Client server network project

By

Group C

Submitted to

The University of Liverpool

MASTER-OF-COMPUTER-SCIENCE

*Software Development in Practice*

Word Count: XXX

09/Oct/2023

REPORT TITLE

Submitted to

The University of Liverpool

Word Count: XXX

09/Oct/2023

TABLE OF CONTENTS

Page

[Chapter 1. Introduction 2](#_Toc147156570)

[Chapter 2. Solution Overview 3](#_Toc147156571)

[2.1 Solution overview 3](#_Toc147156572)

[2.2 Technical Flow 4](#_Toc147156573)

[Chapter 3. Solution Design (Client) 5](#_Toc147156574)

[3.1 Solution Design (Server) 5](#_Toc147156575)

[3.2 Solution Design (Server) 5](#_Toc147156576)

[3.3 Error Handling 6](#_Toc147156577)

[3.4 Design Decisions 6](#_Toc147156578)

[Chapter 4. Testing 6](#_Toc147156579)

[4.1 Serialization Unit Testing 6](#_Toc147156580)

[4.2 Encryption Unit Testing 6](#_Toc147156581)

[4.3 Network Unit Testing 6](#_Toc147156582)

[Chapter 5. Conclusion 7](#_Toc147156583)

[REFERENCES 8](#_Toc147156584)

[APPENDICES 8](#_Toc147156585)

# Introduction

The objective of this project is creating a Client Server Network which once established can facilitate the transmission of encrypted and serialised files. Also, the server must have the seamless mechanism for decrypting and processing these files on the server side.

**SCOPE OF PROJECT**

The code and software architecture report will explain how Group C developed our Client Server Network and deployed it on to the local network.

The project needs to have three key functions:

1. Capable of Secure Communication between client and server.
2. Using Pickle – Serialising the data/encryption before transmission from client.
3. Decrypting the data and demonstrating the information in required format by printing it.

Purpose of Report

This report and the related code created by Group C will demonstrate the following:

* Understanding of network communication, creation, and manipulation of data structures
* Transmission of data between client and server,
* Implementation of advanced features such as data serialisation in various formats and encryption of text files.

The Group decided to break this project into four roles of Project Manager, Architect, Developer and Tester to reflect how this project would be managed as a real time project in industry. This will enable Group C to acquire the essential skills for designing and managing networked systems in real-world scenarios whilst gaining an understanding of how data security works and the importance of data privacy using encryption.

Team has a public repository on GitHub which was used throughout the project for collaboration and version control. Team’s public repository can be found using this link –

[https://github.com/callmehawa/Client\_Server\_Network\_GroupC.](https://github.com/callmehawa/Client_Server_Network_GroupC.git)git

Team Member -

# Solution Overview

Solution overview

Figure 1 is a high-level solution of a simple client-server network to perform data serialisation, encryption, and transfer from a server to a client.

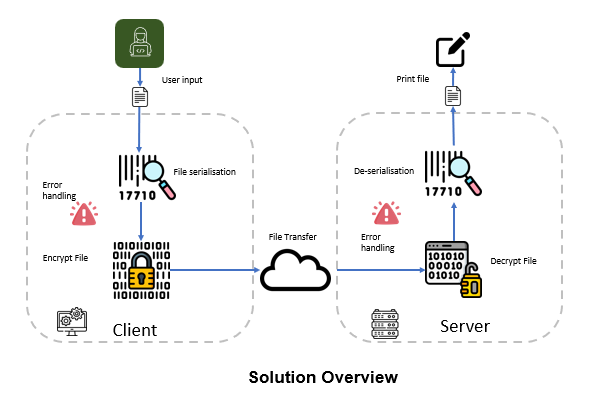


Figure 1. Solution overview

Technical Flow

The Technical Flow diagram (Figure 2) showcases the process flow of a Client-Server network that accomodates various data formats and encrypts the data to protect transmitted information. It also showcases server actions to accept the transmited data, de-seralise, unencrpt and print the received data.

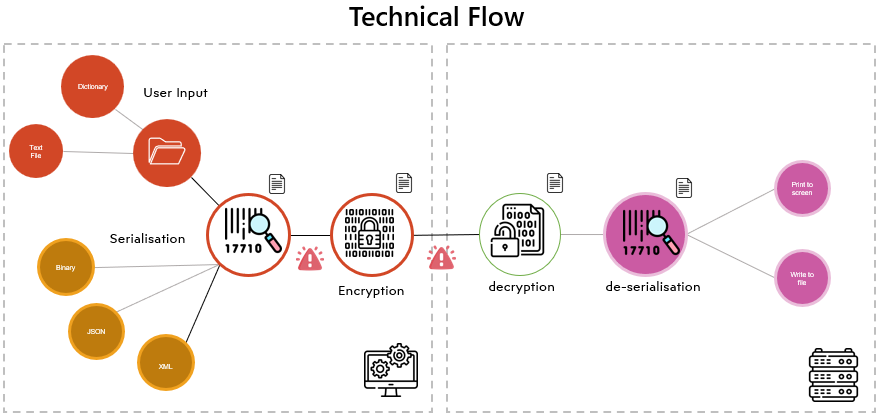
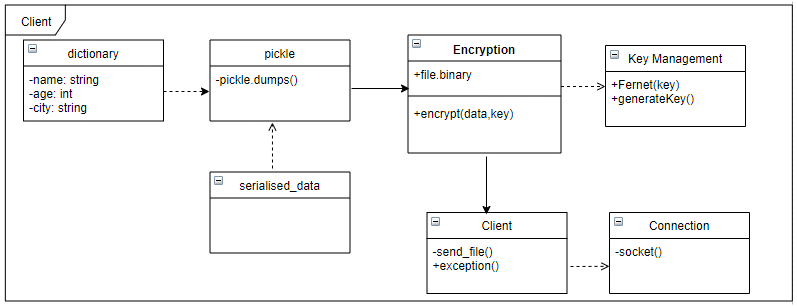


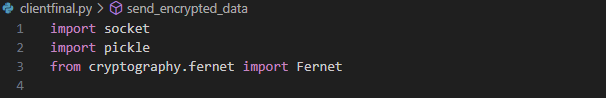
Figure 2. Technical Flow

# Solution Design

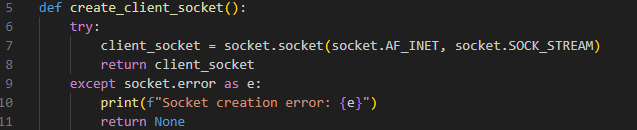
Solution Design (Client)



Python script clientfinal.py is written to serialise a dictionary, encrypt it and send it over to a server using socket programming. Fernet library is used to encrypt the data, pickle module for serialising and socket library to send and receive data over the network. These are imported as below –



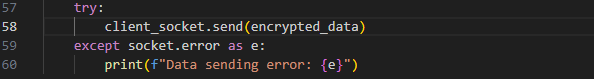
The script is organised into functions, making it readable, maintainable, available for reuse and isolated for effective unit testing.



The script includes comments and docstrings describing each function's purpose and what it does.



Error handling for every function is built into the code using the try-except block.



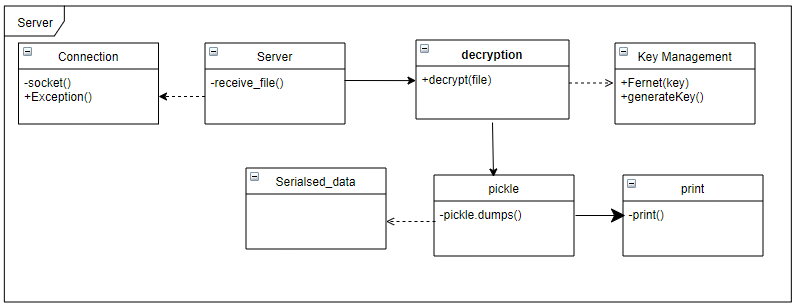
The script operates as follows:

1. A client socket is established by utilising the **create\_client\_socket()** function.
2. Connects to the server using the **connect\_to\_server()** function.
3. New encryption key is generated and written to a file using the **write\_key()** function.
4. Encryption key is loaded from the file using the **load\_key()** function.
5. The **pickle** module converts a dictionary into binary format for easy storage and transfer.
6. The data is serialised and then encrypted using the **encrypt()** function.
7. The data is encrypted and sent to the server using the **send\_encrypted\_data()** function.
8. The function **receive\_response()** is utilised to obtain a response from the server.
9. After receiving a response, the system decodes it and displays it on the console as a response from the server.
10. Before exiting, the script ensures that the client socket is closed.

Error handling within the script:

* If the script cannot create the client socket, it will display an error message and terminate.
* If the client socket fails to connect to the server, an error message is printed, and the socket is closed.
* If the script fails to generate or write the encryption key to the file, an error message is printed, and the script exits.
* If the encryption key cannot be loaded from the file, an error message is printed, and the script exits.
* If the serialised data cannot be encrypted, the script prints an error message and exits.
* The script prints an error message if the encrypted data cannot be sent to the server.
* If the server cannot receive a response, the script prints an error message.
* If any other error occurs, the script prints an error message and exits.

Solution Design (Server)

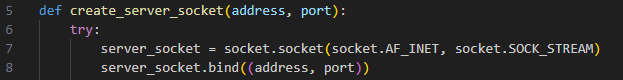


Python script serverfinal.py is written to receive encrypted serialised data from the client. It uses the socket module to create a server socket and accept incoming connections. It then uses the cryptography.fernet module to decrypt the data using a Fernet key.

Finally, it uses the pickle module to deserialise the data and save it to a file. The script includes comments and docstrings describing each function's purpose and what it does.

The script operates as follows:

* **create\_server\_socket()**creates a TCP server socket and binds it to the specified address and port.



* **accept\_connection()** function accepts an incoming connection on the server socket and returns the client socket.



* **load\_fernet\_key() t**his function loads the Fernet encryption key from the specified file.



* **receive\_and\_decrypt\_data() t**his function receives encrypted data from the client socket, decrypts it using the Fernet key, and returns the decrypted data.



* **deserialize\_data() t**his function deserialises the decrypted data using the pickle library.



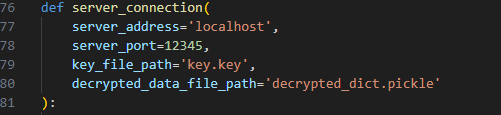
* **save\_decrypted\_data\_to\_file() t**his function saves the deserialised data to the specified file using the pickle library.



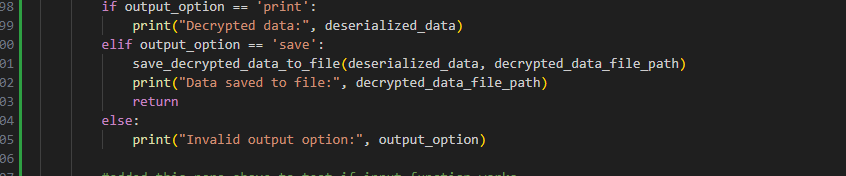
* **send\_response()**this function sends the specified response back to the client socket.



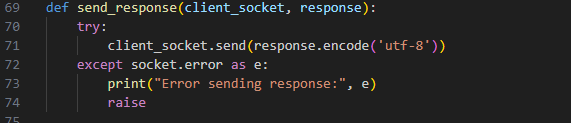
* **server\_connection()**thisfunction is the main entry point for the server. It creates a server socket, accepts a connection from a client, loads the Fernet key, receives and decrypts the data from the client, deserialises the data, saves it to a file, and then sends a response back to the client.



* Once a successful connection is established ‘try’ block will be executed which has the main logic of the server and will call functions to create server socket, load fernet key, de-serialise and decrypt data. Once complete it will as user or an output option of ‘print’ or ‘save’.



Error handling for every function is built into the code using the try-except block.



Error handling within the script:

* Socket errors will result in error messages and exceptions being raised.
* If the key file is not found, an error is printed, and an exception is raised.
* If the encrypted data cannot be decrypted, the code raises an exception and prints an error.
* If the de-serialised data cannot be saved to a file, the code raises an exception and prints an error message.
* If an error occurs while sending the response to the client, the code prints an error message and raises an exception.

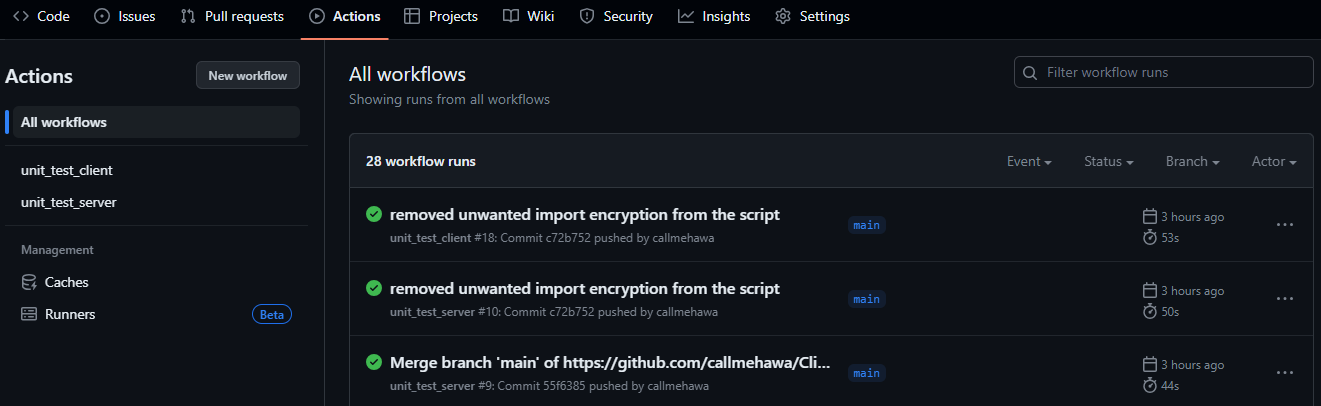
Design Decisions

**Encryption**: Fernet Cryptographic protocol will be used as our encryption tool of choice for the following reasons -

* Fernet uses symmetric key cryptography, which is simple and easy to use.
* Fernet is built on AES (Advance Encryption Standard) and ensures data encrypted is well-protected.

# Unit Testing

Unit testing is essential for maintaining code quality, reducing bugs, and supporting code changes. We have used GitHub Actions, a CI/CD platform, to run unit tests every time code is committed to a GitHub repository.



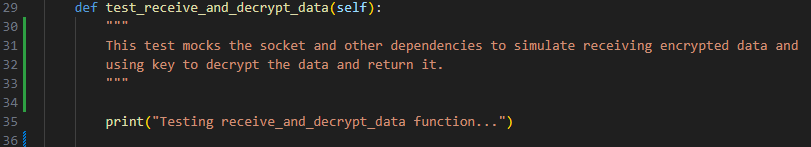
This is done by creating an action flow that defines the steps that should be taken when code is pushed to a branch. In this instance, the workflow will run either unit\_tests\_client.py or unit\_tests\_server.py every time code is pushed or pulled from any branch within the repository. Running unit tests every time code is committed using GitHub Actions, helps in early detection of bugs and improves quality.

Server Unit Testing

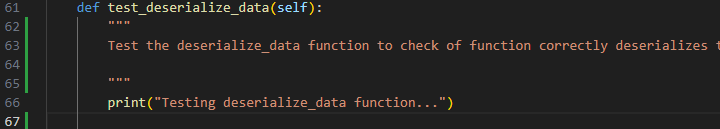
This unit tests are used to test the function within the server script called serverfinal.py. The tests status by creating a key file for testing and then mocks functions of serverfinal.py:

* receive\_and\_decrypt\_data(), which receives encrypted data from a socket and decrypts it using a key.
* deserialize\_data(), which deserializes a given data and returns a dictionary.

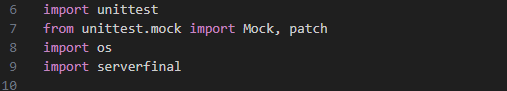
The receive\_and\_decrypt\_data() function receives encrypted data over a socket and decrypts it using a secret key.



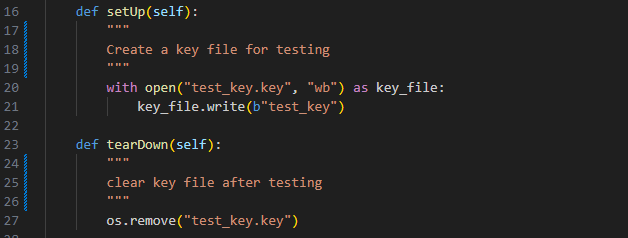
The deserialize\_data() function deserialises a given byte stream into a Python dictionary.



Unittest module provides the basic framework for writing unit tests and unittest.mock module provides methods for mocking objects. Serverfinal.py is imported to enable the script to call functions defined within the client script.



The unit tests use the unittest.TestCase class to create test cases. Each test case starts with a setUp() method, which generates a test\_key to use to decrypt the file and ends with a tearDown() method, which clears the key file after testing.



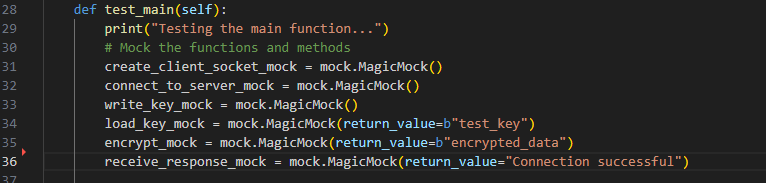
The test cases for the receive\_and\_decrypt\_data() function mock the socket to simulate receiving encrypted data and using the key to decrypt the data. The test case also runs function to return the decrypted data and returns as dictionary.

Client Unit Testing

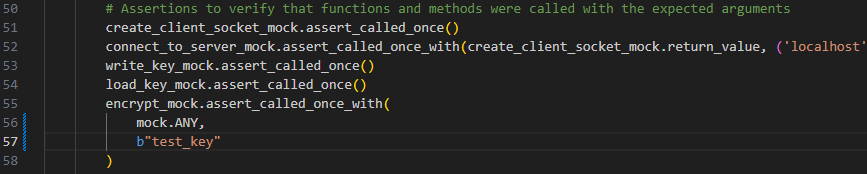
This unit tests are used to test the function within the server script called clientfinal.py. The test uses the unittest and mock modules to test the functionality of the client (clientfinal.py) module without connecting to a server and send and receive data.

The test case starts by creating a key file for testing. It then mocks the following functions and methods in the clientfinal module:

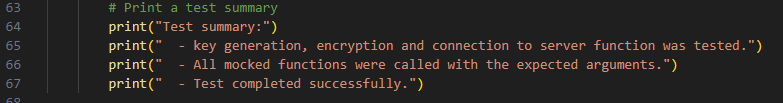
* create\_client\_socket
* connect\_to\_server
* write\_key
* load\_key
* encrypt
* receive\_response



After the main function has been called, assertions are used to verify that the functions and methods in the clientfinal module are called with the expected arguments. This validates that the code is running as intended.



The test case then prints a custom message of the outcome of the test.



# Conclusion

REFERENCES

APPENDICES

* GitHub Repository -
* Client Server Network Project report
* Requirement.txt
* Readme.md
* Screenshot of Github repo
* Screenshot of Logs