**END OF MODULE ASSIGNMENT**

**Building Client/Server Network**

By

**Bilir Sinan, Cheng Tsz Yeung, Lalchan Taras, Migliorini Ricardo**

Submitted to

The University of Liverpool

MASTER-OF-SCIENCE-DATA SCIENCE AND ARTIFICIAL INTELLIGENCE

*CSCK541 Moduel 2 Software Development in Practice January 2024*

Word Count: 1500

18/03/2024

TABLE OF CONTENTS

Table of Contents

[Introduction 3](#_Toc161607522)

[Client/Server Network 3](#_Toc161607523)

[CLIENT/SERVER NETWORK DESIGN 3](#_Toc161607524)

[Client/Server Work Flow 3](#_Toc161607525)

[Server Code 4](#_Toc161607526)

[Initialization 4](#_Toc161607527)

[Connection Handling & Data Processing 4](#_Toc161607528)

[Client-side 5](#_Toc161607529)

[User Data Collection 5](#_Toc161607530)

[Format Selection 5](#_Toc161607531)

[Serialization 5](#_Toc161607532)

[Data Transmission 6](#_Toc161607533)

[Continuation Handling 6](#_Toc161607534)

[Pickle 7](#_Toc161607535)

[Encryption 8](#_Toc161607536)

[Unit Test 9](#_Toc161607537)

[Server\_final\_unittest: 9](#_Toc161607538)

[client\_final\_unittest.py 10](#_Toc161607539)

[encryption\_final.py: 11](#_Toc161607540)

[References 13](#_Toc161607541)

# Introduction

## Client/Server Network

Establishing a client/server network is a fundamental aspect of computer networking and playing a crucial role in various applications such as data communications resource sharing, and distributed systems (Kirvan, P. 2023). This report aims to discuss the implementation of a client/server network using the Python programming language and evaluate its efficiency and effectiveness. We provide valuable insights into the principles of client/server architecture, the design and implementation of Python code, and the measurement of performance through experimentation and testing.

# CLIENT/SERVER NETWORK DESIGN

A client/server network is a type of computer network where multiple clients can access resources and services provided by a central server. The clients establish a network connection with the server, allowing them to submit requests for shared resources using dedicated client software or devices (Britvin et al 2022 and Umesh, 2012). The server, also referred to as the host, receives and processes these requests, and subsequently responds to the clients accordingly. (Britvin et al 2022 and Chakraborty, A. 2020)

## Client/Server Work Flow

The client and server applications initiate communication by generating a shared encryption key (Britvin et al 2022 and Rodríguez et al 2021). Both the client and server load the shared key for encryption and decryption purposes (Rodríguez et al 2021). To ensure secure communication, the client and server verify that their encryption keys match. Once the keys are verified, the client collects data from the user. This data is encrypted using the shared encryption key and transmitted to the server. The server decrypts the received data using the same shared key and stores it securely in a database or other storage mechanism.

To make the stored data usable, the server deserializes it into a format that can be processed. The server then processes the data as needed and may send a response back to the client. Finally, both the client and server applications terminate their operations.

A diagram of a server network

Description automatically generated

Figure 1. Client-Server Network flow diagram.

## Server Code

### Initialization

Our server code initializes a server socket and establishes a connection to the server. We configure various parameters for sending and receiving data, including the local machine name, the designated port, and the maximum byte limit for data transmission. (Jennings, N. n.d and Uchenna et al., 2021)

HOST and PORT specify the local machine name and port number (Sarker et. Al., 2015 and Uchenna et al., 2021). And we create the server socket using the IPv4 address family (AF\_INET) and the TCP protocol (SOCK\_STREAM) (Sarker et. Al., 2015 and Uchenna et al., 2021). It then binds the server socket to the address and setup the server to listen for incoming connections by server\_socket.listen().

A computer screen with text

Description automatically generated

Figure 2. Initialization

### Connection Handling & Data Processing

The code handling client connection and communication are inside the loop, the server receives encrypted data from the client, decrypts it, deserializes it into a Python object, and prints the deserialized data along with its format. If the format is “session\_end”, the loop is broken and the connection is closed.

A computer screen shot of a program code

Description automatically generated

Figure 3. Connection Handling

## Client-side

Our client code is responsible for collecting user data, serializing it into a chosen format (JSON, binary, or XML), encrypting it, where we will have a separate section for explaining the encryption feature, and sending it to the server. It also handles user input validation and allows the user to continue adding data or end the session.

### User Data Collection

We require our user to input their first and last names, standardizing them by removing leading and trailing spaces where Clean\_and\_format\_name() is used to remove leading and trailing spaces and capitalize the user's name. We also capitalize their names by using the function, collect\_user\_data() uses clean\_and\_format\_name() to collect the user's first and last names and return them in a dictionary.

A screen shot of a computer code

Description automatically generated

Figure 4. User Data Collection

### Format Selection

The user is prompted to choose the serialization format for the data (JSON, binary, or XML). The choose\_format() function let the user to choose the serialization format for the data and returns the chosen format.

A computer screen with text

Description automatically generated

Figure 5. Format Selection

### Serialization

The data is serialized into the chosen format using the appropriate library (json, pickle, or xml.etree.ElementTree) (Tangmunchittham and Piromsopa, 2022, and Matthes 2023). Serialize\_data() function takes the user data and the chosen format as input and serializes the data into the appropriate format. Pickle function will be covered later in our Pickle section.

A screen shot of a computer code

Description automatically generated

Figure 6. Serialization

### Data Transmission

The encrypted data is sent to the server using a socket connection (). Client\_socket.sendall() function is used to send the encrypted data to the server.

A screen shot of a computer code

Description automatically generated

Figure 7. Data Transmission

### Continuation Handling

The client prompts the user to indicate whether they want to continue adding data or end the session. If the user chooses to continue, the process repeats. If the user chooses to end the session, an end-of-session signal is sent to the server. Validate\_continuation\_response() prompts the user to indicate whether they want to continue adding data or end the session and returns the user's response.

A screen shot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

Figure 8. Continuation Handling

## Pickle

The pickle module implements binary protocols for serializing and deserializing a Python

object structure for cross-platform storage, data transfer, and minimization (M, R. 2024) while deserialization is the process of recreating the object from the stream of bytes.   
  
In the code, we use the Pickle.dumps() function to serialize the data into the binary format, and pickle.loads() to deserialize binary data received from the client. During deserialize a stream of bytes into a Python object. The encrypted\_data variable contains the encrypted data received from the client, and the cipher variable is used to decrypt the data.

A screen shot of a computer code

Description automatically generated

Figure 9. Pickle Serialize Code

A screen shot of a computer program

Description automatically generated

Figure 10. Pickle Deserialize Code

## Encryption

In the code we generate the encryption key by using the Fernet class from the cryptography module which is savedto a file (Rodríguez et al 2021 and Sredović 2023). The Fernet class provides a simple and secure way to encrypt and decrypt data using symmetric encryption, meaning that the same key is used for both encryption and decryption. (Fernet n.d and Sredović 2023)

The generate\_encryption\_key() function is used to generate a new encryption key while the save\_encryption\_key() function saves the generated encryption key to a file. The main() function calls the generate\_encryption\_key() and save\_encryption\_key() functions to generate and save an encryption key.

A screenshot of a computer program

Description automatically generated

Figure 11. Encryption Code

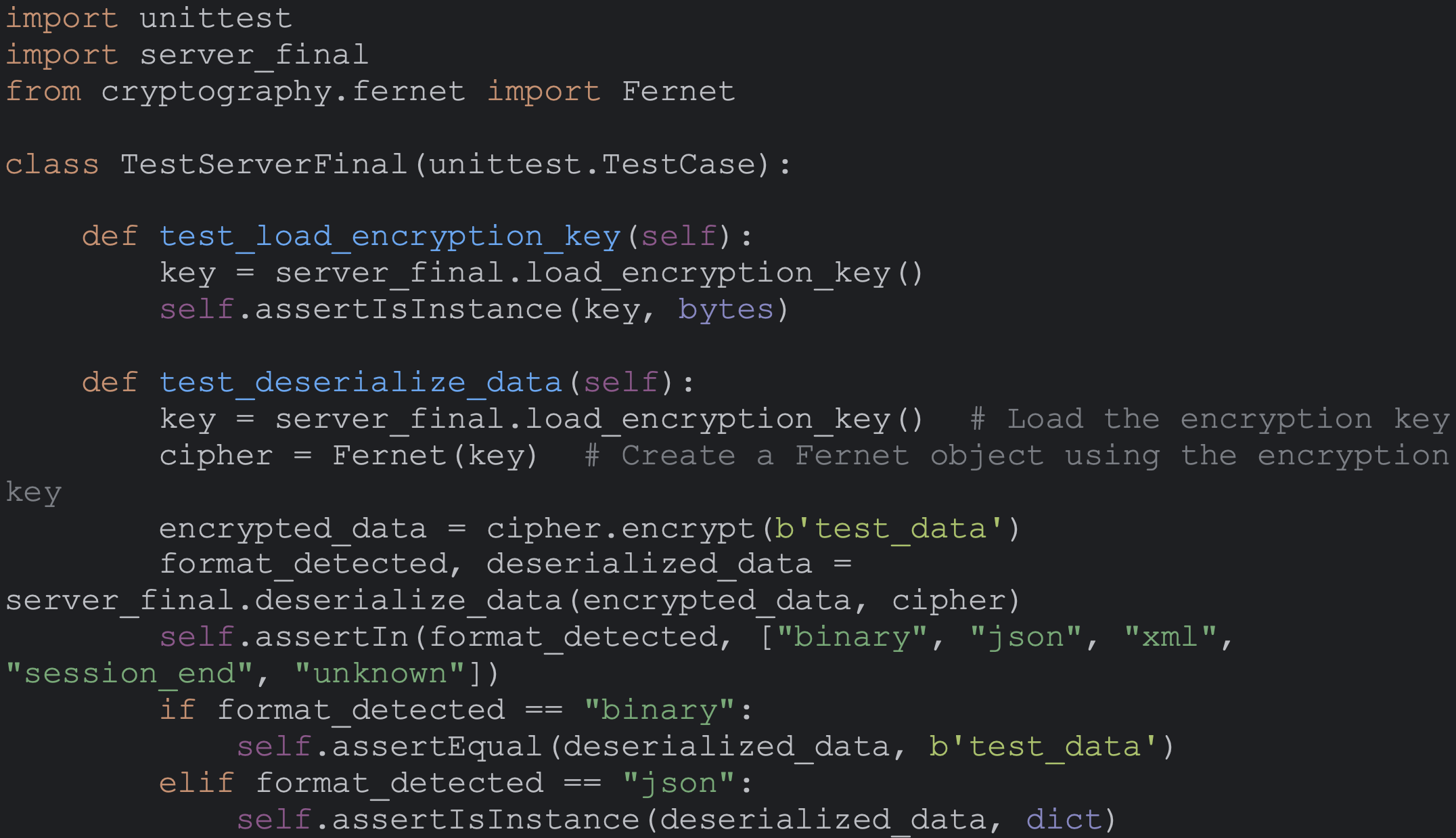
# Unit Test

## Server\_final\_unittest:

The provided code is a unit test script written using the **unittest** framework in Python. It tests two functions from the **server\_final** module: **load\_encryption\_key()** and **deserialize\_data()**.

Here's a breakdown of the code:

1. The script imports the **unittest** module, the **server\_final** module, and the **Fernet** class from the **cryptography.fernet** module.
2. Two test methods are defined within the **TestServerFinal** class:
   * **test\_load\_encryption\_key()**: This method tests the **load\_encryption\_key()** function from the **server\_final** module. It calls the function and checks whether the returned key is an instance of **bytes**.
   * **test\_deserialize\_data()**: This method tests the **deserialize\_data()** function from the **server\_final** module. It loads the encryption key, encrypts test data using Fernet, then deserializes the encrypted data using **deserialize\_data()**. It performs various checks based on the detected format to ensure proper deserialization.
3. The script checks if it is being run directly (**if \_\_name\_\_ == '\_\_main\_\_':**) and if so, it runs all the tests using **unittest.main()**.



A screen shot of a computer program

Description automatically generated

Figure 12. Unit test - Server Code

A screen shot of a computer

Description automatically generated

Figure 13. Expected Result - Server Code

Test result success.

## client\_final\_unittest.py

The provided code is a Python script that serves as a client application. It connects to a server over a TCP socket, collects user data, serializes it based on the chosen format (JSON, binary, or XML), encrypts the serialized data using Fernet encryption, and sends it to the server. The client can send multiple data sets until the user chooses to stop.

Here's a breakdown of the code:

1. It imports necessary modules such as **socket** for networking, **Fernet** from **cryptography.fernet** for encryption, **json** for JSON serialization, **pickle** for binary serialization, and **xml.etree.ElementTree** for XML serialization.
2. It defines several helper functions:
   * **load\_encryption\_key()**: Loads the encryption key from a file.
   * **clean\_and\_format\_name(name)**: Cleans and formats the user's name.
   * **collect\_user\_data()**: Collects user's first and last names with standardization.
   * **choose\_format()**: Allows the user to choose the data serialization format with input validation.
   * **validate\_continuation\_response()**: Asks the user whether to continue and validates the response.
   * **serialize\_data(data, format\_choice)**: Serializes data based on the chosen format.
3. The **main()** function:
   * Loads the encryption key.
   * Creates a Fernet cipher object using the encryption key.
   * Establishes a connection to the server.
   * Enters a loop to collect user data, serialize it, encrypt it, and send it to the server. The loop continues until the user chooses to stop.
   * Sends a session end signal to the server when the user chooses to stop.
4. The script checks if it is being run directly (**if \_\_name\_\_ == "\_\_main\_\_":**) and if so, it calls the **main()** function.

A screenshot of a computer program

Description automatically generated

Figure 14. Unit test - Client Code

A screen shot of a computer

Description automatically generated

Figure 15. Expected Result - client Code

Test result success.

## encryption\_final.py:

The provided Python script contains unit tests for the **generate\_encryption\_key()** and **save\_encryption\_key()** functions from the **encryption\_final** module. These tests are defined using the **unittest** framework.

Here's a breakdown of the code:

1. The script imports the **unittest** module and the **Fernet** class from the **cryptography.fernet** module.
2. It imports the **encryption\_final** module, presumably containing the functions to be tested.
3. Two test methods are defined within the **TestEncryptionFinal** class:
   * **test\_generate\_encryption\_key()**: This method tests the **generate\_encryption\_key()** function from the **encryption\_final** module. It calls the function and checks whether the returned key is an instance of **bytes**.
   * **test\_save\_encryption\_key()**: This method tests the **save\_encryption\_key()** function from the **encryption\_final** module. It generates a new encryption key using **Fernet.generate\_key()**, saves it using **encryption\_final.save\_encryption\_key()**, and should include assertions to check if the key is saved to the file correctly. Currently, this test does not have assertions added.
4. The script checks if it is being run directly (**if \_\_name\_\_ == '\_\_main\_\_':**) and if so, it runs all the tests using **unittest.main()**.

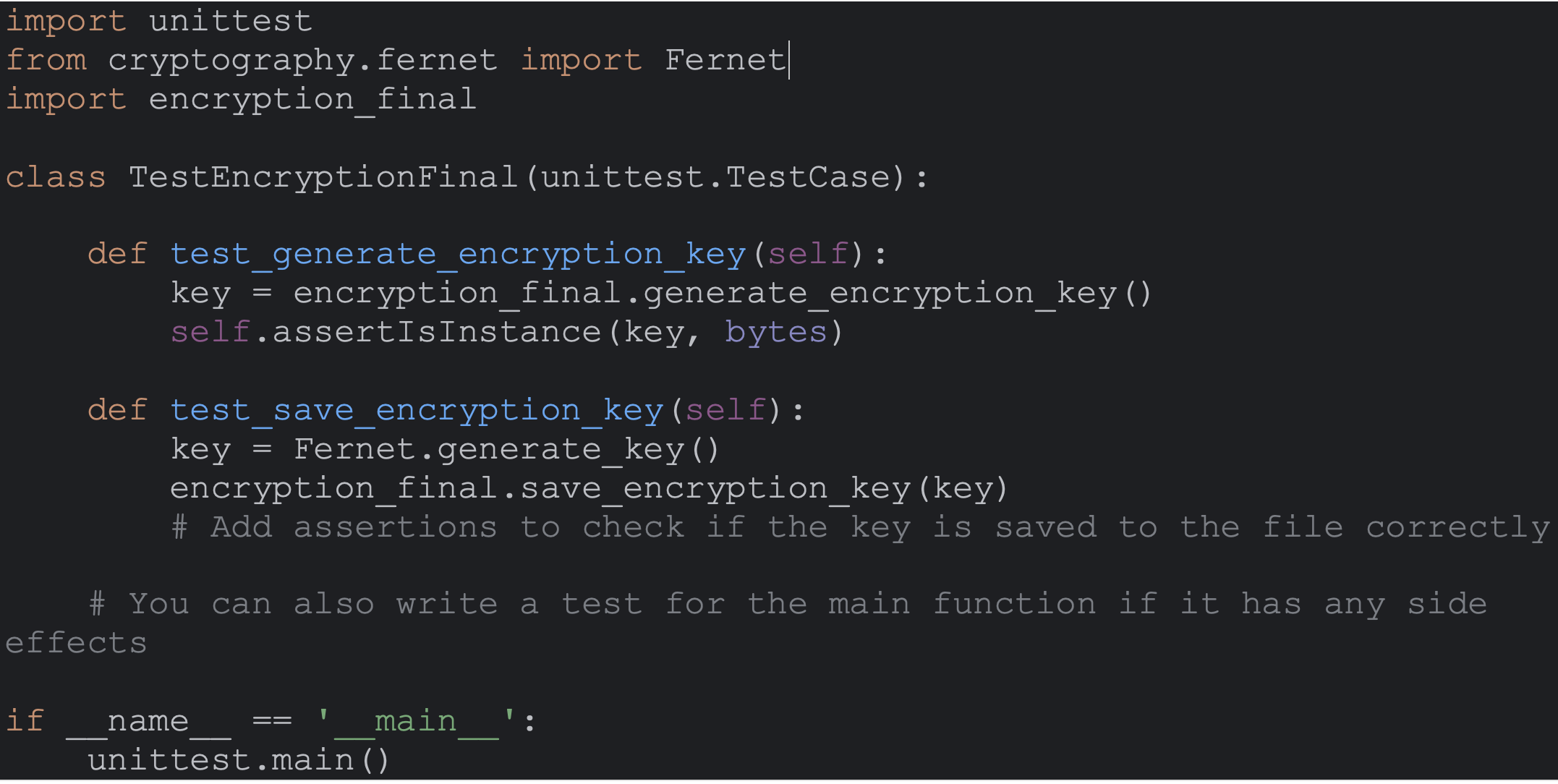


Figure 16. Unit test - Encryption Code

A screenshot of a computer

Description automatically generated

Figure 17. Expected Result - Encryption Code

Test result success.

# References and Bibliography

Britvin, A., Alrawashdeh, J.H. and Tkachuk, R. (2022). Client-server system for parsing data from web pages. *Advances in Cyber-Physical Systems*, *7*(1), pp.8-13.

Chakraborty, A. (2020) *System design basics: Client-server architecture*, *Medium*. Available at: <https://towardsdatascience.com/system-design-basics-getting-started-with-the-client-server-architecture-b02f9c9daae8> (Accessed: 17 March 2024).

docs.python.org. (n.d.). *unittest — Unit testing framework — Python 3.8.2 documentation*. [online] Available at: <https://docs.python.org/3/library/unittest.html>.

Fernet (n.d) *Fernet (symmetric encryption) - Cryptography 43.0.0.dev1 documentation*. Available at: [https://cryptography.io/en/latest/fernet/#](https://cryptography.io/en/latest/fernet/) (Accessed: 17 March 2024).

Hunt, J. (2019). Sockets in Python. *Advanced Guide to Python 3 Programming*, pp.457–470. doi:https://doi.org/10.1007/978-3-030-25943-3\_39.

Jennings, N. (n.d) *Socket programming in python (guide)*, *Real Python*. Available at: https://realpython.com/python-sockets/ (Accessed: 17 March 2024).

Kirvan, P. (2023) *What is a client-server network?: Definition from TechTarget*, *SearchWindowsServer*. Available at: <https://www.techtarget.com/searchwindowsserver/definition/client-server-network> (Accessed: 17 March 2024).

M, R. (2024) *Python pickle power: Mastering Python’s object serialization and Deserialization Technique*, *Medium*. Available at: <https://python.plainenglish.io/pickle-power-mastering-pythons-object-serialization-and-deserialization-technique-55f92cced894> (Accessed: 17 March 2024).

Matthes, E. (2023). *OOP in Python, part 13: Serialization and ABCs*. [online] Available at: <https://www.mostlypython.com/oop-in-python-part-13-serialization/> [Accessed 17 Mar. 2024].

‌

Python Software Foundation (2019). *pickle — Python object serialization — Python 3.7.3 documentation*. [online] Python.org. Available at: <https://docs.python.org/3/library/pickle.html>.

‌

Rodríguez R. G., M Gerardo Castang and Vanegas, C.A. (2021). Information Encryption and Decryption Analysis, Vulnerabilities and Reliability Implementing the RSA Algorithm in Python. *Springer eBooks*, pp.391–404. doi:https://doi.org/10.1007/978-3-030-89654-6\_28.

Sarker, M.F. and Washington, S., 2015. *Learning Python Network Programming*. Packt Publishing Ltd.

Sayfan, G. (2022). *Serialization and Deserialization of Python Objects: Part 1 | Envato Tuts+*. [online] Available at: <https://code.tutsplus.com/serialization-and-deserialization-of-python-objects-part-1--cms-26183t> [Accessed 17 Mar. 2024].

‌

Sredovic, U., 2023. Cryptography and cryptographic packages in Python. *Bánki Közlemények (Bánki Reports)*, *5*(1), pp.1-4.

Tangmunchittham, E. and Piromsopa, K. (2022). *An Analysis of Python Serialization towards Distributed Systems*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ECTI-CON54298.2022.9795388.

Umesh, P. (2012). Plotting with Python. *CSI Communications*, p.30.

Uchenna, U.I., Gregory, U.S., Virginus, U.N., Angela, O.A., Enyioma, C.K., Ezeora, N.J., Nneoma, O.C., Kenechukwu, A. and Michael, I.U. (2021). Exploring a Secured Socket Python Flask Framework in Real Time Communication System. *Asian Journal of Research in Computer Science*, pp.77–87. doi:https://doi.org/10.9734/ajrcos/2021/v8i130194.

‌