Python socket encrypted communication

This report explains the rationale behind the encrypted data exchange in a client-server architecture. The architecture was implemented using Python and an external socket library that fully facilitates the connection between server and client. The select module also supports communication between multiple clients based on one server. Most importantly, every communication can be encrypted using Advanced Encryption Standard (AES) with a SHA hashing algorithm from SSL and TLS libraries.

Software Architecture

client.py
communication.py
main.py
server.py

Figure 1: Class modules

```python
def send(channel, *args):
    buffer = pickle.dumps(args)
    value = socket.htonl(len(buffer))
    size = struct.pack( fmt: "L",  *v: value)
    channel.send(size)
    channel.send(buffer)
```

```python
def receive(channel):
    size = struct.calcsize("L")
    size = channel.recv(size)
    try:
        size = socket.ntohl(struct.unpack( __format: "L", size)[0])
    except struct.error as e:
        return ''
    buf = ""
    while len(buf) < size:
        buf = channel.recv(size - len(buf))
    return pickle.loads(buf)[0]
```

Figure 2: Methods in communication class

The chat communication program consists of three (four) classes that can be observed from above. Both client and server models can be easily understood to support the communication establishment, data transmission, and encryption via the class initialization and running methods. There are a few helper functions such as retrieving client name and port. However, the core functionalities of data transmission are implemented in a separate class called communication that allows both client and server to send and receive messages via serialization and deserialization which also integrates SOLID principles. The main class can be neglected as it is only used for testing.

Encryption

```python
# Impose encryption
self.context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
self.context.load_cert_chain(certfile="cert.pem", keyfile="cert.pem")
self.context.load_verify_locations('cert.pem')
self.context.set_ciphers('AES128-SHA')
```

Figure 3: Server encryption method

```python
# Impose encryption
self.context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
self.context.set_ciphers('AES128-SHA')
```

Figure 4: Client encryption method

```python
self.server = self.context.wrap_socket(self.server, server_side=True)
```

```python
self.sock = self.context.wrap_socket(self.sock, server_hostname=host)
```

Figure 5: Wrap context and socket

The TLS 1.2 encryption is achieved by an external library that supports Secure Sockets Layer (SSL) and Transport Layer Security (TLS). By breaking down the encryption rationale, the first three lines of code in the server class are responsible for initialising SSL/TLS instances and verifying the digital certificate called cert.pem for authentication purposes. The next one is the actual cryptography that specifies the transmitted message that can be encrypted and integrated by a symmetric encryption approach, AES and a hashing function, SHA-1. Figure 5 illustrates that the client and server must wrap their context (message) and existing socket with SSL to enable message encryption.

Wireshark captures

101	31.582105	172.24.4.221	13.107.21.239	TLSv1.2	646 Client Hello
107	31.590703	13.107.21.239	172.24.4.221	TLSv1.2	159 Server Hello, Certificate, Certificate Status, Server Key Exchange, Server Hello Done
109	31.594804	172.24.4.221	13.107.21.239	TLSv1.2	212 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
110	31.594994	172.24.4.221	13.107.21.239	TLSv1.2	153 Application Data
111	31.595182	172.24.4.221	13.107.21.239	TLSv1.2	915 Application Data
115	31.601571	13.107.21.239	172.24.4.221	TLSv1.2	396 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
116	31.601571	13.107.21.239	172.24.4.221	TLSv1.2	123 Application Data
118	31.601752	172.24.4.221	13.107.21.239	TLSv1.2	92 Application Data
121	31.605008	13.107.21.239	172.24.4.221	TLSv1.2	92 Application Data

40	11.688625	172.24.4.221	52.64.217.27	TLSv1.2	700 Client Hello
42	11.727379	52.64.217.27	172.24.4.221	TLSv1.2	2974 Server Hello
44	11.728905	52.64.217.27	172.24.4.221	TLSv1.2	114 [TCP Previous segment not captured] , Ignored Unknown Record
48	11.730185	172.24.4.221	52.64.217.27	TLSv1.2	180 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
49	11.730820	172.24.4.221	52.64.217.27	TLSv1.2	153 Application Data
50	11.731320	172.24.4.221	52.64.217.27	TLSv1.2	4890 Application Data
51	11.731411	172.24.4.221	52.64.217.27	TLSv1.2	244 Application Data
53	11.766504	52.64.217.27	172.24.4.221	TLSv1.2	225 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
54	11.766504	52.64.217.27	172.24.4.221	TLSv1.2	132 Application Data
57	11.766772	172.24.4.221	52.64.217.27	TLSv1.2	92 Application Data
59	11.822274	52.64.217.27	172.24.4.221	TLSv1.2	1462 Application Data
60	11.822274	52.64.217.27	172.24.4.221	TLSv1.2	92 Application Data

Figure 5: Wireshark packages

Due to the communication between the two clients based on a server, Wireshark captured two sets of packages to handle secure server connection and data exchange. The prerequisite of message encryption was the safe connection between client and server that was achieved through client and server greetings packets (Client/Server Hello). Afterwards, another two packets indicated that the client encrypted using the public key extracted from the server certificate (cert.pem) and sent to the server followed by the identification of TLS handshake accomplishment. More specifically, one of the labels on these packages with “Encrypted Handshake Message” demonstrates the completion of hashing the exchanged message at which the message was transmitted and encrypted successfully.

Furthermore, the Client Hello packet sends a TLS packet with a random byte that generates the AES encryption key whereas the Server Hello packet sends back the public certificate. The Client Key Exchange packet represents the client sending a client key exchange message encrypted using the server public key extracted from the certificate. The cipher spec protocol tells the server that all the subsequent messages transmitted will be encrypted, followed by the signal of accomplishment via packet Encrypted Handshake Message. The last packet, New Session Ticket informs the clients and server that multiple communication can be implemented within one session.

Conclusion

Overall, the chat program supports associations among multiple clients while maintaining a safe and reliable message transmission that can also be observed via Wireshark. AES was introduced as symmetric cryptography that usually provides a more efficient speed and accessibility than other encryption techniques.