Instituto Politécnico Nacional

Escuela Superior de Cómputo

Selected Topics of Cryptography

Grupo: 7CM2

Lab Session 5: More advances of project 1

Profesora

Díaz Santiago Sandra

Alumnos

Hernández Reyes Julio César
Herrera Pelayo Carlos

Fecha de entrega: 31/05/2023

Exercises

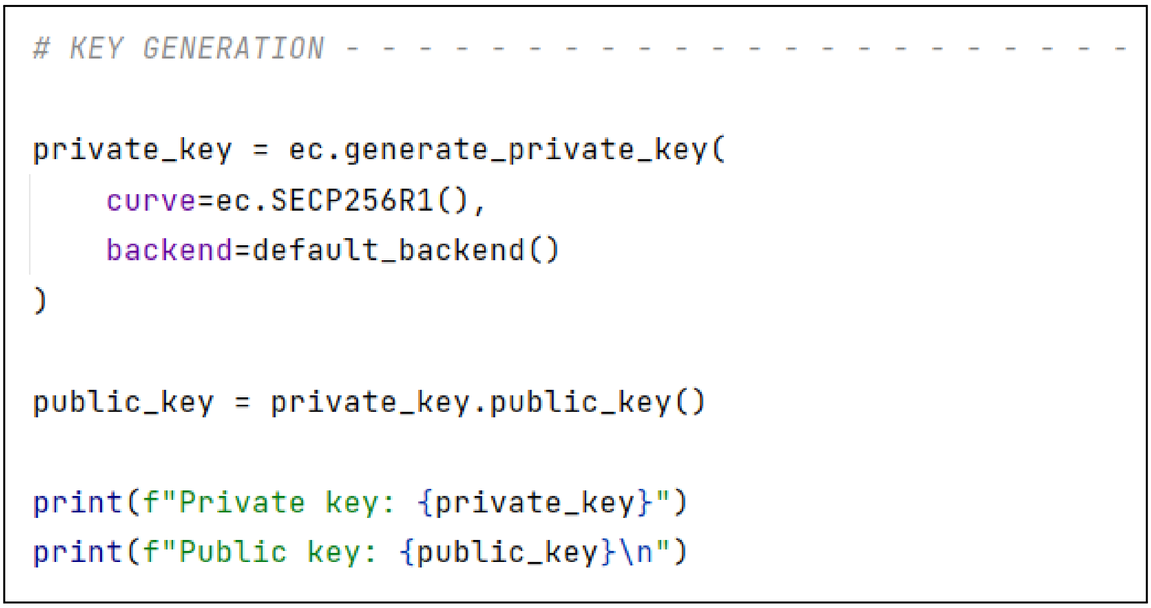
- Implement the key exchange protocol Diffie-Hellman on elliptic curves.
- Use the shared secret as a session-key to encipher the communication. Send messages using AES-128 and the mode of operation GCM.

Implementation

General

The following code is used for both the client and server.

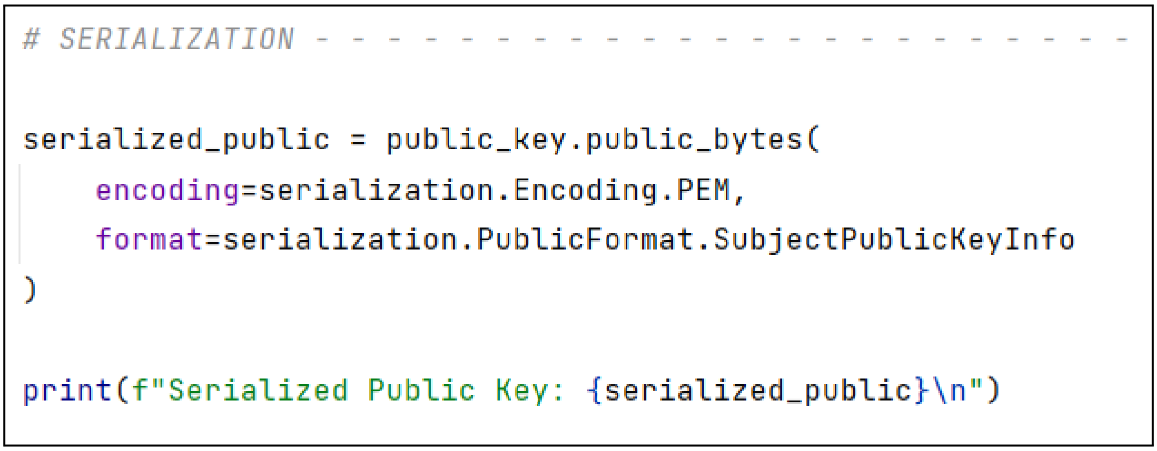
By selecting the NIST P-256 curve, we generate a private key for each peer. Next, we obtain the public key from the private key.



```python
# KEY GENERATION - - - - - - - - - - - - - - - - - - - - - - -

private_key = ec.generate_private_key(
    curve=ec.SECP256R1(),
    backend=default_backend()
)

public_key = private_key.public_key()

print(f"Private key: {private_key}")
print(f"Public key: {public_key}\n")
```

Generating the private and public keys

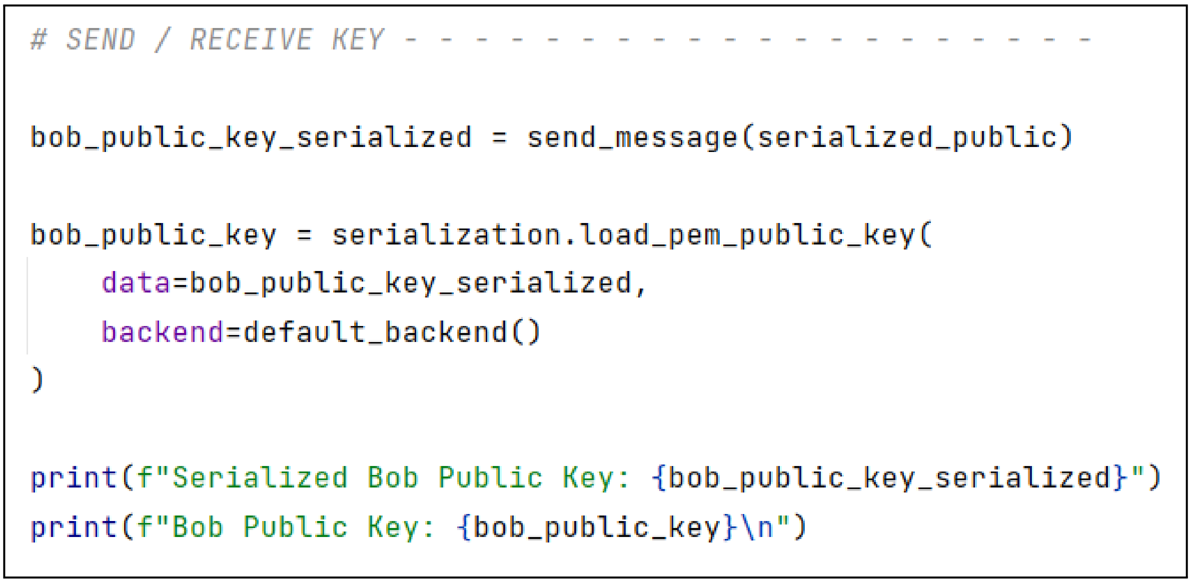
In order to share the public key, we need to save it using a standard format, like PEM. We also need to represent it as a byte stream so it can be sent across a network, that means we need to encapsulate the key.



```python
# SERIALIZATION - - - - - - - - - - - - - - - - - - - - - - - -

serialized_public = public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)

print(f"Serialized Public Key: {serialized_public}\n")
```

Serializing the key in PEM format

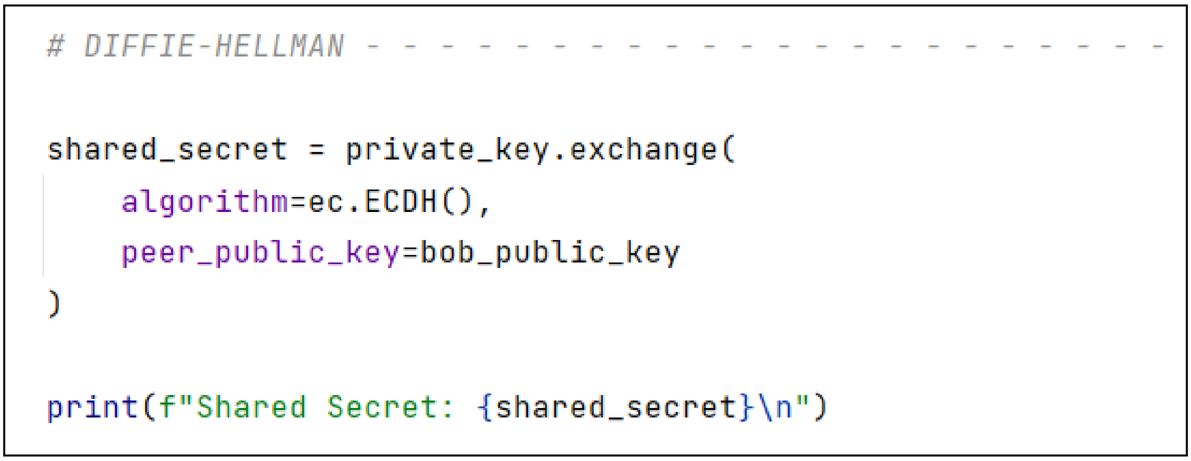
As we send our public key already serialized, we are receiving the peer's public key serialized. Knowing it is a key represented in PEM format, we need to decapsulate it.



```
# SEND / RECEIVE KEY - - - - - - - - - - - - - - - - - - - -

bob_public_key_serialized = send_message(serialized_public)

bob_public_key = serialization.load_pem_public_key(
    data=bob_public_key_serialized,
    backend=default_backend()
)

print(f"Serialized Bob Public Key: {bob_public_key_serialized}")
print(f"Bob Public Key: {bob_public_key}\n")
```

Receiving a serialized public key and recovering it

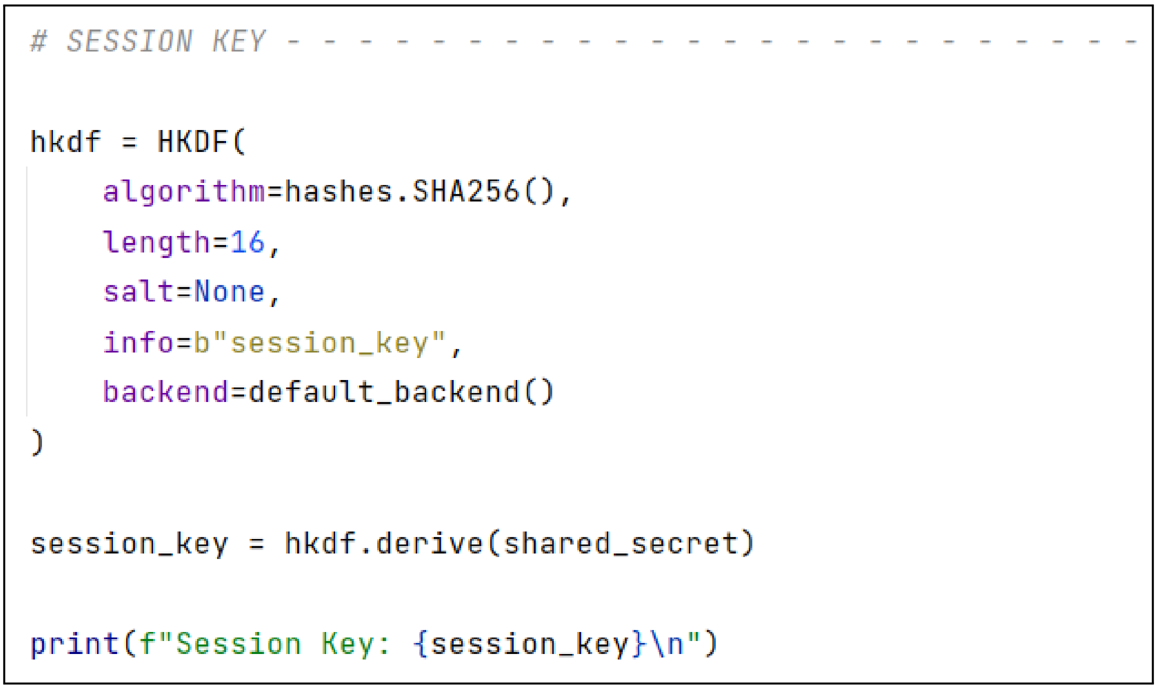
Now we are prepared to create a shared key.



```
# DIFFIE-HELLMAN - - - - - - - - - - - - - - - - - - - - - -

shared_secret = private_key.exchange(
    algorithm=ec.ECDH(),
    peer_public_key=bob_public_key
)

print(f"Shared Secret: {shared_secret}\n")
```

Diffie-Hellman operation on the received key

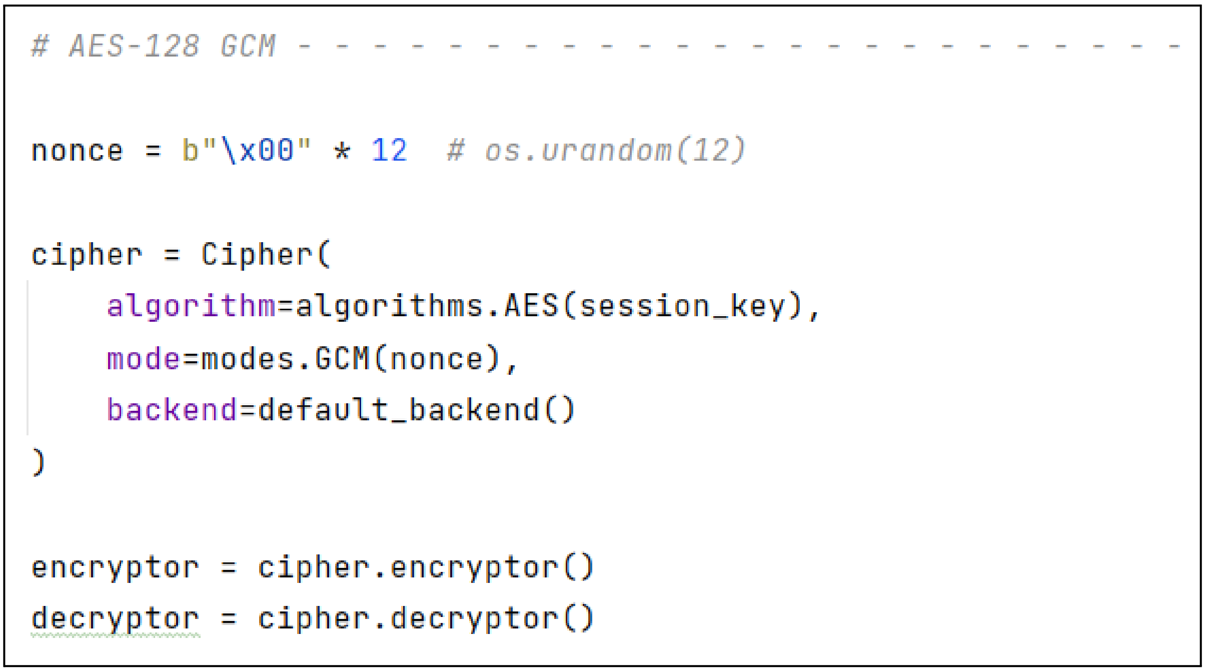
If we are going to use AES-128, we need to adjust the shared key to conform to the key length of 16 needed for the cipher, so we use a key derivation function to obtain what will be our session key.



```
# SESSION KEY - - - - - - - - - - - - - - - - - - - - - - - -

hkdf = HKDF(
    algorithm=hashes.SHA256(),
    length=16,
    salt=None,
    info=b"session_key",
    backend=default_backend()
)

session_key = hkdf.derive(shared_secret)

print(f"Session Key: {session_key}\n")
```

Adjusting the shared key by processing it with a key derivation function

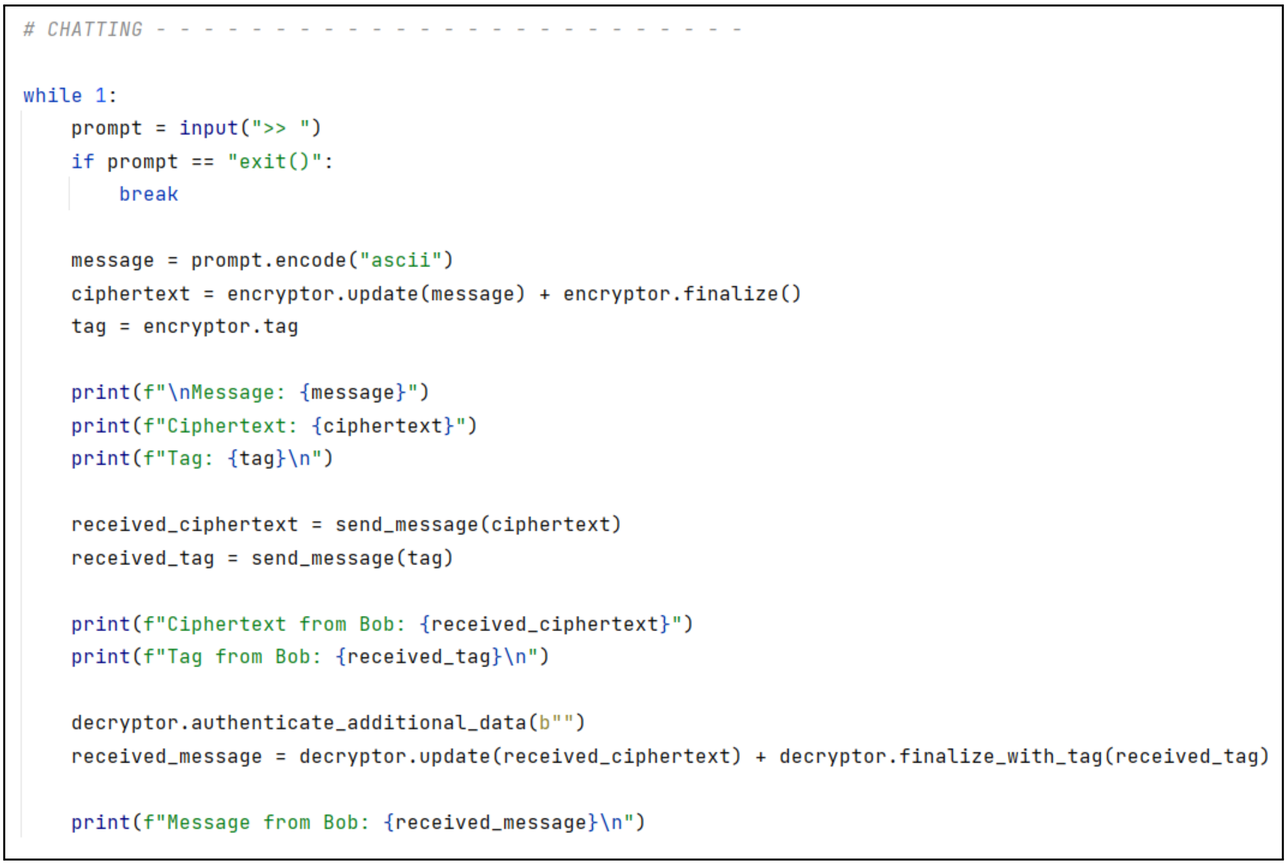
Using the mode GCM with AES-128, we need an initialization vector (nonce) of 12 bytes. Now the cipher is set up. Calling AES with a 16 bytes key and GCM with a 12 bytes nonce, we now produce an encryptor and decryptor.



```
# AES-128 GCM - - - - - - - - - - - - - - - - - - - - - - - -

nonce = b"\x00" * 12  # os.urandom(12)

cipher = Cipher(
    algorithm=algorithms.AES(session_key),
    mode=modes.GCM(nonce),
    backend=default_backend()
)

encryptor = cipher.encryptor()
decryptor = cipher.decryptor()
```

Creating a cipher object by specifying AES-128 mode GCM

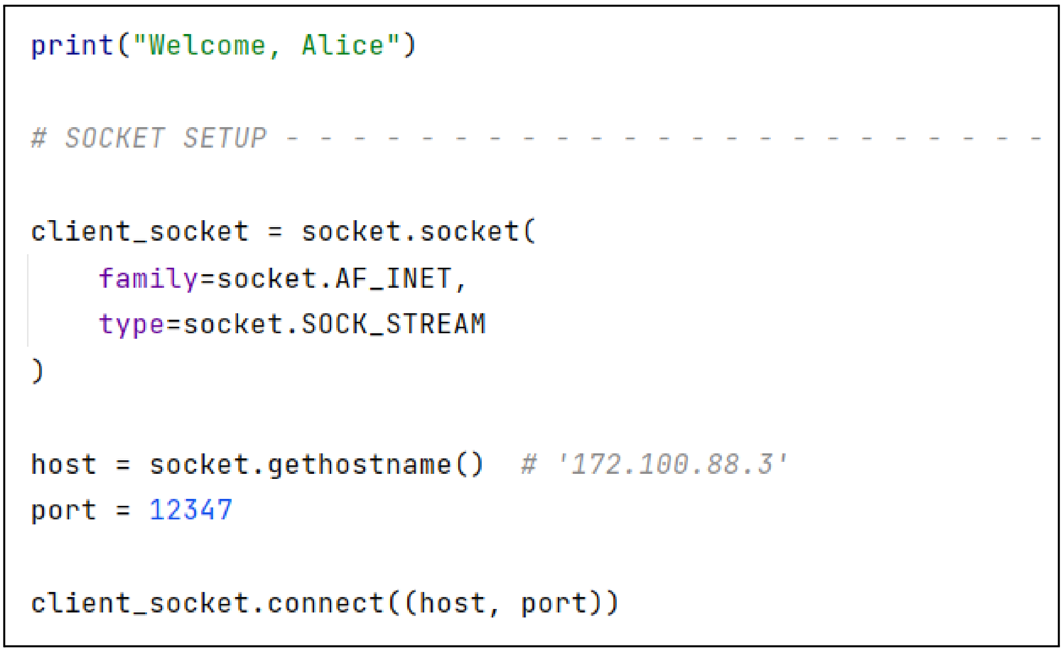
The chat prompts the user for a string message, encrypts it and sends it, along with the tag produced. A ciphertext is received with a tag, it is decrypted and printed in the console.



```python
# CHATTING - - - - - - - - - - - - - - - - - - - - - - - - -

while 1:
    prompt = input(">> ")
    if prompt == "exit()":
        break

    message = prompt.encode("ascii")
    ciphertext = encryptor.update(message) + encryptor.finalize()
    tag = encryptor.tag

    print(f"\nMessage: {message}")
    print(f"Ciphertext: {ciphertext}")
    print(f"Tag: {tag}\n")

    received_ciphertext = send_message(ciphertext)
    received_tag = send_message(tag)

    print(f"Ciphertext from Bob: {received_ciphertext}")
    print(f"Tag from Bob: {received_tag}\n")

    decryptor.authenticate_additional_data(b"")
    received_message = decryptor.update(received_ciphertext) + decryptor.finalize_with_tag(received_tag)

    print(f"Message from Bob: {received_message}\n")
```

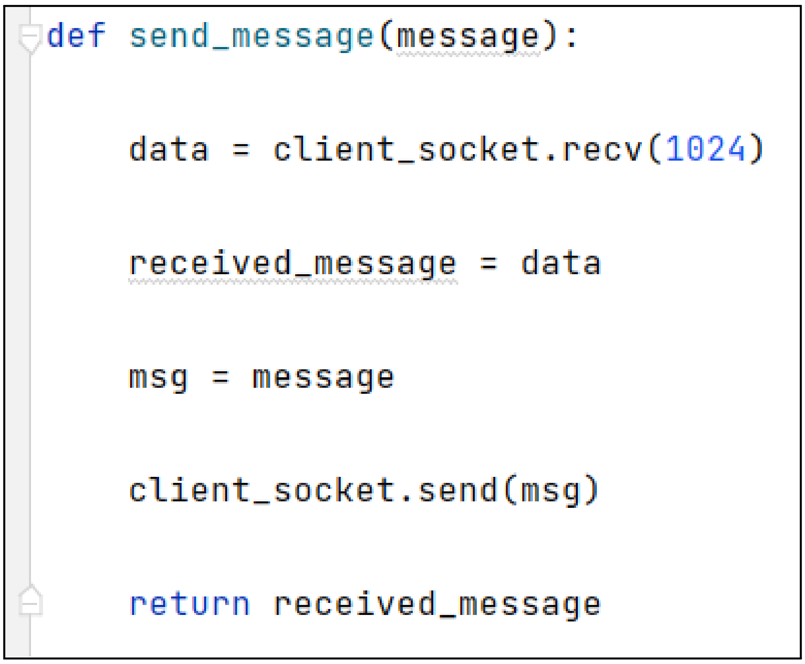
Chat that uses a session key to encrypt and decrypt messages

Client Side



```
print("Welcome, Alice")

# SOCKET SETUP - - - - - - - - - - - - - - - - - - - - - - -

client_socket = socket.socket(
    family=socket.AF_INET,
    type=socket.SOCK_STREAM
)

host = socket.gethostname()  # '172.100.88.3'
port = 12347

client_socket.connect((host, port))
```

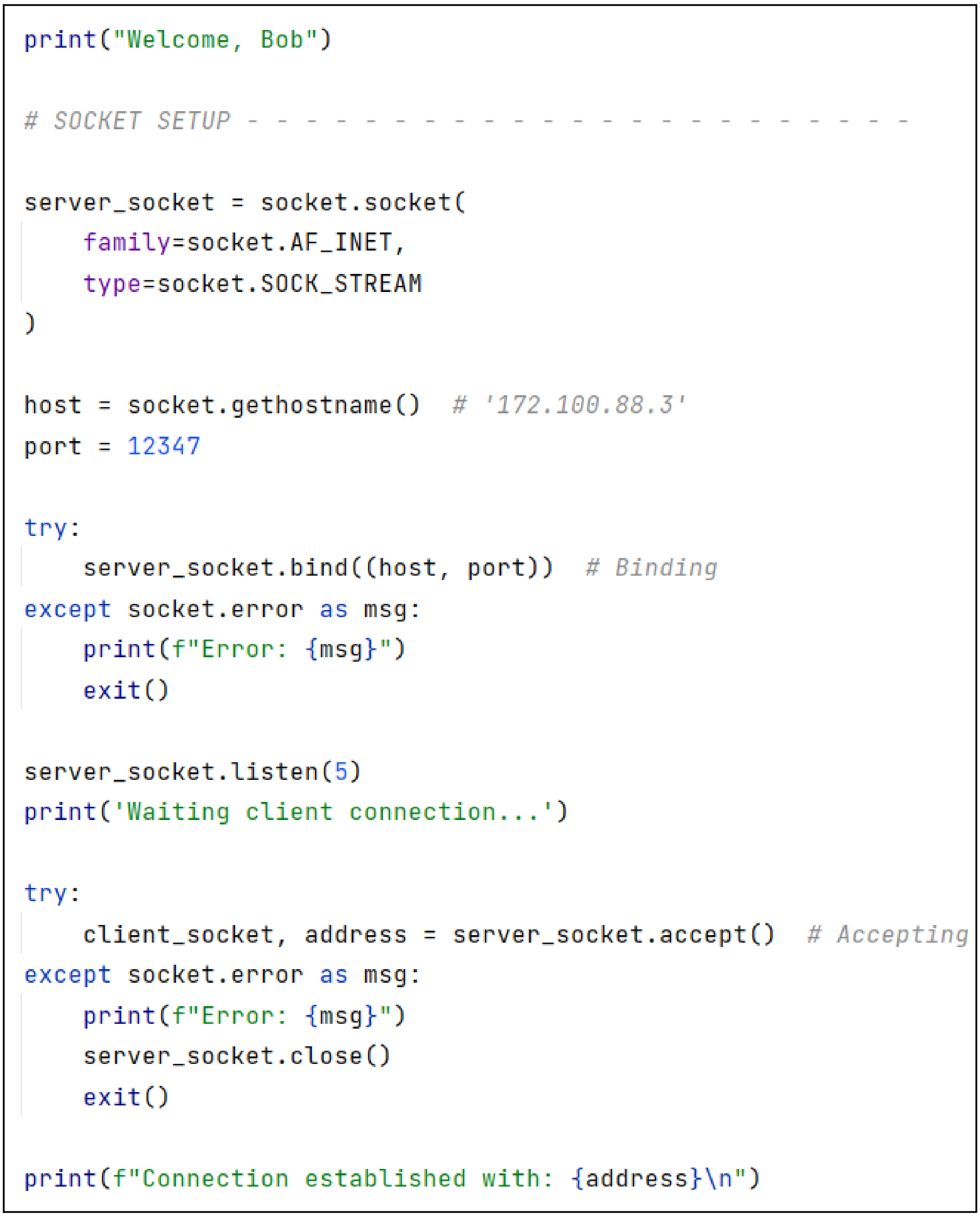
Socket setup.



```
def send_message(message):

    data = client_socket.recv(1024)

    received_message = data

    msg = message

    client_socket.send(msg)

    return received_message
```

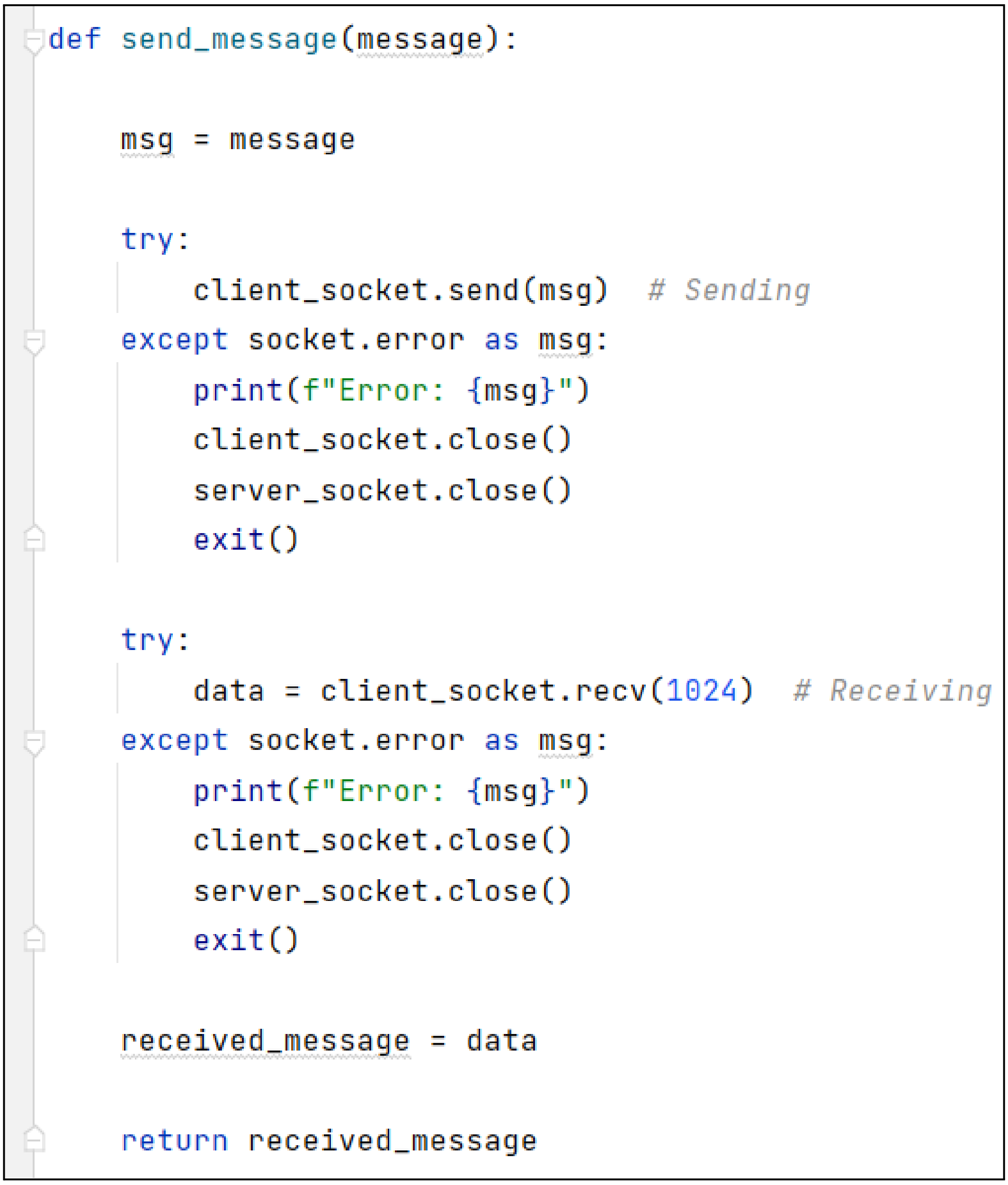
Sender and receiver function.

Client Side



```python
print("Welcome, Bob")

# SOCKET SETUP - - - - - - - - - - - - - - - - - - - - - - -

server_socket = socket.socket(
    family=socket.AF_INET,
    type=socket.SOCK_STREAM
)

host = socket.gethostname()  # '172.100.88.3'
port = 12347

try:
    server_socket.bind((host, port))  # Binding
except socket.error as msg:
    print(f"Error: {msg}")
    exit()

server_socket.listen(5)
print('Waiting client connection...')

try:
    client_socket, address = server_socket.accept()  # Accepting
except socket.error as msg:
    print(f"Error: {msg}")
    server_socket.close()
    exit()

print(f"Connection established with: {address}\n")
```

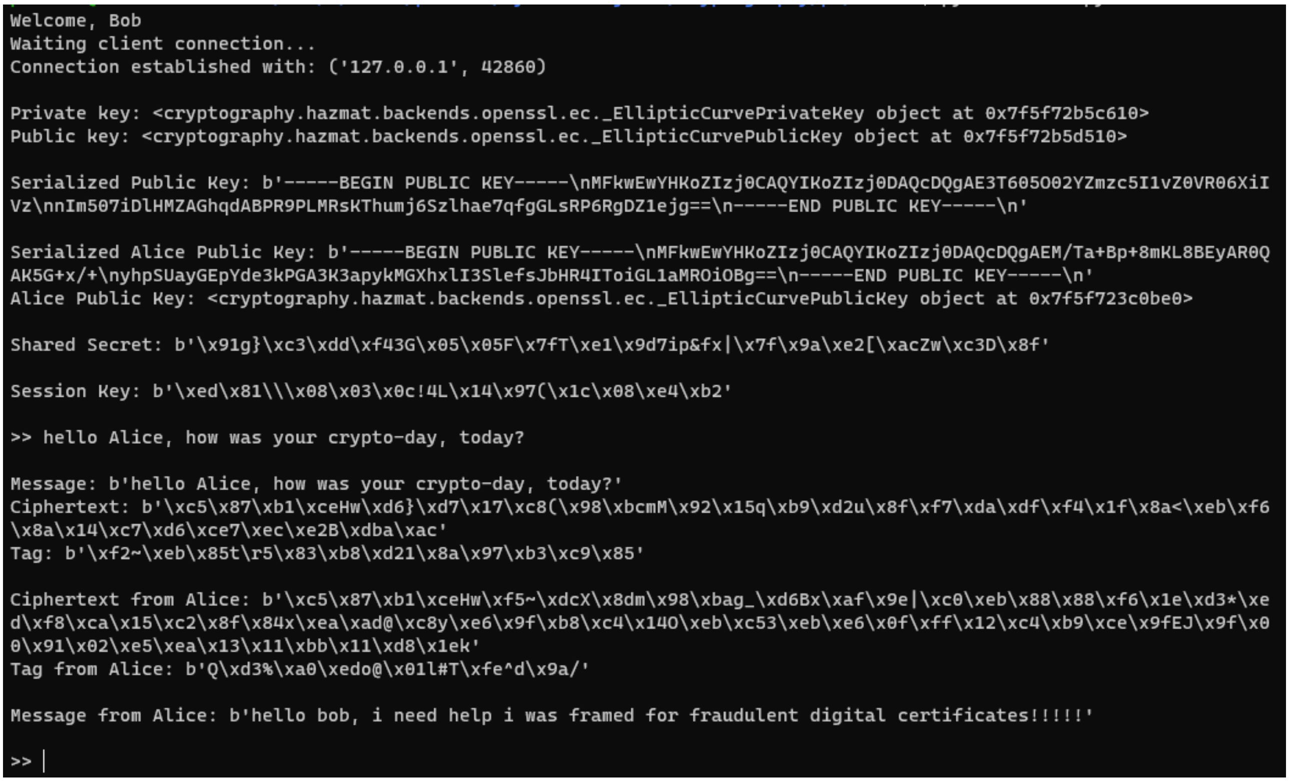
Socket setup.



```python
def send_message(message):

    msg = message

    try:
        client_socket.send(msg)  # Sending
    except socket.error as msg:
        print(f"Error: {msg}")
        client_socket.close()
        server_socket.close()
        exit()

    try:
        data = client_socket.recv(1024)  # Receiving
    except socket.error as msg:
        print(f"Error: {msg}")
        client_socket.close()
        server_socket.close()
        exit()

    received_message = data

    return received_message
```

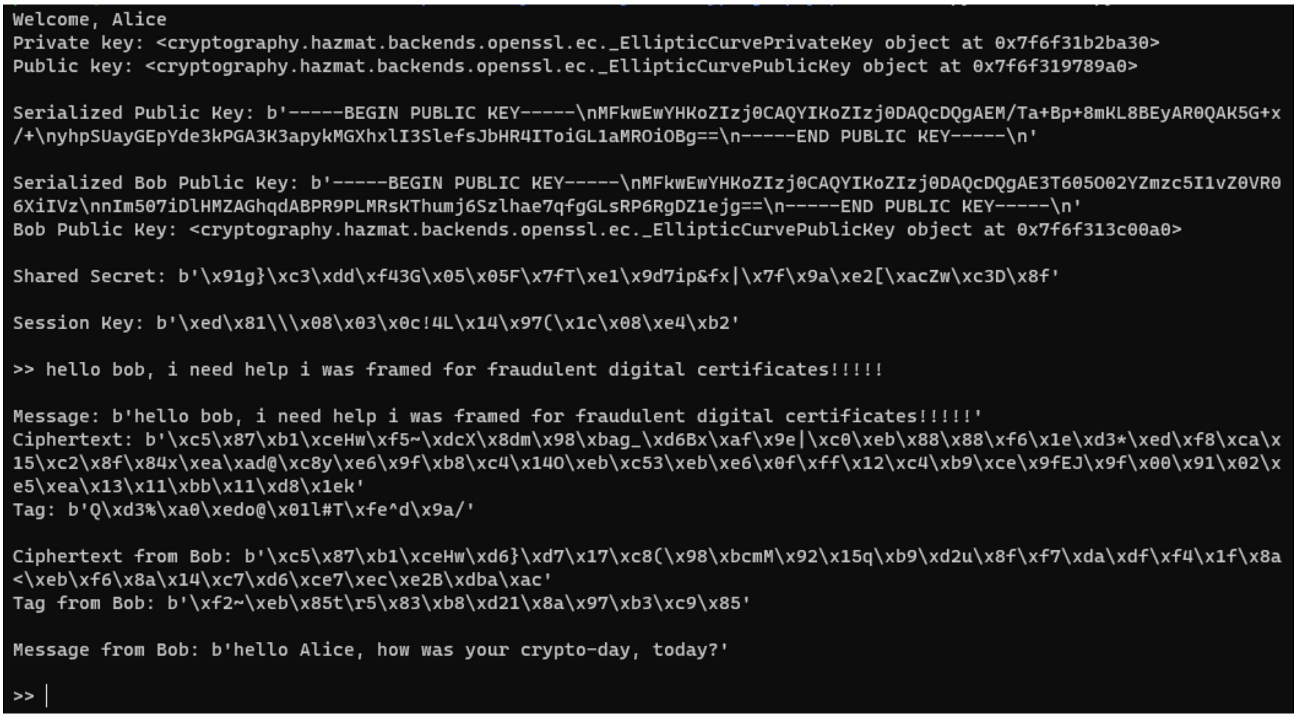
Sender and receiver function.

Output



```
Welcome, Bob
Waiting client connection...
Connection established with: ('127.0.0.1', 42860)

Private key: <cryptography.hazmat.backends.openssl.ec._EllipticCurvePrivateKey object at 0x7f5f72b5c610>
Public key: <cryptography.hazmat.backends.openssl.ec._EllipticCurvePublicKey object at 0x7f5f72b5d510>

Serialized Public Key: b'-----BEGIN PUBLIC KEY-----\nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE3T605O02YZmzc5I1vZ0VR06XiI
Vz\nnIm507iDlHMZAGhqdABPR9PLMRsKThumj6Szlhae7qfgGLsRP6RgDZ1ejg==\n-----END PUBLIC KEY-----\n'

Serialized Alice Public Key: b'-----BEGIN PUBLIC KEY-----\nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEM/Ta+Bp+8mKL8BEyAR0Q
AK5G+x/+\nyhpSUayGEpYde3kPGA3K3apykMGXhxlI3SlefsJbHR4IToiGL1aMROiOBg==\n-----END PUBLIC KEY-----\n'
Alice Public Key: <cryptography.hazmat.backends.openssl.ec._EllipticCurvePublicKey object at 0x7f5f723c0be0>

Shared Secret: b'\x91g}\xc3\xdd\xf43G\x05\x05F\x7fT\xe1\x9d7ip&fx|\x7f\x9a\xe2[\xacZw\xc3D\x8f'

Session Key: b'\xed\x81\\\x08\x03\x0c!4L\x14\x97(\x1c\x08\xe4\xb2'

>> hello Alice, how was your crypto-day, today?

Message: b'hello Alice, how was your crypto-day, today?'
Ciphertext: b'\xc5\x87\xb1\xceHw\xd6}\xd7\x17\xc8(\x98\xbcmM\x92\x15q\xb9\xd2u\x8f\xf7\xda\xdf\xf4\x1f\x8a<\xeb\xf6
\x8a\x14\xc7\xd6\xce7\xec\xe2B\xdba\xac'
Tag: b'\xf2~\xeb\x85t\r5\x83\xb8\xd21\x8a\x97\xb3\xc9\x85'

Ciphertext from Alice: b'\xc5\x87\xb1\xceHw\xf5~\xdcX\x8dm\x98\xbag_\xd6Bx\xaf\x9e|\xc0\xeb\x88\x88\xf6\x1e\xd3*\xe
d\xf8\xca\x15\xc2\x8f\x84x\xea\xad@\xc8y\xe6\x9f\xb8\xc4\x14O\xeb\xc53\xeb\xe6\x0f\xff\x12\xc4\xb9\xce\x9fEJ\x9f\x0
0\x91\x02\xe5\xea\x13\x11\xbb\x11\xd8\x1ek'
Tag from Alice: b'Q\xd3%\xa0\xedo@\x01l#T\xfe^d\x9a/'

Message from Alice: b'hello bob, i need help i was framed for fraudulent digital certificates!!!!!'

>> |
```

Server output



```
Welcome, Alice
Private key: <cryptography.hazmat.backends.openssl.ec._EllipticCurvePrivateKey object at 0x7f6f31b2ba30>
Public key: <cryptography.hazmat.backends.openssl.ec._EllipticCurvePublicKey object at 0x7f6f319789a0>

Serialized Public Key: b'-----BEGIN PUBLIC KEY-----\nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEM/Ta+Bp+8mKL8BEyAR0QAK5G+x
/+\nyhpSUayGEpYde3kPGA3K3apykMGXhxlI3SlefsJbHR4IToiGL1aMROiOBg==\n-----END PUBLIC KEY-----\n'

Serialized Bob Public Key: b'-----BEGIN PUBLIC KEY-----\nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE3T605O02YZmzc5I1vZ0VR0
6XiIVz\nnIm507iDlHMZAGhqdABPR9PLMRsKThumj6Szlhae7qfgGLsRP6RgDZ1ejg==\n-----END PUBLIC KEY-----\n'
Bob Public Key: <cryptography.hazmat.backends.openssl.ec._EllipticCurvePublicKey object at 0x7f6f313c00a0>

Shared Secret: b'\x91g}\xc3\xdd\xf43G\x05\x05F\x7fT\xe1\x9d7ip&fx|\x7f\x9a\xe2[\xacZw\xc3D\x8f'

Session Key: b'\xed\x81\\\x08\x03\x0c!4L\x14\x97(\x1c\x08\xe4\xb2'

>> hello bob, i need help i was framed for fraudulent digital certificates!!!!!

Message: b'hello bob, i need help i was framed for fraudulent digital certificates!!!!!'
Ciphertext: b'\xc5\x87\xb1\xceHw\xf5~\xdcX\x8dm\x98\xbag_\xd6Bx\xaf\x9e|\xc0\xeb\x88\x88\xf6\x1e\xd3*\xed\xf8\xca\x
15\xc2\x8f\x84x\xea\xad@\xc8y\xe6\x9f\xb8\xc4\x14O\xeb\xc53\xeb\xe6\x0f\xff\x12\xc4\xb9\xce\x9fEJ\x9f\x00\x91\x02\x
e5\xea\x13\x11\xbb\x11\xd8\x1ek'
Tag: b'Q\xd3%\xa0\xedo@\x01l#T\xfe^d\x9a/'

Ciphertext from Bob: b'\xc5\x87\xb1\xceHw\xd6}\xd7\x17\xc8(\x98\xbcmM\x92\x15q\xb9\xd2u\x8f\xf7\xda\xdf\xf4\x1f\x8a
<\xeb\xf6\x8a\x14\xc7\xd6\xce7\xec\xe2B\xdba\xac'
Tag from Bob: b'\xf2~\xeb\x85t\r5\x83\xb8\xd21\x8a\x97\xb3\xc9\x85'

Message from Bob: b'hello Alice, how was your crypto-day, today?'

>> |
```

Client output