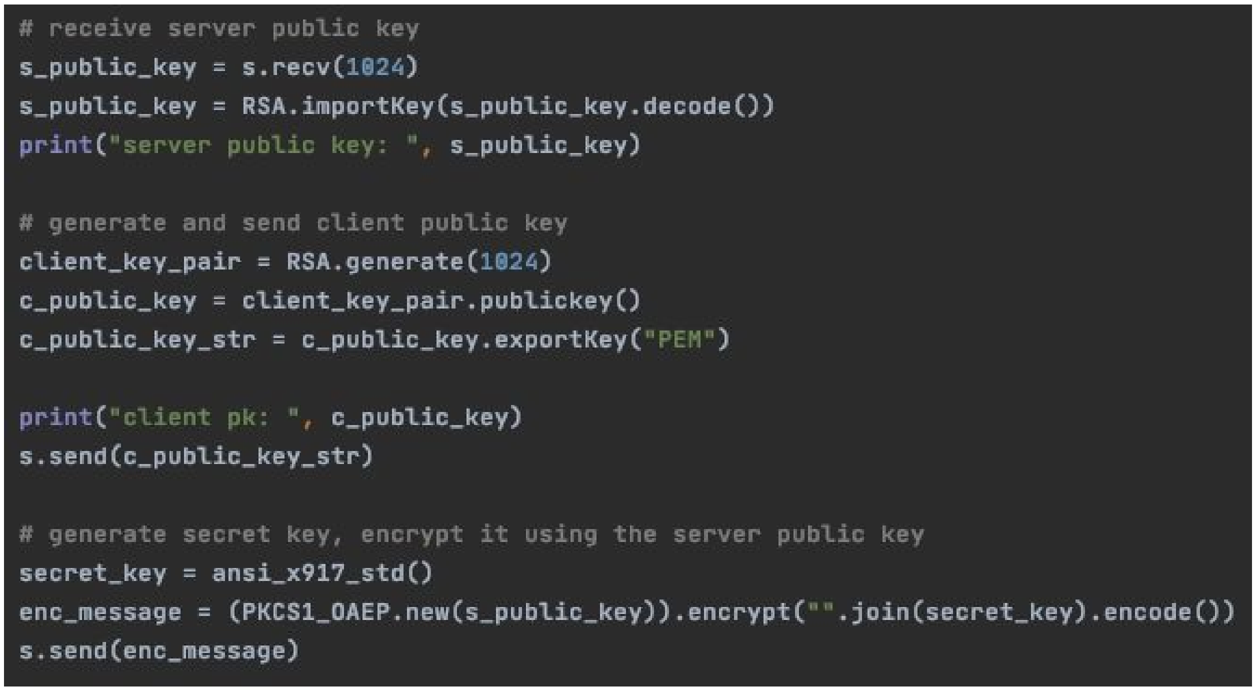
- ANSI x9.17 Standard



```python
# Simplified version of the ANSI x9.17 PRNG Standard, see p.83 of the book listed in the header
def ansi_x917_std():
    for i in range(4):
        T = str(datetime.now())
        if i == 0:
            V = ThreeDESTwo("".join(rand_char_key(len(T))))
        R = ThreeDESTwo(string_xor(V, ThreeDESTwo(T)))
        V = ThreeDESTwo(string_xor(R, ThreeDESTwo(T)))

    #  The last element of the list
    return R[-1][-1]
```

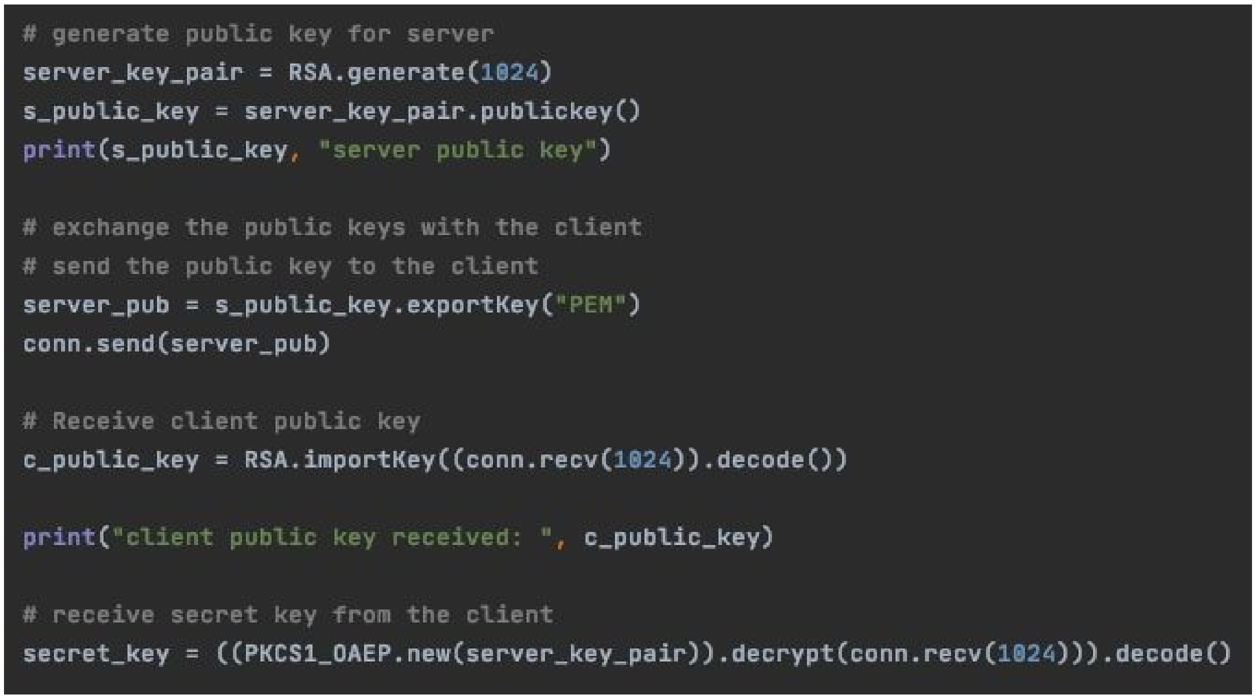
- PKC using RSA

  (Taken from client.py)



```python
# receive server public key
s_public_key = s.recv(1024)
s_public_key = RSA.importKey(s_public_key.decode())
print("server public key: ", s_public_key)

# generate and send client public key
client_key_pair = RSA.generate(1024)
c_public_key = client_key_pair.publickey()
c_public_key_str = c_public_key.exportKey("PEM")

print("client pk: ", c_public_key)
s.send(c_public_key_str)

# generate secret key, encrypt it using the server public key
secret_key = ansi_x917_std()
enc_message = (PKCS1_OAEP.new(s_public_key)).encrypt("".join(secret_key).encode())
s.send(enc_message)
```

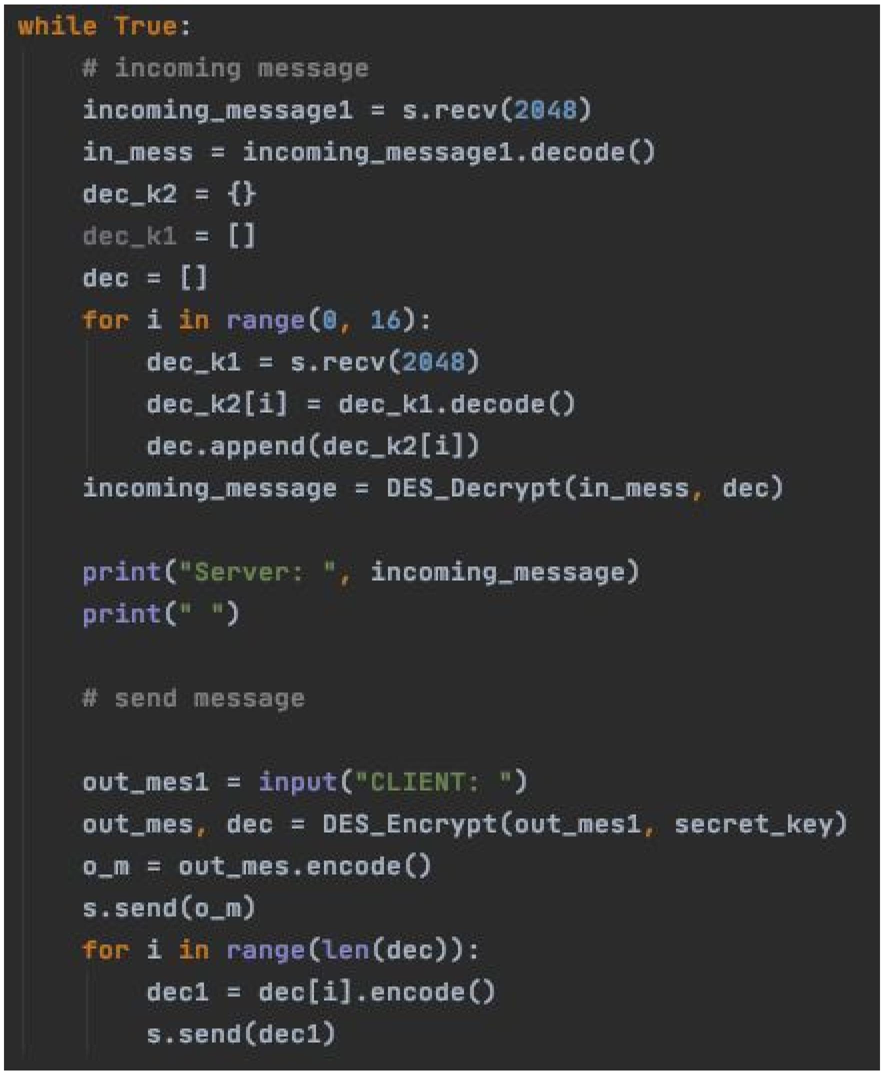
  (Taken from server.py)



```python
# generate public key for server
server_key_pair = RSA.generate(1024)
s_public_key = server_key_pair.publickey()
print(s_public_key, "server public key")

# exchange the public keys with the client
# send the public key to the client
server_pub = s_public_key.exportKey("PEM")
conn.send(server_pub)

# Receive client public key
c_public_key = RSA.importKey((conn.recv(1024)).decode())

print("client public key received: ", c_public_key)

# receive secret key from the client
secret_key = ((PKCS1_OAEP.new(server_key_pair)).decrypt(conn.recv(1024))).decode()
```

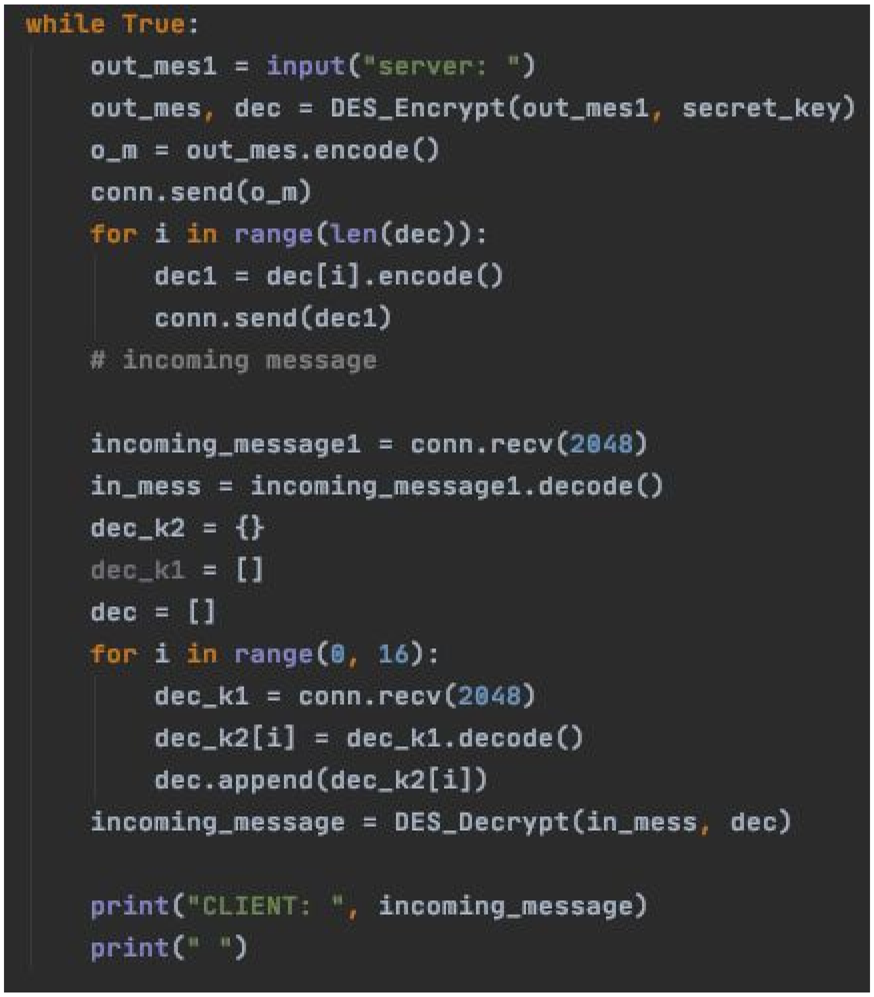
- Main loop of sending messages

  (Taken from client.py)



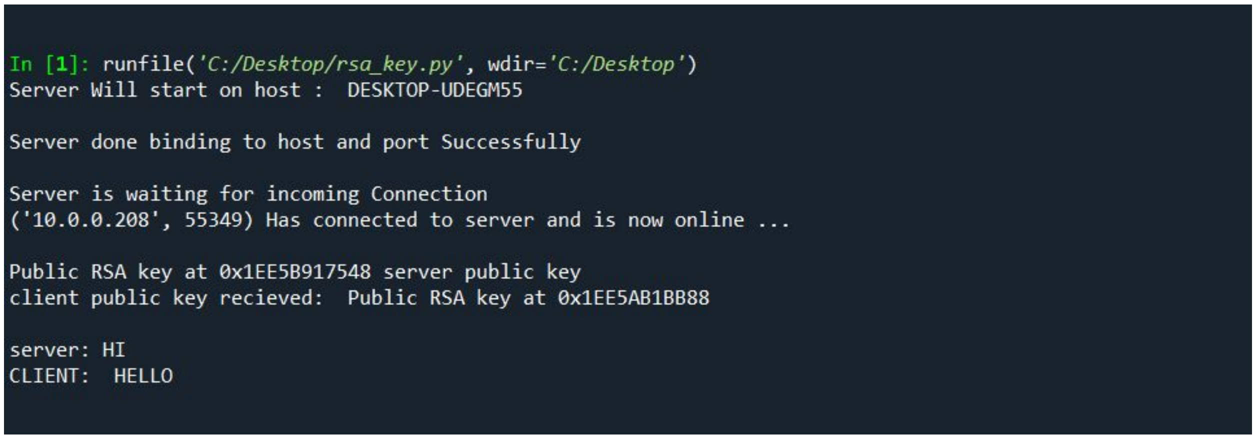
```python
while True:
    # incoming message
    incoming_message1 = s.recv(2048)
    in_mess = incoming_message1.decode()
    dec_k2 = {}
    dec_k1 = []
    dec = []
    for i in range(0, 16):
        dec_k1 = s.recv(2048)
        dec_k2[i] = dec_k1.decode()
        dec.append(dec_k2[i])
    incoming_message = DES_Decrypt(in_mess, dec)

    print("Server: ", incoming_message)
    print(" ")

    # send message

    out_mes1 = input("CLIENT: ")
    out_mes, dec = DES_Encrypt(out_mes1, secret_key)
    o_m = out_mes.encode()
    s.send(o_m)
    for i in range(len(dec)):
        dec1 = dec[i].encode()
        s.send(dec1)
```

(Taken from server.py)

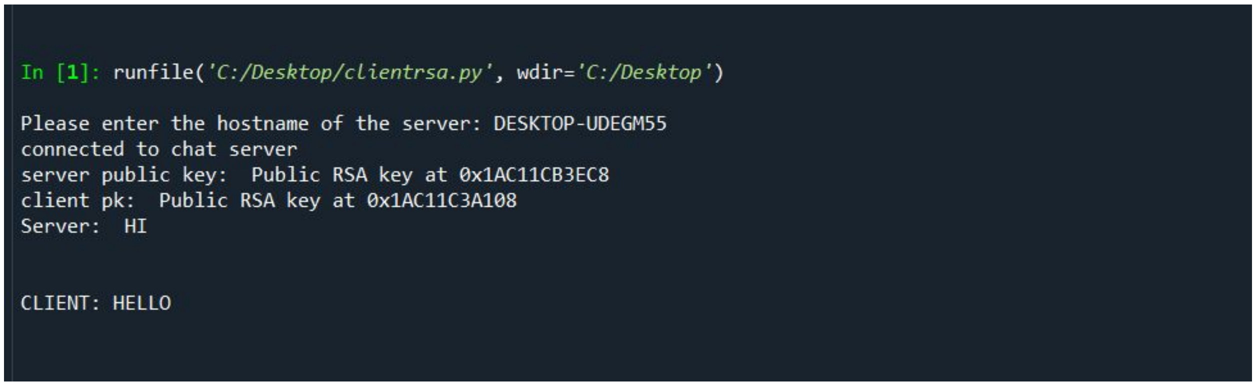


```
while True:
    out_mes1 = input("server: ")
    out_mes, dec = DES_Encrypt(out_mes1, secret_key)
    o_m = out_mes.encode()
    conn.send(o_m)
    for i in range(len(dec)):
        dec1 = dec[i].encode()
        conn.send(dec1)
    # incoming message

    incoming_message1 = conn.recv(2048)
    in_mess = incoming_message1.decode()
    dec_k2 = {}
    dec_k1 = []
    dec = []
    for i in range(0, 16):
        dec_k1 = conn.recv(2048)
        dec_k2[i] = dec_k1.decode()
        dec.append(dec_k2[i])
    incoming_message = DES_Decrypt(in_mess, dec)

    print("CLIENT: ", incoming_message)
    print(" ")
```

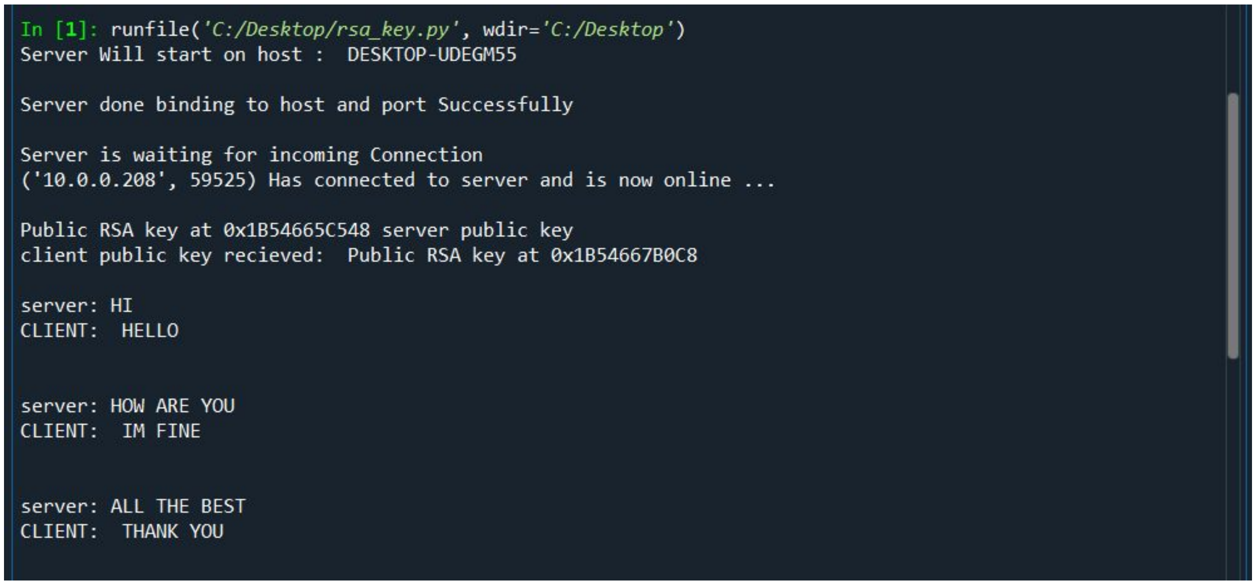
Snapshots of Running Samples



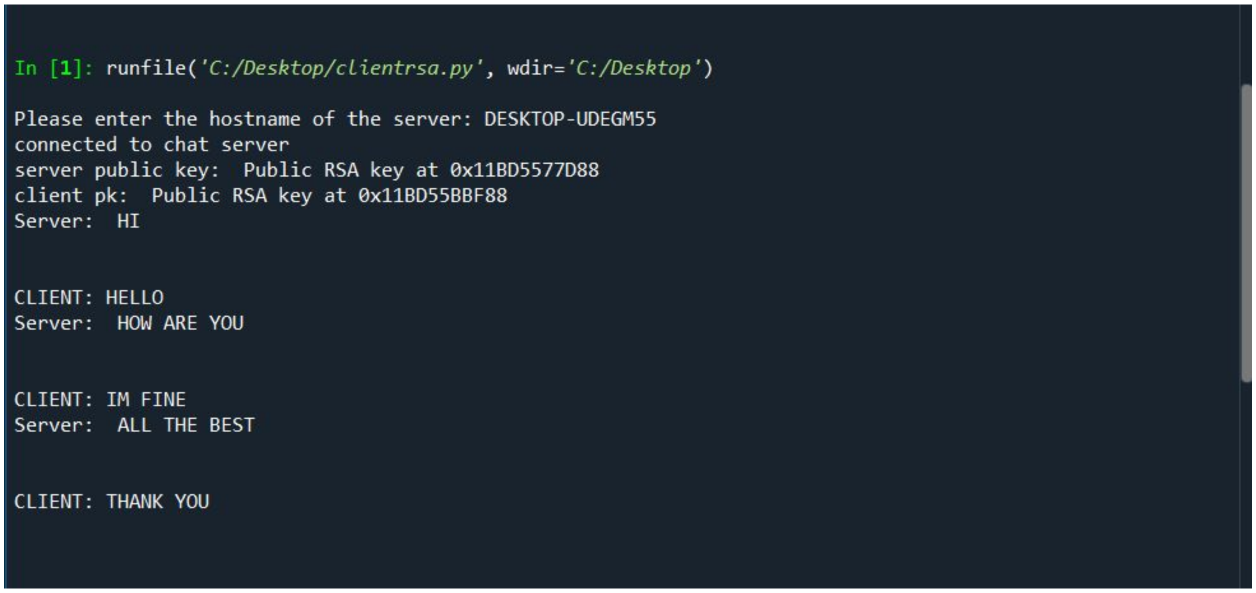
```
In [1]: runfile('C:/Desktop/rsa_key.py', wdir='C:/Desktop')
Server Will start on host :  DESKTOP-UDEGM55

Server done binding to host and port Successfully

Server is waiting for incoming Connection
('10.0.0.208', 55349) Has connected to server and is now online ...

Public RSA key at 0x1EE5B917548 server public key
client public key recieved:  Public RSA key at 0x1EE5AB1BB88

server: HI
CLIENT:  HELLO
```



```
In [1]: runfile('C:/Desktop/clientrsa.py', wdir='C:/Desktop')

Please enter the hostname of the server: DESKTOP-UDEGM55
connected to chat server
server public key:  Public RSA key at 0x1AC11CB3EC8
client pk:  Public RSA key at 0x1AC11C3A108
Server:  HI


CLIENT: HELLO
```



```
In [1]: runfile('C:/Desktop/rsa_key.py', wdir='C:/Desktop')
Server Will start on host :  DESKTOP-UDEGM55

Server done binding to host and port Successfully

Server is waiting for incoming Connection
('10.0.0.208', 59525) Has connected to server and is now online ...

Public RSA key at 0x1B54665C548 server public key
client public key recieved:  Public RSA key at 0x1B54667B0C8

server: HI
CLIENT:  HELLO


server: HOW ARE YOU
CLIENT:  IM FINE


server: ALL THE BEST
CLIENT:  THANK YOU
```



```
In [1]: runfile('C:/Desktop/clientrsa.py', wdir='C:/Desktop')

Please enter the hostname of the server: DESKTOP-UDEGM55
connected to chat server
server public key:  Public RSA key at 0x11BD5577D88
client pk:  Public RSA key at 0x11BD55BBF88
Server:  HI


CLIENT: HELLO
Server:  HOW ARE YOU


CLIENT: IM FINE
Server:  ALL THE BEST


CLIENT: THANK YOU
```