

# BASTION

## SECURE MAIL FORTRESS



By Ryan Forster

## Contents

1.0 Introduction .....	5
1.1 System Requirements.....	6
2.0 Design and Architecture .....	7
2.1 Bastion Overview.....	7
2.2 Core Features .....	7
2.3 Architecture Diagram.....	8
2.4 Security Layers .....	8
2.5 Simplicity by Design.....	9
3.0 Implementation .....	10
3.1 Development Environment .....	10
3.2 File Structure Overview .....	12
3.3 Coding Methodology, Implementation Process and Evolution .....	14
3.3.1 Development Approach .....	14
3.3.2 Programming Language and Libraries.....	15
3.3.3 Code Style and Documentation .....	15
3.3.4 Error handling .....	16
3.4 Security Challenges and Mitigation.....	16
3.4.1 Brute-Force and Credential Stuffing Attacks.....	16
3.4.2 Man-in-the-middle (MitM) and Eavesdropping .....	16
3.4.3 Message Tampering and Replay Attacks .....	17
3.4.4 Packet Sniffing.....	17
3.4.5 Local Time Manipulation.....	17
3.4.6 Lack of GUI and User error .....	18
3.5 Testing Environment and Procedure .....	18
3.5.1 Virtual Lab Setup .....	18
3.5.2 Tools Used.....	19
3.5.3 Testing Procedure.....	19
3.5.4 Summary of Test Outcomes.....	23
4.0 Evaluation .....	24
4.1 Protocol Strengths.....	24
4.1.1 Security by Design.....	24
4.1.2 End to End Encryption and Integrity .....	24

4.1.3 User Authentication and Secure Credentials.....	24
4.1.4 Lightweight and Modular Design .....	24
4.1.5 Compatibility with Virtualized Test Environments.....	24
4.1.6 <i>Resistance to common Cyber Attacks</i> .....	24
4.2 Limitations of Bastion.....	25
4.2.1 No Asynchronous Communication.....	25
4.2.2 Limited User Scalability and Management.....	25
4.2.3 No GUI or Web interface .....	25
4.2.4 Certificate Management and Trust Issues .....	25
4.2.5 Message Store Basic .....	25
4.2.6 No Protection Against Advanced Threats .....	25
4.2.7 No Multi-Factor Authentication (MFA).....	26
4.3 Comparing with Existing Solutions .....	26
4.3.1 Bastion Vs. Traditional Email (SMTP + TLS/PGP) .....	26
4.3.2 Bastion Vs. Secure Messaging Apps (Signal, ProtonMail) .....	27
4.4 Summary of Evaluation .....	27
5.0 Conclusion .....	29
6.0 Recommendations.....	30
6.1 Better Usability with GUI.....	30
6.2 Transition to a Database-Backed Backend .....	30
6.3 Advanced Brute Force Mitigation.....	30
6.4 Certificate Authority .....	30
6.5 More Encryption Options .....	30
6.6 Secure Message Deletion.....	31
6.7 Test on more platforms .....	31
7.0 References .....	32
8.0 Appendix.....	34
8.1 Bastion Server Code Iterations .....	34
8.2 Bastion Testing.....	60
8.3 Bastion – Secure Email Fortress .....	66

## Table of Figures

Figure 1: Architecture of Bastion Protocol .....	8
Figure 2: OpenSSL command.....	10
Figure 3: Visual Studio Code IDE environment .....	11
Figure 4: VMware Workstation Pro environment.....	12
Figure 5: File Structure .....	12
Figure 6: Basic Functionality.....	19
Figure 7: Hydra attack failed .....	20
Figure 8: Custom Script Brute Force Attack.....	21
Figure 9: Wireshark Sniffing Packet.....	22
Figure 10: Messa tampering HMAC.....	23

## List of Tables

Table 1: Core Features .....	7
Table 2: Tools and Environment .....	11
Table 3: Tools used .....	19
Table 4: Summary of Test Outcomes .....	23
Table 5: Bastion Vs. Traditional Email (SMTP + TLS/PGP) .....	26
Table 6: Bastion Vs. Secure Messaging Apps.....	27

# 1.0 Introduction

Now that digital communication affects all parts of our daily lives, how we protect our messages has become more crucial. Though email is one of the most popular methods to connect online, the system it relies on — SMTP — has security limitations. Throughout the years, additional steps like adding TLS and PGP were meant to boost security, but many times they are not required, difficult to configure or not used widely. In other words, these communications are likely to be intercepted, altered or imitated by hackers.

This project examines a more effective way. Rather than changing outdated protocols, it offers Bastion which is designed from scratch to be safe by design. Bastion is not built to replace email for all users around the world. In fact, it is a working demo that demonstrates how contemporary encryption and user identification systems help build a light, safe and secure protocol. The name refers to the fact that Bastion acts as a secure place for confidential communication that nobody can change.

Python is the programming language used to build Bastion and its security is accomplished through many layers. The connection is encrypted with SSL/TLS as the very start. Each Diffie-Hellman exchange guarantees a secret key that is only used in that session and Fernet is used to protect the messages by symmetric encryption. A HMAC code is put onto every message for authentication and passwords are securely lodged with bcrypt hashing. In addition, Bastion has light defence against brute-force attacks which keeps it secure from most common cybersecurity problems.

A major advantage of this project is that it is very straightforward. Bastion is completely independent and does not require additional services. It serves as an independent protocol for a client and server to talk, where all its features, including saving messages, user logins, encryption and session management, are built in. So, it's ideal to use for teaching, experimenting or as a standard to develop other secure messengers.

In order to validate the process, Bastion was examined using virtual machines in a controlled environment. The security system was tested against common challenges such as guessing passwords and stealing data and its encryption was confirmed by reviewing the content of captured traffic as messages continued to be hidden. The results showed that Bastion reliably supports the principles of confidentiality, integrity and authentication.

We are, in essence, focused on designing and testing a communication system that is safe from the outset. This subject brings together practical programming with cryptography, showing that you don't need complicated technology to achieve secure communication. The upcoming sections explain Bastion's design, implementation, testing and analysis, giving a picture of both the minor and major aspects involved in secure protocol creation.

## 1.1 System Requirements

The Bastion secure communication protocol was developed, deployed and tested by using the following system configurations and requirements.

### 1. Hardware:

- Processor: Dual-core 2.0 GHz or higher
- RAM: Minimum 2 GB (4 GB is recommended for VMs)
- Storage: 100 MB free for Bastion files and logs

### 2. Software Requirements:

- Operating System:
  - Server: Ubuntu 22.04 LTS
  - Client: Kali Linux 2023.1 (Or any Linux-based OS with Python support)
- Python version: Python 3.10 or above
- Python packages:
  - Cryptography
  - Bcrypt
  - Ssl (built-in with Python)
  - Socket (built-in with Python)
- Networking tools for testing (Kali Linux):
  - Hydra
  - Wireshark
  - Custom Python script
- Optional:
  - VMware or Virtual Box for isolated testing

By following these specifications, Bastion could be easily run and checked for accuracy in different situations such as secure client-server simulations, checking attack risks and monitoring the functioning of protocols.

All of the source code, documentation and necessary files for Bastion – Secure Mail Fortress can be found at the public source listed below.

<https://github.com/Codeon45/Bastion>

## 2.0 Design and Architecture

Bastion was designed with security and efficiency in mind; the two major goals were: security by design and simplicity in execution. While there are many secure messaging solutions, many of them rely on retrofitting encryption into systems that already exist like SMTP, which wasn't built with security in mind (*Mailspike Technologies, 2024*). This is where Bastion truly differs, it's built from the ground up with authentication, privacy and data integrity as its founding pillars.

### 2.1 Bastion Overview

Bastion offers simple and strong protection to make sure messages reach their intended recipients. Everything sent on this protocol is secured by TLS encryption from the time it leaves until it gets to its destination. In other words, protocol also involves:

- Users and servers check each other's identities first.
- Adding new users is simple because it can be done right through the server interface.
- On startup, the client asks for the server IP which helps make the program more flexible and easier to use.
- Session key exchange are done by using Diffie-Hellman (D-H)
- Fernet and HMAC to encrypt and verify the messages sent in the project.
- Bastion includes email simulations and safe storage for sent messages
- Data for all user credentials is stored as hashes by bcrypt.
- Protection for brute force attacks

Both usefulness and safety are maintained thanks to this design. This work is meant to demonstrate a better solution can be made and tested locally, not replace SMTP.

### 2.2 Core Features

Bastion accomplishes its main security aims by combining different known and working cryptography and network tools:

Table 1: Core Features

Component	Techonology used	Purpose
Transport Security	TLS via python module ssl	Encrypts client server communication
Session Encryption	Diffie-Hellman Key Exchange	Establishes a shared secret for each session
Message Security	Fernet + HMAC	Confidentiality and Integrity
User Auth	Bcrypt	Secure password hashing
Storage	JSON files	Simulates server-side mailboxes

The purpose of utilizing these libraries was to maintain tough security standards without creating extra obstacles for implementers (*Github.io, 2025*).

## 2.3 Architecture Diagram

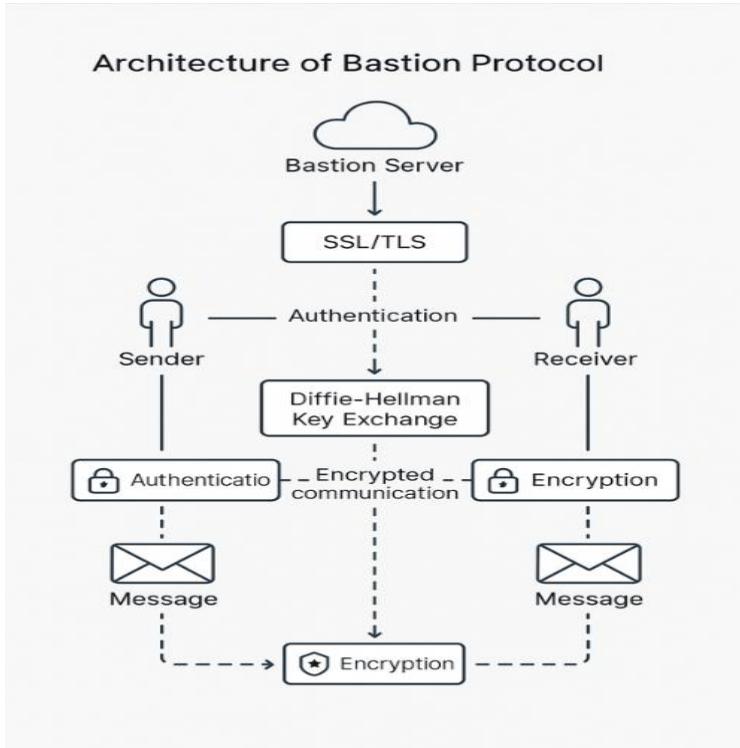


Figure 1: Architecture of Bastion Protocol

The diagram highlights how Bastion works by showing client and server interaction that occurs over a channel secured with TLS. It proves the authentication flow steps, a key exchange method and shuttling protected messages via Fernet and HMAC (*Mertens, 2025*). A secure socket is used for all communication, ensuring that data stays confidential and correct all the time.

## 2.4 Security Layers

Bastion is designed to defend data by using several layers of security at once. All traffic is securely encrypted using TLS 1.2 (*Banreet, 2024*). By default, the server uses a self-signed certificate which the client verifies until it's manually disabled during development. Using Diffie-Hellman Key Exchange, a new key is agreed upon for each session, so any messages sent before won't be compromised if a key is stolen (*Gillis, 2022*). The symmetric keys are used for encrypting messages, that come from the Diffie-Hellman secret and then hashed by HKDF (*Krawczyk and Pasi Eronen, 2025*). It makes certain that your conversation will be private. HMAC is added to each message to confirm it wasn't altered as it moved from one network to another (*GeeksforGeeks, 2020*). All user passwords are stored by using the bcrypt algorithm. Before storing, their passwords are both hashed and salted with bcrypt to avoid brute-force attacks (*Grigutyté, 2023*). If someone is blocked for too many unsuccessful login attempts, their account remains locked out for another 30 seconds. All these layers come together to shield data from eavesdropping, stopping people from making copies of data and help us avoid using other hacking methods.

## 2.5 Simplicity by Design

Bastion has excellent encryption, but its architecture is kept very simple by design. Because Bastion uses TCP, it eliminates the need for third-party server support and all of its information is saved in easy-to-use files that do not require databases. If you need email for a small group or a place where Postfix (*Postfix.org, 2025*) would be too much, Bastion makes a better choice. Because everything is transparent in the architecture, you can easily fix, test and analyse your system. It may not be ready for production, but it does have the function of a working, secure and testable mail alternative.

The decision was made not to include GUI because it would have added weight and complexity to what was designed to be a simple and clean command-line system.

## 3.0 Implementation

### 3.1 Development Environment

To make Bastion, a simple and practical toolset was built that allows for easy access, flexible testing and clear implementation. I built the main application on a **Windows PC** using **Visual Studio Code** because it is easy and has the built-in terminal feature that lets me iterate and debug as I create.

The core development of the Bastion Protocol used **Python 3.13**, mainly because it is easy to read and cryptographic libraries are widely available. Python's **ssl** module and the **cryptography** library are used by the protocol to encrypt sockets with **TLS** and to apply **Fernet** encryption, **HMACs** and **Diffie-Hellman** key exchanges. In addition, the **bcrypt** module was chosen for hashing passwords.

To simulate real world attacks and conditions, two virtual machines were set up:

- Ubuntu 22.04 LTS as the Server
- Kali Linux as the client machine and also attacker, specifically for launching brute force attacks and packet sniffing.

These virtual machines were run using VMware Workstation Pro. Both were configured to be in the same network, allowing them to communicate with each other (**Bastiaansen and Lanigan, 2019**).

Before starting development, cryptographic bcrypt packages were installed by using pip command, and additionally **OpenSSL** was used to generate a self-signed certificate and key for the server (**Openssl.org, 2025**). Command used:

```
C:\Users\ryan>openssl req -x509 -newkey rsa:4096 -keyout bastion_key.pem -out bastion_cert.pem -days 365 -nodes
```

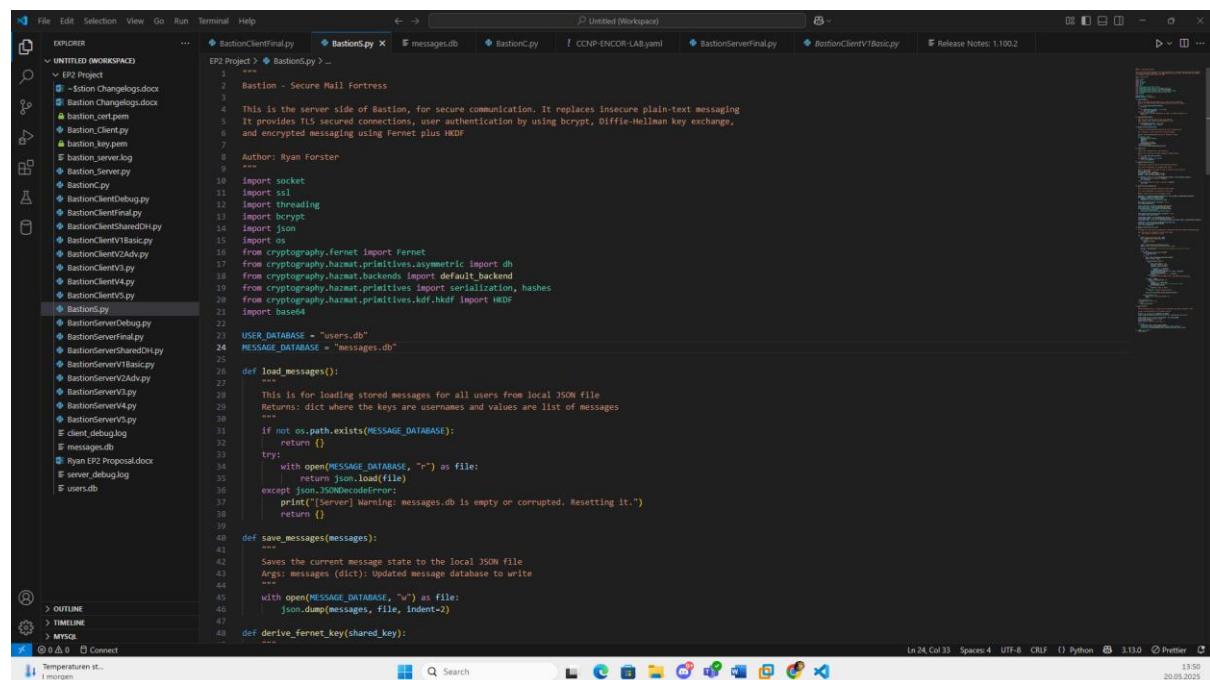
Figure 2: OpenSSL command

After the creation of the certificate, it's used for wrapping the server in TLS and allows the client to authenticate to the server during the handshake process. During the development, certificate verification was momentarily disabled, but for testing and deployment it was turned back on again to test full SSL validation.

Below is a table containing the main tools and environments:

*Table 2: Tools and Environment*

Tools/Component:	Description:
Python 3.13	Programming Language
Visual Studio Code	IDE
Ubuntu 22.04	Server for Bastion
Kali Linux	Client and Attacker
VMware Workstation Pro	Virtual Machine Management tool
Cryptography library	Encryption, HMAC and key exchange
Bcrypt module	Password Hashing
Ssl module	TLS socket wrapping and certificate handling
OpenSSL	Generat



*Figure 3: Visual Studio Code IDE environment*

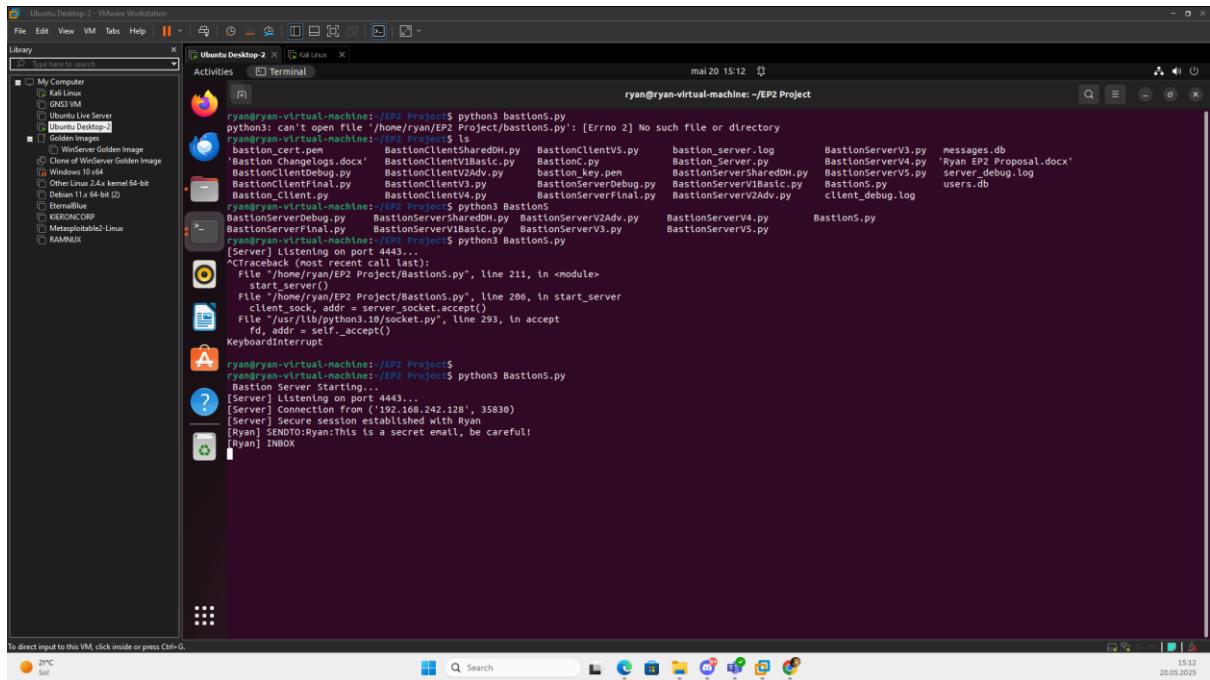


Figure 4: VMware Workstation Pro environment

### 3.2 File Structure Overview

Throughout the development of Bastion, a clean and minimal file structure was kept, in order to keep it simple, easy to test and clear separation of roles between client and server. Because Bastion is not built on top of an already existing mail infrastructure, there are no complex directory trees, and it doesn't rely on external databases. Everything is organized inside the project folder, like data, logic and security materials.

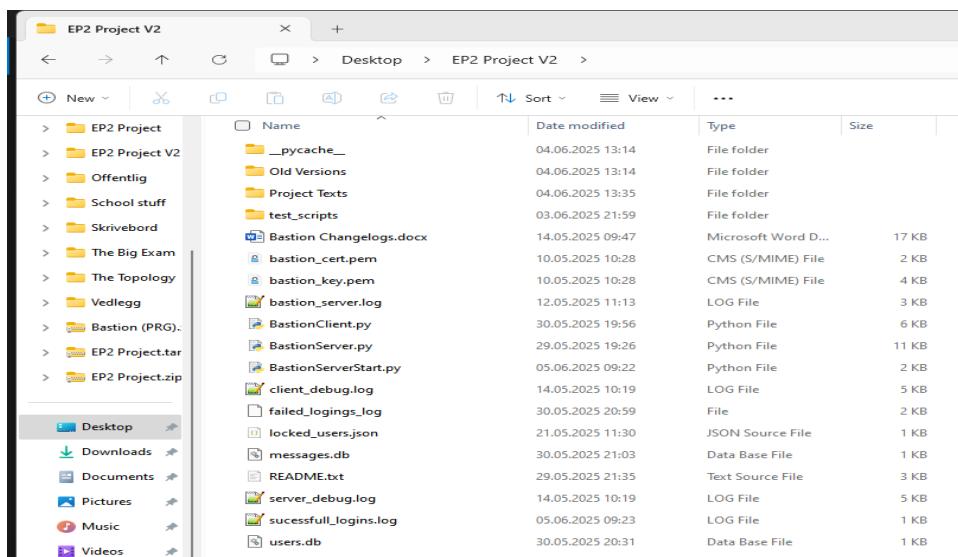


Figure 5: File Structure

Key Components:

- **BastionServer.py**
  - Here you can access all the server-side code from Bastion. The protocol includes authenticating users, setting up Transport Layer Security (TLS), changing keys with Diffie-Hellman, capturing messages and placing them into the user's inbox. It further logs and resists attempts by brute force attackers.
- **BastionServerStart.py**
  - This is a launcher to run the server or add users to the users.db file, making it easy to register new users (**BSS**)
- **BastionClient.py**
  - This is Bastion's code for the client side. When you log in, it sets up a connection, exchanges keys, displays a menu and lets you communicate through messages.
- **Bastion\_cert.pem / bastion\_key.pem**
  - These are the files generated with OpenSSL, they are the certificates needed for enabling TLS communication between server and client.
- **Users.db**
  - A Jason file used to simulate a database. Credentials are stored in it, passwords are hashed by bcrypt.
- **Messages.db**
  - Messages in this file are grouped with a username for each grouping. An entry is made up of a collection of received messages which provides a real-looking, message-organized inbox (**BCSM**).
- **Failed\_logins\_log**
  - Try to log in unsuccessfully and the file will note when and where you attempted to access, using time and IP information. With this, you can find brute force attacks and look into various security events.
- **Successful Logins Log**
  - This file keeps track of all successful logins, with the same functions as failed logins log, using time and IP information. Its good function to keep track of users.
- **Locked\_users.db**
  - Users stored in this file are blocked, and the file ensures they remain blocked even if server goes down, by always loading the file at start.

Other files are README, Changelogs, debugs and folder for scripts used for testing attacks against Bastion.

- With the README file, future users can find descriptions of how to set up the protocol.
- Changelogs is a list of changes that Bastion has gone through its development
- To fix bugs, debug was implemented when the Bastion was being programmed.
- Test Scripts contains all the scripts used during testing, mostly for attacks (BCPS, BSRA)

### 3.3 Coding Methodology, Implementation Process and Evolution

Bastion protocol was formed using a planned and continuous process, applying both secure programming concepts and hands-on experimentation. The goal was to develop a working model that clearly explains basic ideas in secure communication without the extra complexity of heavy or old frameworks. The work was broken into different phases, each built on the last and had continuous testing along the way.

#### 3.3.1 Development Approach

Bastion was created using a bottom-up coding method involving modules.

##### **Phase 1 – Secure connection**

TLS (SSL) was the first part to be implemented, with the goal in mind of encrypting the communication during transmission. In this process, a certificate was generated with OpenSSL and inserting the ssl module in the code (**BSV1, BCV1**)

##### **Phase 2 – User Authentication**

For the authentication part, python coding and bcrypt was used. Passwords are hashed and salted before storing them in the file users.db. During the log in phase, these credentials are checked against the hashes in that file. (**BSV5, BCV5**)

##### **Phase 3 – Secure Session Establishment**

Due to Diffie-Hellman Exchange, the server and client could negotiate a session key on the fly and use the keys for their communication. After that, HKDF uses the key to generate a secret which Fernet uses for symmetric encryption (**BSV6, BCV6**).

##### **Phase 4 – Message Handling**

After the channel was secured, messaging and receiving features were added. Messages sent over Fernet are encrypted and integrity comes from HMAC. Messages are saved to messages.db (BSM) by the server, so users can access them at any time (**BSV8, BSV9**)

## **Phase 5 – Security Hardening**

To harden the security of Bastion, brute force protection was added along with account lockout, and failed log in attempts. Each failed attempt is logged with a timestamp, IP address and temporary bans for too many failed attempts repeatedly (**BSF**).

## **Phase 6 – Testing and Attacking**

Bastion was put through brute force attacks, TLS sniffing and tampering tests. In this part of development, both the inbox cleansing and error handling options were improved (**BSBD**, **BSBD2**)

### **3.3.2 Programming Language and Libraries**

The project was implemented entirely with Python, due to its good readability and strong ecosystem for cryptography and networking. These were the libraries used:

Library	Purpose
Socket	Basic TCP communication
Ssl	TLS encryption
Cryptography.fernet	Symmetric Encryption (Fernet)
Cryptography.hazmat	Diffie-Hellman and HKDF key derivation
Bcrypt	Secure password hashing
Json	Storage for credentials and messages
Threading	Handling multiple clients simultaneously
Datetime/time	Account lockouts and timeout

Because of these modules, Bastion could use a modern and secure messaging system without needing tools such as databases or SMTP servers.

### **3.3.3 Code Style and Documentation**

Functions were structured so their names made sense, they had inline comments and were described by Python docstrings to ensure no confusion. To make code more modular, their goal was to make functions short and focused on what they needed to do. For example:

```
def derive_fernet_key(shared_key):
```

```
    """
```

Derives a Fernet-compatible key using HKDF from the DH shared secret.

```
    """
```

As a result, the code was simple and easy to work on. In the future, it can be used with different languages or frameworks.

### 3.3.4 Error handling

Functions were set up to deal with difficulties due to disconnection, decryption, unusual format errors, timeout cases and altering messages (using HMAC).

As a result, users continue to have stable experiences and malicious attempts do not cause the server to crash.

## 3.4 Security Challenges and Mitigation

When creating Bastion, main attention was given to blocking and anticipating usual cybersecurity threats. Every stage of setting up the system presented problems with keeping data safe at rest, while transferring and during the authentication process. The next sections describe the main risks included in the attack surface and how they were handled in the Bastion protocol.

### 3.4.1 Brute-Force and Credential Stuffing Attacks

#### **Challenge:**

This type of attack means the system tries all kinds of usernames and passwords in a fraction of a second. If usernames are something others can guess, it becomes even more dangerous. If no defence is in place, a would-be attacker can illegally access the system or overwhelm it by making too many login attempts (*Owasp.org, 2025*).

#### **Mitigation:**

After a person tries to log in unsuccessfully three times in a row, their account is temporarily put on hold for 30 seconds. Because the server adds timestamps, the timeout cannot be avoided by setting the computer clock manually.

All attempts when a login fails are saved with the client's IP and the time the attempt was made. There is now traceability which can be checked in detail during forensic evaluation.

After a login fails, a very short delay of about two seconds is added to make attacks by robots less possible. As the users.db data would contain only bcrypt hashes and some salt, there would be only a low chance of brute-force or table attacks succeeding on them.

### 3.4.2 Man-in-the-middle (MitM) and Eavesdropping

#### **Challenge:**

Should an intruder get their hands on unencrypted messages being sent along the network, they may be able to read or change private data. All client-server systems are vulnerable to this type of risk (*Cisco, 2025*).

#### **Mitigation:**

All data between the client and server is kept private using TLS 1.2. For the development phase, a self-signed certificate is applied which users must verify by themselves.

All encrypted messages come with an HMAC signature. Should a single byte change while transmitting the message, the server will deny the message and add a record of it in the log.

Using the Diffie-Hellman certificate, each client creates a different key for each communication. If session keys were to change, the new session would still be protected.

### 3.4.3 Message Tampering and Replay Attacks

#### **Challenge:**

Challenge: In systems using messages, attackers may try to break in by changing, doubling or adding messages to change the actions of the server or pretend to be users (*Stepan Ilyin, 2025*).

#### **Mitigation:**

With Fernet Encryption + HMAC, your messages are encrypted as well as signed. Prior to processing, timestamps and HMACs found in Fernet tokens are always checked.

If a token is picked up by intruders, the data within can't be read or used again since the session key is not known.

After you download a message from the database on the server, it is automatically cleared out of your inbox. It stops others from copying or altering what's in the message.

### 3.4.4 Packet Sniffing

#### **Challenge:**

Wireshark, as well as similar tools, allows you to capture info sent over an unencrypted network connection. It is a major concern for SMTP, for example, since messages travel without being encoded (*GeeksforGeeks, 2018*).

#### **Mitigation:**

With Bastion's TLS encryption, any traffic we transmit over its network looks like a secure encrypted messages when looked at using network analysis tools.

Since Bastion relies on a distinct protocol, it becomes much harder for automatic sniffer tools to make sense of transmitted data.

Demonstration: No human-readable material was present in the captured packets, confirming that Bastion blocks any data in plain form.

### 3.4.5 Local Time Manipulation

#### **Challenge:**

With the lockout depending on the client's clock, someone could easily evade it by setting their time back.

### **Mitigation:**

All the calculations for enforcing lockouts happen based on the server's time from `datetime.now()`. Because of this, you can't use a local clock change to unlock accounts or make a key valid again.

### **3.4.6 Lack of GUI and User error**

#### **Challenge:**

Although only a minor concern, lack of a graphical user interface could make users more likely to do the wrong thing. Whenever users submit the wrong data or validation is missed, errors may be found.

#### **Mitigation:**

The client uses a simple plan to show each step in the process to the user.

The system efficiently handles receiving empty inputs, malformed messages and problems with unexpected disconnects.

A system could be built using libraries such as **Tkinter** or **PyQt** to enable using a GUI, instead of regularly having to type commands and format input manually (*GeeksforGeeks, 2017*).

## **3.5 Testing Environment and Procedure**

Testing the reliability and security of Bastion was made possible by creating a test setting in virtualization. The test-run was meant to replicate actual communication situations and see if Bastion can handle well-known cyberattacks such as brute-force logins and packet checking. This section covers the organization of the lab, the equipment and tools applied and the ways the protocol is tested.

### **3.5.1 Virtual Lab Setup**

Testing was done with two virtual machines operating within a local host setup.

- In the Ubuntu 22.04 LTS (Server), Bastion was hosted and set up to use a self-signed TLS certificate on port 4443 (BCCF, BCCT). The server ensured login, handled giving out keys and managed saving the messages.
- I used Kali Linux as a Bastion client when I wanted to connect and as an attack platform for testing with tools like Hydra, Wireshark and my own brute-force script.

All the machines were connected to a sealed virtual network so they could talk to each other using IP addresses. Both systems could call out to each other and communicate securely via sockets over the given port.

### 3.5.2 Tools Used

Table 3: Tools used

Tool	Purpose
Python 3.13	Running Bastion Client and Server scripts
Hydra	Brute-force attack
Wireshark	Capturing and Analysing packets
Custom Python Script	Brute-force attack script
RockYou.txt	Worldlist for brute-force attack
OpenSSL	Generate TLS certificates

### 3.5.3 Testing Procedure

#### 1. Normal Use Case Scenario

##### Purpose:

Encourage secure login, establish a key exchange and make sure messages are sent. (BSBF, BCBF1)

##### Steps:

- TLS is used for connecting clients to the server.
- A valid username and password are required to use the username (BCWU).
- Diffie-Hellman is used to share the session key.
- A Fernet encryption algorithm and HMAC technique are used for the message.

##### Outcome:

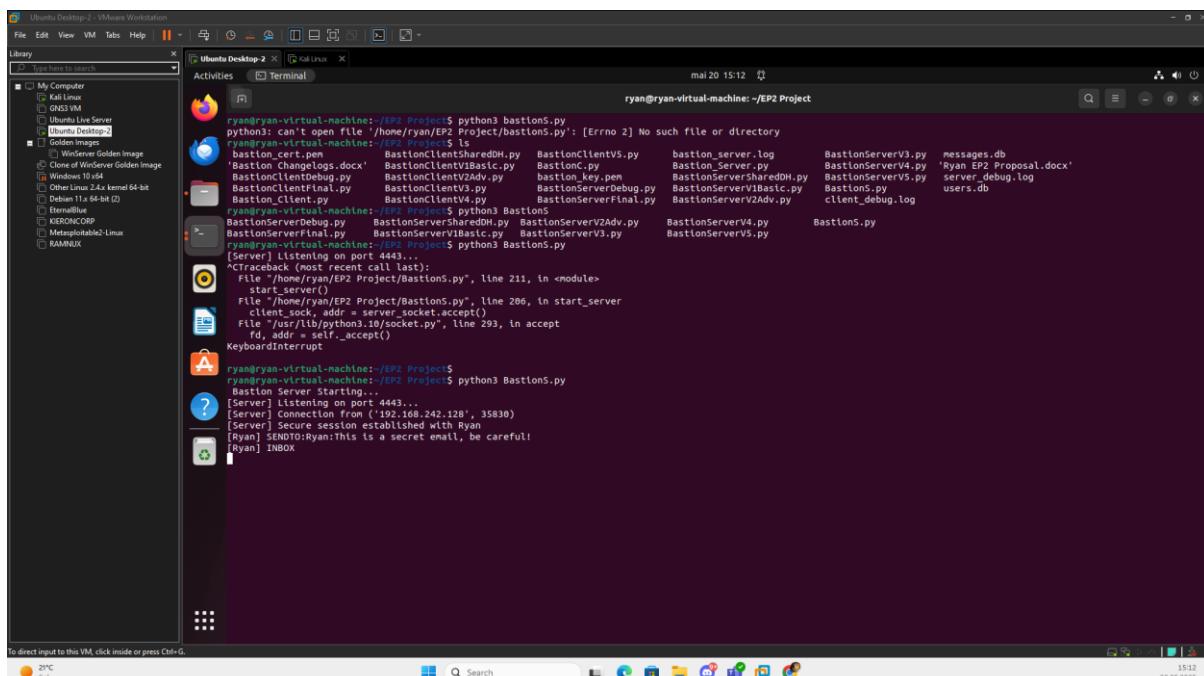


Figure 6: Basic Functionality

Secure message is saved to your account and is now accessible in your inbox, after which the message is removed once you have read it.

**It passed the test** — both the features and security succeeded.

## 2. Hydra Brute Force Attack

## Purpose:

Investigate if automated tools such as Hydra are able to force a Bastion login without a password.

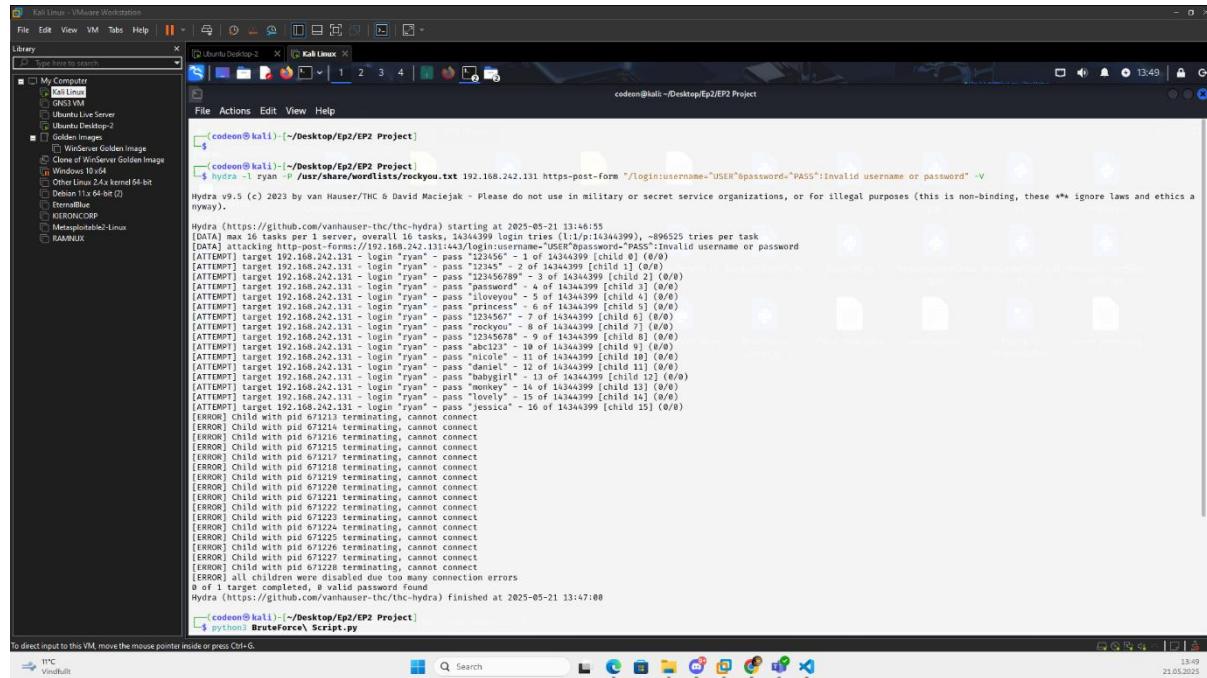
### **Attempted Method:**

Hydra was utilized for attempting logins against Bastion, using RockYou.txt wordlist ([Lurey, 2023](#)).

```
hydra -l ryan -P /usr/share/wordlists/rockyou.txt 192.168.242.131 -s 4443 <protocol>
```

But instead of **HTTP POST**, **SSH**, **FTP** or any other protocol families commonly used like **HTTPS**, Hydra cannot read the protocol Bastion uses.

## Outcome:



*Figure 7: Hydra attack failed*

All of the requests to Hydra produced errors and the connection was lost.

There were no attempts to connect with the Bastion service at any point.

### **Conclusion:**

It is impossible to attack Bastion using Hydra because Bastion communicates with a unique and encrypted protocol. As a consequence, Hydra cannot fully make sense of how the login

system works. All brute-force efforts should be performed by making a custom Python script that reproduces the protocol. Thanks to the way protocol was designed and TLS encryption, Bastion did not fall to Hydra.

### 3. Custom Script Brute Force Attack

#### Purpose:

Since Hydra had no chance, now Bastion brute force defence will be tested with a custom python script (BCBF, BCBFA).

#### Steps:

- Python script allows you to repeat guessing passwords.
- Each time a login didn't succeed, the counter went up.

#### Outcome:

```
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certif
└──(codeon㉿kali)-[~/Desktop/Ep2/EP2 Project]
$ python3 BruteForce\ Script.py
[*] Trying password: 1234
[Server] Invalid username or password.
[*] Trying password: password
[Server] Invalid username or password.
[*] Trying password: admin
[Server] Too many failed attempts. User locked for 30 seconds.
[*] Trying password: noroff
[Server] Too many failed attempts. Try again later.
[*] Trying password: letmein
[Server] Too many failed attempts. Try again later.
[*] Trying password: Noroff
[Server] Too many failed attempts. Try again later.

└──(codeon㉿kali)-[~/Desktop/Ep2/EP2 Project]
$ █
```

Figure 8: Custom Script Brute Force Attack

- Locking the account was initiated at the third consecutive failure.
- Your login request was not successful due to timing out.
- Having to delay and track on the server side slowed down the attempts to retry requests.

The outcome was Pass, thanks to server validation.

### 4. Wireshark Packet Sniffing

#### Purpose:

Check when sensitive data (such as login and messages) could be snatched during the attack.

## Steps:

- Wireshark was used on the Kali machine at the same time the test emails were sent.
- The traffic was studied using the TLS filter.

## Outcome:

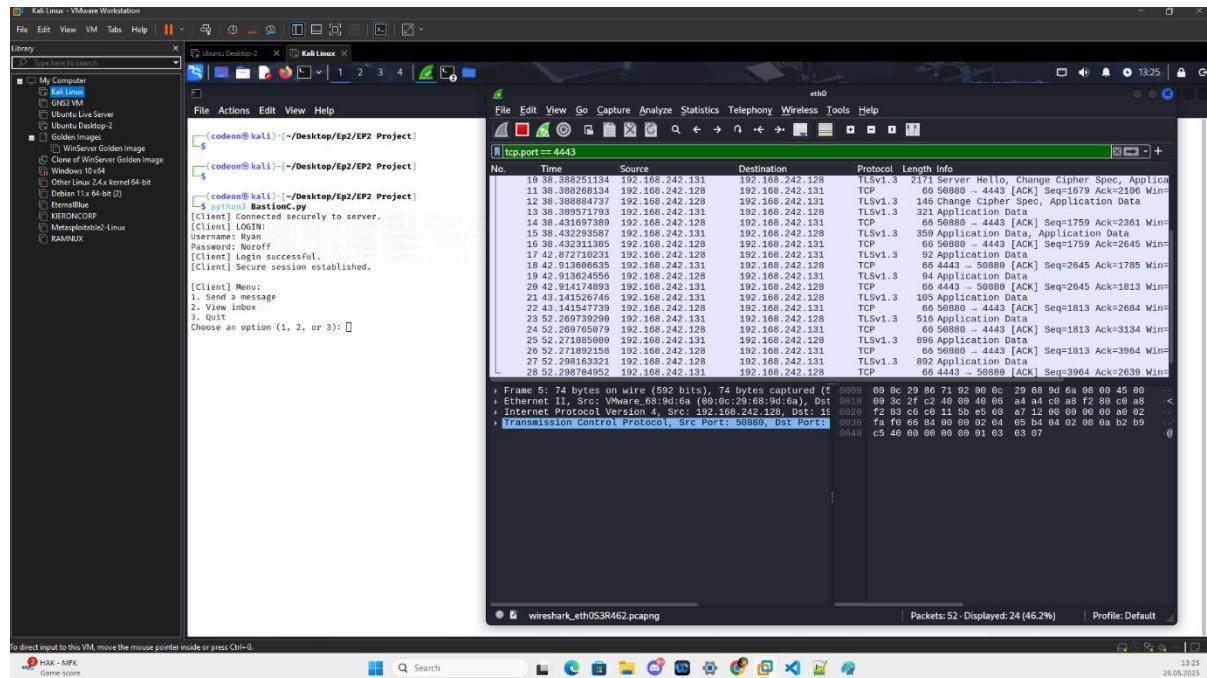


Figure 9: Wireshark Sniffing Packet

- No readable data on the packets could be seen because all traffic was encrypted. (BCWS)
- TLS authentication was performed, the data was encrypted and the certificate was transferred in an encrypted way.

The examination concluded with TLS + Fernet fully securing the traffic.

## 5. Tampering and Replay attacks

### Purpose:

To find out if captured or modified packets could be utilized again.

### Steps:

- Encrypted messages had to be manually recopied and sent out a second time.
- Messages that were altered were discovered to have an invalid HMAC.

## Outcome:

The screenshot shows a terminal window titled "ryan@ryan-virtual-machine: ~/EP2 Project". The user has run several Python scripts to handle file transfers and messaging. A red box highlights a warning message from the server indicating that an HMAC check failed due to possible tampering.

```

ryan@ryan-virtual-machine:~/EP2 Project$ python3 Bastions.py
[Server] Listening on port 4443...
[Server] Located at 192.168.242.128:4443
[Server] Connection from ('192.168.242.128', 32786)
[Server] Secure session established with Ryan
[Warning] HMAC Check failed - possibly tampering going on.
[Server] Connection from ('192.168.242.128', 42674)
[Server] Secure session established with Ryan
[Warning] HMAC Check failed - possibly tampering going on.
^C[Traceback (most recent call last):
  File "/home/ryan/EP2 Project/Bastions.py", line 298, in <module>
    start_server()
  File "/home/ryan/EP2 Project/Bastions.py", line 289, in start_server
    client_sock, addr = server_socket.accept()
  File "/usr/lib/python3.10/socket.py", line 293, in accept
    fd, addr = self._accept()
KeyboardInterrupt

ryan@ryan-virtual-machine:~/EP2 Project$ python3 Bastions.py
[Server] Listening on port 4443...
[Server] Connection from ('192.168.242.128', 47352)
[Server] Secure session established with Ryan
[Ryan] SENDTO:ElenaHey
[Server] Connection from ('192.168.242.128', 59692)
[Server] Secure session established with Elena
[Elena] INBOX
[Server] Connection from ('192.168.242.128', 50880)
[Server] Secure session established with Ryan

```

Figure 10: Message tampering HMAC

- The server rejected any packets it said had failed in its HMAC checks.
- Trying to read the log again was not possible because of unique keys per session.

The result was clear — the integrity of the message was maintained.

### 3.5.4 Summary of Test Outcomes

Table 4: Summary of Test Outcomes

Test Type	Outcome	Notes
Secure Messaging	Passed	Encryption with TLS, DH, Fernet, and HMAC worked according to plan.
Brute-Force Protection	Passed	Lockouts out after 3 failed attempts.
Packet Sniffing	Passed	Nothing is readable because of encryption
Message Integrity	Passed	HMAC protected integrity
Replay Attempts	Passed	HMAC and session keys completely stopped it.

## 4.0 Evaluation

### 4.1 Protocol Strengths

Bastion shows several important features that prove it is effective for secure communication. It is designed to be very secure from the start and skips the problems found in older systems such as SMTP that do not adequately consider encryption or integrity.

#### 4.1.1 Security by Design

While other security systems add an encryption layer onto existing protocols, Bastion puts in place secure protocols. Through TLS, communication is secure from the start and using Diffie-Hellman gives each conversation its own secret key. Knowing that one session key can be stolen, Bastion assures that previous and future sessions remain unaffected.

#### 4.1.2 End to End Encryption and Integrity

Fernet automatically uses a symmetric key to secure and time-sensitive each message. It is also improved by HMAC verification, checking if the message has escaped tampering while sent across the network.

#### 4.1.3 User Authentication and Secure Credentials

All user credentials are always coded or put into some form other than plain text. Therefore, passwords are not stored directly but are hashed with bcrypt, a hashing method especially built to stop brute-force attempts. By using the login process, security is protected and when users fail to log in many times, accounts are automatically locked, stopping diction and brute-force attacks.

#### 4.1.4 Lightweight and Modular Design

Bastion is designed so that it does not complicate things. Python and common libraries such as ssl, bcrypt and cryptography are used for all features, making everything easy to check and keep updated. JSON files save messages in the project, standing in for inboxes and avoiding the need for a full database for use in a prototype or academic demo.

#### 4.1.5 Compatibility with Virtualized Test Environments

Bastion was successfully installed and tested in virtual machines that ran Ubuntu (server) and Kali Linux (client). It proves that Bastion can easily be used on various systems and supports testing security systems using real world hacking tools.

#### 4.1.6 Resistance to common Cyber Attacks

Attempts to guess or crash passwords with brute force were put to the test using Bastion as well as using Hydra and custom scripts. Because of rate limiting, hashing user credentials and leaving accounts locked after a timeout, all the attacks were either stopped or rendered unsuccessful — this demonstrated that there was a strong defence against most standard authentication attacks.

## 4.2 Limitations of Bastion

Although Bastion is a lightweight and safe method for communication, its limits should be recognized when planning to use it or expand it outside of academic research.

### 4.2.1 No Asynchronous Communication

Currently, Bastion functions using a synchronized client-server approach. You need to be connected actively to be able to send and receive messages. Since Bastion expects online, end-to-end sessions, the system is not practical for many common email tasks.

### 4.2.2 Limited User Scalability and Management

Users are authenticated using the flat JSON file named users.db. However, it isn't possible to use this process in larger settings. Users cannot sign up or be assigned roles and the system does not work with major authentication systems (like LDAP and OAuth). For enterprise-scale systems, Bastion has to be changed in a major way before it can be implemented.

### 4.2.3 No GUI or Web interface

Bastion is entirely controlled using command lines. Because of this, the protocol is easy to develop and test, but it may be tough for users without technical knowledge. If the interface was improved with graphs or a web version, it would be much easier for people who aren't familiar with terminal tools to use it.

While a GUI based on Tkinter was tried, it was put aside when it showed issues and the goal was to maintain the main program's basic and focused CLI nature

### 4.2.4 Certificate Management and Trust Issues

While being developed, Bastion relies on TLS certificates that clients consider unknown and thus not automatically trusted. While testing works best with these certificates, using them in the real world allows for security vulnerabilities if users do not question their authenticity. For production, you should only use a certificate authority (CA) that you trust.

### 4.2.5 Message Store Basic

New inboxes are created with the help of local JSON files. As a result, it does not include proper access control, redundancy or persistence features like today's message queues or mail servers do. There isn't a way to search, sort or manage messages other than pulling them up and deleting them.

### 4.2.6 No Protection Against Advanced Threats

The security system in Bastion can withstand brute force and simple replay attacks, but it does not have defence for more sophisticated threats.

- Timing attacks
- Traffic Analysis
- Denial of Service (DOS)

- Advanced MitM strategies, like certificate spoofing

#### 4.2.7 No Multi-Factor Authentication (MFA)

At this moment, Bastion depends solely on username-password login. Standard practice in today's security systems is to require multi-factor authentication. Using token-based MFA or confirming with a mobile app would make access and security much stronger than before.

### 4.3 Comparing with Existing Solutions

Assessing Bastion's practical value requires a comparison with both old and current methods of secure communication systems. For better understanding, comparing these categories requires using either traditional email methods (SMTP with STARTTLS or PGP) or popular secure messaging apps like Signal or ProtonMail.

#### 4.3.1 Bastion Vs. Traditional Email (SMTP + TLS/PGP)

*Table 5: Bastion Vs. Traditional Email (SMTP + TLS/PGP)*

Criteria	Bastion	SMTP + TLS/PGP
Encryption	Mandatory TLS + Fernet	Optional TLS, PGP is user-dependent
Authentication	Built-in with bcrypt hashing	External, often basic
Integrity	HMAC Verification	PGP Signature (if configured)
Easy of Setup	Easy Python-based setup	Needs mail server, DNS, PGP handling
Message Delivery	Simulated Inbox (JSON)	Asynchronous delivery via server relay
Replay Protection	Session bound keys, HMAC	Nothing, unless configured
Brute Force Defence	Account lockout, logging	Rarely implemented natively
Usability	CLI only	Widely used, GUI clients available

#### Interpretation:

SMTP-based email can be safely used by adding Pretty Good Privacy (PGP) or TLS, but it is the user who must set things up correctly (*Buckbee, 2020*). The design of Bastion makes security automatic, so it cannot be used insecurely by accident. Bastion does not offer the flexible systems of features found in regular email platforms.

### 4.3.2 Bastion Vs. Secure Messaging Apps (Signal, ProtonMail)

Table 6: Bastion Vs. Secure Messaging Apps

Criteria	Bastion	Signal/ProtonMail
Encryption	DH, Fernet, HMAC	Modern ciphers
Infrastructure	Self hosted JSON based system	Full cloud-based backend
Authentication	Username, password	MFA, biometric, device paring
Interface	CLI only	Polished GUI/mobile apps
Scalability	Limited to local setups	Designed for millions of users
Open Source	Yes	Yes (Signal), Partially (ProtonMail)
Traffic Analysis	No	Yes

#### Interpretation:

Current messaging apps like Signal or ProtonMail offer more advanced features and a better design; however, their private, closed setup worries some users about privacy and transparency (*Signal Messenger, 2025*) (*Proton, 2023*). The best use for Bastion is in education or with team communications, since it is more focused on control and exposure than on growing or ease of use.

#### Conclusion:

Bastion isn't intended to compete with large-scale business tools or simple consumer apps. It is designed to show that by using essential security principles — TLS, DH key exchange, symmetric encryption, HMAC and authentication — one can develop a new secure communication method. It has strong security features and administration, but its lack of scalability, user interface and useful options keep it behind more grown-up systems.

## 4.4 Summary of Evaluation

Bastion was designed as a secure communication example addressing the common issues found in email and long-established protocols. On the basis of testing and analysis, these conclusions were made:

#### Strengths

- With Bastion, every message you send or receive is always encrypted and validated. Unlike retrofitted approaches, security is built in and essential from the beginning.
- It was made more convenient for users to type in the server IP while connected and have accounts generated by the server.
- Use of TLS, Fernet and HMACs in message checks guarantees that stolen or tampered messages are noticed by the system.

- All passwords are protected by hashing them with bcrypt and by locking out accounts when brute-force attacks take place.
- The system is both simple to use and easy to check. By choosing simple storage methods like JSON, it is able to safeguard transactions with complex cryptographic algorithms.
- Tests show that Bastion successfully resists brute force, replay attacks and packet sniffing.

## **Limitations**

- Since you need to type commands in a dedicated window, Bastion might not be the easiest platform for people without IT knowledge. A GUI was created briefly, but it was decided to remove it for the sake of improved testing and simpler deployment.
- Bastion is developed to fit small environments such as secure lab networks or small teams. It has not been created to work with huge volumes of traffic.
- Certificates must be set and security environments paired manually which takes some knowledge of the process.
- There is No Queue or Delivery Delay: Unlike regular postal systems, Bastion doesn't store or send messages through multiple servers. Messages are saved directly on your device.

## **Verdict**

Bastion meets its aim by providing a secure email-like communication method prototype. It gives a clear understanding of how several cryptographic tools can be worked into a secure system using only their internal features. Bastion works efficiently and usefully in teaching, research and controlled scenarios.

## 5.0 Conclusion

Bastion, a secure messaging protocol, was developed in this project to fix basic security vulnerabilities found in standard email platforms. Unlike SMTP which is not as secure since it was not developed with these main principles from the start, Bastion was built with confidentiality, integrity and authentication at the center.

Bastion relies on modern techniques such as TLS for safe data delivery, Diffie-Hellman for selecting session keys, Fernet for locked data, HMAC to verify messages and bcrypt to protect user passwords. They were chosen for being secure as well as for matching suitably with streamlined, easy to maintain code structures. In addition, functions were added to keep track of login failures and turn off accounts temporarily, simulating reliable behavior for users.

Testing and implementing Bastion was done by Python in an Ubuntu server and a Kali Linux client under controlled conditions. Testing revealed that the protocol can tolerate brute-force threats, that messages cannot be read by packet sniffers and that key agreement and encryption functions are flawless.

Bastion wasn't created to replace modern email platforms, but it shows that secure protocols can be designed without needing additional security features afterward. The project could still be developed by adding options for asynchronous delivery, building a graphical interface and ensuring the servers will support the project's growth.

All things considered; Bastion succeeded in creating a safe prototype for communication. It gives us practical insight into cryptography and serves as a learning guide for people studying secure systems. Performing and testing actions helped show how protocols are built, systems are secured and challenges in dealing with real cyber threats occur.

## 6.0 Recommendations

Bastion – Secure Mail Fortress has proven to be a strong and secure messaging system, but there are still several areas where making improvements can improve its efficiency, how easy it is to use and how quickly it can grow.

### 6.1 Better Usability with GUI

Work on adding a graphical user interface (GUI) was attempted, but it turned out to be too difficult, so a command-line interface was kept for stability and ease of use. To be used more widely, especially by people who do not code, providing a solid GUI with frameworks such as Tkinter, PyQt or Electron (via Python bridge) would enhance user comfort and make things simpler.

### 6.2 Transition to a Database-Backed Backend

At this moment, JSON files are used by Bastion to save the details of users and their exchanged messages. It is appropriate for creating a prototype, but with a growing number of users, it might not work as well. You can achieve higher performance, reliable indexing and stronger data security by moving to SQLite when testing locally and using PostgreSQL when hosting remotely (**Abiodun Easuola, 2024**).

### 6.3 Advanced Brute Force Mitigation

While Bastion uses strong security by limiting failed logins and locking accounts for a time, the security can still be improved with the following add-ons:

- IP Based rate limiting
- CAPTCHA Support (if GUI included)
- Permanent lock outs if failed attempts keep happening

### 6.4 Certificate Authority

Now, Bastion relies on self-signed TLS certificates, so it works for testing but not for larger, real deployments. Using a recognized Certificate Authority such as Let's Encrypt would create more trust in the system and eliminate TLS validation issues (**Letsencrypt.org, 2024**)

### 6.5 More Encryption Options

The current use of encryption in Bastion is Fernet (AES-CBC behind the scenes). In future versions, it would be useful to add some of the following:

- Using ChaCha20 allows for speeding up encryption on phone devices (**Crawford, 2023**).
- Regularly changing the keys for forward secrecy

## **6.6 Secure Message Deletion**

Messages get deleted after opening them in the inbox, even so, it may still not be fully erased securely. In addition, secure methods or temporary saving of messages which allow users to control, would improve privacy.

## **6.7 Test on more platforms**

Bastion was so far run within a virtual machine (Ubuntu Server and Kali Linux). Having more tests on Windows, macOS and various network configurations would help Bastion be available and safe for more people.

## 7.0 References

- Abiodun Eesuola (2024). *SQLite vs PostgreSQL: A Detailed Comparison*. [online] Datacamp.com. Available at: <https://www.datacamp.com/blog/sqlite-vs-postgresql-detailed-comparison> [Accessed 5 Jun. 2025].
- Banreet (2024). *Enable Transport Layer Security (TLS) 1.2 overview - Configuration Manager*. [online] Microsoft.com. Available at: <https://learn.microsoft.com/en-us/intune/configmgr/core/plan-design/security/enable-tls-1-2> [Accessed 27 May 2025].
- Bastiaansen, R. and Lanigan, R. (2019). *VMware*. [online] Search VMware. Available at: <https://www.techtarget.com/searchvmware/definition/VMware> [Accessed 29 May 2025].
- Buckbee, M. (2020). *What is PGP Encryption and How Does It Work?* [online] Varonis.com. Available at: <https://www.varonis.com/blog/pgp-encryption> [Accessed 29 May 2025].
- CISA (2025). *Secure by Design | CISA*. [online] Cybersecurity and Infrastructure Security Agency CISA. Available at: <https://www.cisa.gov/securebydesign> [Accessed 27 May 2025].
- Cisco. (2025). *All Roads Lead to Ransomware: Inside Cisco Talos Threat Hunters*. [online] Available at: <https://www.cisco.com/site/us/en/learn/topics/security/what-is-a-cyberattack.html#~types-of-attacks> [Accessed 27 May 2025].
- Crawford, D. (2023). *What is ChaCha20? | Proton VPN*. [online] Proton VPN. Available at: <https://protonvpn.com/blog/chacha20?srsltid=AfmBOoolCZPeGRFEWXdgDEQNHUh8ZflKzJwgRBrsC3S6pniDbwX8f3S> [Accessed 5 Jun. 2025].
- Cryptography.io. (2025). *Welcome to pyca/cryptography — Cryptography 46.0.0.dev1 documentation*. [online] Available at: <https://cryptography.io/en/latest/> [Accessed 27 May 2025].
- GeeksforGeeks (2017). *Python Tkinter*. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/python-gui-tkinter/> [Accessed 27 May 2025].
- GeeksforGeeks (2018). *What is Packet Sniffing?* [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/what-is-packet-sniffing/> [Accessed 27 May 2025].
- GeeksforGeeks (2020). *What is HMAC (Hash based Message Authentication Code)?* [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/what-is-hmacha-shash-based-message-authentication-code/> [Accessed 27 May 2025].
- Github.io. (2025). *Libraries & Pandas*. [online] Available at: <https://broadinstitute.github.io/2024-09-27-python-intro-lesson/libraries.html> [Accessed 29 May 2025].
- Gillis, A.S. (2022). *Diffie-Hellman key exchange (exponential key exchange)*. [online] Search Security. Available at: <https://www.techtarget.com/searchsecurity/definition/Diffie-Hellman-key-exchange> [Accessed 27 May 2025].

- Grigutyté, M. (2023). *What is Bcrypt and how it works?* [online] NordVPN. Available at: <https://nordvpn.com/blog/what-is-bcrypt/> [Accessed 27 May 2025].
- Letsencrypt.org. (2024). *Let's Encrypt.* [online] Available at: <https://letsencrypt.org/> [Accessed 5 Jun. 2025].
- Lurey, C. (2023). *Understanding RockYou.txt: A Tool for Security and a Weapon for Hackers.* [online] Keeper Security Blog. Available at: <https://www.keepersecurity.com/blog/2023/08/04/understanding-rockyou-txt-a-tool-for-security-and-a-weapon-for-hackers/> [Accessed 29 May 2025].
- Mailspike Technologies (2024). *Understanding the Issues with SMTP Protocol.* [online] Anubisnetworks.com. Available at: <https://www.anubisnetworks.com/blog/understanding-the-issues-with-smtp-protocol> [Accessed 27 May 2025].
- Mertens, X. (2025). *Have You Ever Heard of the Fernet Encryption Algorithm?* - SANS Internet Storm Center. [online] Available at: <https://isc.sans.edu/diary/30146> [Accessed 29 May 2025].
- Openssl.org. (2025). *openssl-req - OpenSSL Documentation.* [online] Available at: <https://docs.openssl.org/master/man1/openssl-req/> [Accessed 27 May 2025].
- Owasp.org. (2025). *Brute Force Attack | OWASP Foundation.* [online] Available at: [https://owasp.org/www-community/attacks/Brute\\_force\\_attack](https://owasp.org/www-community/attacks/Brute_force_attack) [Accessed 27 May 2025].
- Postfix.org. (2025). *The Postfix Home Page.* [online] Available at: <https://www.postfix.org/> [Accessed 27 May 2025].
- Proton. (2023). *Proton: Privacy by default.* [online] Available at: <https://proton.me/> [Accessed 29 May 2025].
- Python documentation. (2018). *ssl — TLS/SSL wrapper for socket objects.* [online] Available at: <https://docs.python.org/3/library/ssl.html> [Accessed 27 May 2025].
- Signal Messenger. (2025). *Signal Messenger: Speak Freely.* [online] Available at: <https://signal.org/> [Accessed 29 May 2025].
- Stepan Ilyin (2025). *Replay Attacks.* [online] Wallarm.com. Available at: <https://www.wallarm.com/what/replay-attacks> [Accessed 27 May 2025].
- W3schools.com. (2025). *W3Schools.com.* [online] Available at: <https://www.w3schools.com/python/> [Accessed 29 May 2025].

# 8.0 Appendix

## 8.1 Bastion Server Code Iterations

### Bastion Server V1 (BSV1)

```
File Edit Selection View Go Run Terminal Help
UNTITLED WORKSPACE
EP2 Project
- $tion Changelog.docx
- Bastion Changelog.docx
- bastion.cert.pem
- BastionClient1.py
- bastion.key.pem
- bastion_server.log
Bastion Server.py
BastionClientDebug.py
BastionClientFinal.py
BastionClientSharedH.py
BastionClientV1Basic.py
BastionClientV2Adv.py
BastionClientV3.py
BastionClientV4.py
BastionClientV5.py
BastionServerDebug.py
BastionServerFinal.py
BastionServerSharedH.py
BastionServerV1Basic.py
BastionServerV2Adv.py
BastionServerV3.py
BastionServerV4.py
BastionServerV5.py
client_debug.log
Ryan EP2 Proposal.docx
server_debug.log
users.db

Bastion Server.py
1 import socket
2 import ssl
3 import threading
4 import logging
5 from cryptography.fernet import Fernet
6
7 # Pre-shared symmetric key (must be the same on client and server)
8 PRE_SHARED_KEY = b'2zEOpskXWGD08tCiz_0mlwIG-Qht2qSISYja3XXB8='
9
10 def start_bastion_server():
11     """
12         Starts the Bastion Server using SSL/TLS encryption.
13         listens for client connections, receives an encrypted message, and decrypts it.
14     """
15     cipher = Fernet(PRE_SHARED_KEY)
16
17     # Setting up a secure server socket using SSL/TLS (using the new certificate)
18     context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
19     context.load_cert_chain(certfile="bastion.cert.pem", keyfile="bastion.key.pem")
20     context.minimum_version = ssl.TLSVersion.TLSv1_2
21     context.maximum_version = ssl.TLSVersion.TLSv1_3
22
23     # Creating and starting the server
24     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
25     server_socket.bind(("0.0.0.0", 4443))
26     server_socket.listen()
27
28     print("[Bastion] Secure Server is Running... (Listening on port 4443.)")
29
30     while True:
31         client_socket, address = server_socket.accept()
32         print("[Bastion] Connection established with [address]")
33
34         secure_socket = context.wrap_socket(client_socket, server_side=True)
35         print("[Bastion] Secure connection established using SSL/TLS.")
36
37         encrypted_message = secure_socket.recv(4096)
38         print("[Bastion] Received Encrypted Message:", encrypted_message)
39
40         decrypted_message = cipher.decrypt(encrypted_message)
41         print("[Bastion] Decrypted Message:", decrypted_message.decode())
42
43         secure_socket.close()
44         print("[Bastion] Connection closed.")
45
46     # Run the Server
47     if __name__ == '__main__':
48         start_bastion_server()

In 45, Col 27  Spaces 4  UTF-8  CRLF  Python  3.11.0  PyCharm 12:08 14.05.2025
```

### Bastion Server V2 (BSV2)

```
File Edit Selection View Go Run Terminal Help
UNTITLED WORKSPACE
EP2 Project
- $tion Changelog.docx
- Bastion Changelog.docx
- bastion.cert.pem
- BastionClient1.py
- bastion.key.pem
- bastion_server.log
Bastion Server V2Adv.py
BastionClient1.py
BastionClientV3.py
BastionServerV1Basic.py
BastionServerV2Adv.py
BastionServerV3.py
BastionServerV4.py
BastionServerV5.py
client_debug.log
Ryan EP2 Proposal.docx
server_debug.log
users.db

Bastion Server V2Adv.py
1 import socket
2 import ssl
3 import threading
4 import logging
5 from cryptography.fernet import Fernet
6
7 # Configuration
8 PRE_SHARED_KEY = b'2zEOpskXWGD08tCiz_0mlwIG-Qht2qSISYja3XXB8='
9 LOGFILE = "bastion_server.log"
10
11 # Setup Logging
12 logging.basicConfig(filename=LOGFILE, level=logging.INFO, format='%(asctime)s - %(message)s')
13
14 # Function to Handle Each Client
15 def handle_client(secur_sock, address):
16     try:
17         logging.info("Connection established with [address]")
18         print("[Bastion] Connection established with [address]")
19
20         # Using the pre-shared symmetric key
21         cipher = Fernet(PRE_SHARED_KEY)
22
23         # Receiving and decrypting the encrypted message
24         encrypted_message = secur_sock.recv(4096)
25         logging.info("Encrypted Message from [address]: [encrypted message]")
26         print("[Bastion] Received Encrypted Message:", encrypted_message)
27
28         decrypted_message = cipher.decrypt(encrypted_message)
29         logging.info("Decrypted Message from [address]: [decrypted message.decode()]")
30         print("[Bastion] Decrypted Message:", decrypted_message.decode())
31
32         # Sending a response to the client (acknowledgment)
33         response = ("Message received: [decrypted_message.decode()]")
34         secur_sock.sendall(response.encode())
35
36     except Exception as e:
37         logging.error("Error with [address]: [str(e)]")
38         print("[Bastion] Error with [address]: [str(e)]")
39
40     finally:
41         secur_sock.close()
42         logging.info("Connection closed with [address]")
43         print("[Bastion] Connection closed with [address]")
44
45 # Function to Start the Server
46 def start_bastion_server():
47     """
48         Starts the Bastion Server using SSL/TLS encryption.
49         listens for multiple client connections, receives and logs encrypted messages, and responds.
50     """

In 81, Col 3  Spaces 4  UTF-8  CRLF  Python  3.11.0  PyCharm 12:10 14.05.2025
```

## Bastion Server V3 (BSV3)

```

File Edit Selection View Go Run Terminal Help < - > Untitled (Workspace) EP2 Project > BastionServerV3.py ...
47     while True:
48         # Receiving and decrypting the encrypted message
49         encrypted_message = secure_socket.recv(4096)
50         if not encrypted_message:
51             break
52
53         print("[Bastion] Received Encrypted Message:", encrypted_message)
54         decrypted_message = cipher.decrypt(encrypted_message).decode()
55         print("[Bastion] Decrypted Message from [username]:", decrypted_message)
56         logging.info("Message from [username]: [decrypted_message]")
57
58         # Storing the message in the user's inbox
59         inboxes[username].append(decrypted_message)
60         save_inboxes(inboxes)
61
62         # Sending a confirmation response to the client
63         response = f"Message received and stored in your inbox."
64         secure_socket.sendall(response.encode())
65
66     except Exception as e:
67         logging.error(f"Error with [address]: {str(e)}")
68         print(f"[Bastion] Error with [address]: {str(e)}")
69
70     finally:
71         secure_socket.close()
72         logging.info("Connection closed with [address]")
73         print("[Bastion] Connection closed with [address]")
74
75     * Function to Start the Server
76     def start_bastion_server():
77         """
78             Starts the Bastion Server using SSL/TLS encryption.
79             Listens for multiple client connections, receives and logs encrypted messages, and stores them in user-specific inboxes.
80         """
81         print("[Bastion] Attempting to start the secure server...")
82
83     try:
84         # Setting up a secure server socket using SSL/TLS (using the new certificate)
85         context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
86         context.load_cert_chain(certfile="bastion_cert.pem", keyfile="bastion_key.pem")
87         context.minimum_version = ssl.TLSVersion.TLSv1_2
88         context.maximum_version = ssl.TLSVersion.TLSv1_2
89
90         # Creating and starting the server
91         server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
92         server_socket.bind(("0.0.0.0", 4443))
93         server_socket.listen(5)

```

In 24, Col 27. Spaces:4 - UTF-8 - CRLF. ( Python 3.13.0. 12:12 14.05.2025

## Bastion Server V4 (BSV4)

```

File Edit Selection View Go Run Terminal Help < - > EP2 Project > BastionServerV4.py ...
1 import socket
2 import ssl
3 import threading
4 import logging
5
6 logging.basicConfig(filename="server_debug.log", level=logging.DEBUG, format='%(asctime)s - %(message)s')
7
8 def handle_client(secure_socket, address):
9     logging.info("Secure connection established with {address}")
10    print("[Server] Secure connection with {address}")
11
12    try:
13        while True:
14            message = secure_socket.recv(1024).decode()
15            if not message:
16                break
17            print("[Server] Received: {message}")
18            logging.info("Received from client: {message}")
19
20            # Echoing the message back
21            response = f"Server received (SSL): {message}"
22            secure_socket.send(response.encode())
23            logging.info("Sent to client: {response}")
24
25        except Exception as e:
26            print("[Server] Error: {str(e)}")
27            logging.error(f"Error: {str(e)}")
28
29        secure_socket.close()
30        logging.info("Secure connection closed with {address}")
31        print("[Server] Secure connection closed with {address}")
32
33    def start_server():
34        print("[Server] Starting Secure SSL Server...")
35        logging.info("SSL server started.")
36
37        context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)

```

In 1, Col 1. Spaces:4 - UTF-8 - CRLF. ( Python 3.13.0. 14:25 ENG NO 29/05/2025

```

File Edit Selection View Go Run Terminal Help < - > EP2 Project > BastionServerV4.py ...
33 def start_server():
34
35     context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
36     context.load_cert_chain(certfile="bastion_cert.pem", keyfile="bastion_key.pem")
37
38     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
39     server_socket.bind(("localhost", 4443))
40     server_socket.listen(5)
41
42     print("[Server] listening on port 4443 (SSL)...") 
43
44     while True:
45         client_socket, address = server_socket.accept()
46         secure_socket = context.wrap_socket(client_socket, server_side=True)
47         thread = threading.Thread(target=handle_client, args=(secure_socket, address))
48         thread.start()
49
50
51     if __name__ == "__main__":
52         start_server()

```

## Bastion Server V5 (BSV5)

```
import socket
import ssl
import threading
import bcrypt
import json
import os
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives.asymmetric import dh
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization

USER_DATABASE = "users.db"

def load_users():
    if not os.path.exists(USER_DATABASE):
        return []
    with open(USER_DATABASE, "r") as file:
        return json.load(file)

def authenticate(client_sock):
    users = load_users()
    sock.sendall("LOGIN".encode())
    username = sock.recv(1024).decode().strip()
    password = sock.recv(1024).decode().strip()

    if username in users and bcrypt.checkpw(password.encode(), users[username].encode()):
        sock.sendall("Login successful.".encode())
        return username
    else:
        sock.sendall("Invalid username or password.".encode())
        return None

def perform_dh_key_exchange(sock):
    parameters = dh.generate_parameters(generator=2, key_size=2048, backend=default_backend())
    param_bytes = parameters.parameter_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.ParameterFormat.PKCS3
    )
    sock.sendall(param_bytes)

    server_private_key = parameters.generate_private_key()
    server_public_key = server_private_key.public_key()

    server_public_bytes = server_public_key.public_bytes(
        serialization.Encoding.PEM,
        serialization.PublicFormat.SubjectPublicKeyInfo
    )

def handle_client(secure_sock, addr):
    try:
        print("[Server] Connection from", addr)
        user = authenticate_client(secure_sock)
        if not user:
            secure_sock.close()
            return

        cipher = perform_dh_key_exchange(secure_sock)
        print("[Server] Secure session established with", user)

        while True:
            encrypted_msg = secure_sock.recv(4096)
            if not encrypted_msg:
                break
            try:
                msg = cipher.decrypt(encrypted_msg).decode()
                print("[" + user + "] " + msg)
                response = cipher.encrypt("Message received securely.".encode())
                secure_sock.send(response)
            except Exception as e:
                print("[Error decrypting message]", e)
                break
    except Exception as e:
        print("[Server error]", e)
    finally:
        secure_sock.close()

def start_server():
    context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
    context.load_cert_chain(certfile="bastion_cert.pem", keyfile="bastion_key.pem")
```

```
def start_server():
    context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
    context.load_cert_chain(certfile="bastion_cert.pem", keyfile="bastion_key.pem")

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((localhost, 4443))
    server_socket.listen(5)
    print("[Server] Listening on port 4443...")

    while True:
        client_sock, addr = server_socket.accept()
        secure_sock = context.wrap_socket(client_sock, server_side=True)
        threading.Thread(target=handle_client, args=(secure_sock, addr)).start()
```

```
if __name__ == "__main__":
    start_server()
```

## Bastion Server V6 (BSV6)

```
File Edit Selection View Go Run Terminal Help ... Untitled (Workspace) EP2 Project > BastionServerSharedDH.py & derive_fernet_key
1 import socket
2 import ssl
3 import threading
4 import bcrypt
5 import json
6 import os
7 import base64
8 from cryptography.fernet import Fernet
9 from cryptography.hazmat.primitives.asymmetric import dh
10 from cryptography.hazmat.backends import default_backend
11 from cryptography.hazmat.primitives import serialization, hashes
12 from cryptography.hazmat.primitives.kdf.kdf4
13 import base64
14
15 USER_DATABASE = "users.db"
16
17 def derive_fernet_key(shared_key):
18     derived_key = KDF(
19         algorithm=hashes.SHA256(),
20         length=32,
21         salt=None,
22         info=b"bastion-session",
23         backend=default_backend()
24     ).derive(shared_key)
25     return base64.urlsafe_b64encode(derived_key)
26
27 def load_users():
28     if not os.path.exists(USER_DATABASE):
29         return {}
30     with open(USER_DATABASE, "r") as file:
31         return json.load(file)
32
33 def authenticate_client(sock):
34     users = load_users()
35     sock.sendall("LOGIN ".encode())
36     username = sock.recv(1024).decode().strip()
37     password = sock.recv(1024).decode().strip()
38
39     if username in users and bcrypt.checkpw(password.encode(), users[username].encode()):
40         sock.sendall("Login successful.".encode())
41         return username
42     else:
43         sock.sendall("Invalid username or password.".encode())
44         return None
45
46 def perform_dh_key_exchange(sock):
47     parameters = dh.generate_parameters(generator=2, key_size=2048, backend=default_backend())
48     param_bytes = parameters.parameter_bytes
49
50     def handle_client(secure_sock, addr):
51         user = authenticate_client(secure_sock)
52         if not user:
53             secure_sock.close()
54             return
55
56         server_private_key = parameters.generate_private_key()
57         server_public_key = server_private_key.public_key()
58         server_public_bytes = server_public_key.public_bytes(
59             serialization.Encoding.PEM,
60             serialization.PublicFormat.SubjectPublicKeyInfo
61         )
62         sock.sendall(len(server_public_bytes).to_bytes(4, "big"))
63         sock.sendall(server_public_bytes)
64
65         client_public_length = int.from_bytes(sock.recv(4), "big")
66         client_public_bytes = sock.recv(client_public_length)
67         client_public_key = serialization.load_pem_public_key(client_public_bytes, backend=default_backend())
68
69         shared_key = server_private_key.exchange(client_public_key)
70         fernet_key = derive_fernet_key(shared_key)
71         return Fernet(fernkey)
72
73     cipher = perform_dh_key_exchange(secure_sock)
74     print("[Server] Secure session established with", user)
75
76     while True:
77         encrypted_msg = secure_sock.recv(4096)
78         if not encrypted_msg:
79             break
80         try:
81             msg = cipher.decrypt(encrypted_msg).decode()
82             print(f"[User] {msg}")
83             response = cipher.encrypt("Message received securely.".encode())
84             secure_sock.send(response)
85         except Exception as e:
86             print(f"[Error decrypting message]", e)
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
```

```
File Edit Selection View Go Run Terminal Help ... Untitled (Workspace) EP2 Project > BastionServerSharedDH.py & derive_fernet_key
1 def handle_client(secure_sock, addr):
2     try:
3         encrypted_msg = secure_sock.recv(4096)
4         if not encrypted_msg:
5             break
6         try:
7             msg = cipher.decrypt(encrypted_msg).decode()
8             print(f"[User] {msg}")
9             response = cipher.encrypt("Message received securely.".encode())
10            secure_sock.send(response)
11        except Exception as e:
12            print(f"[Error decrypting message]", e)
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
```

```
File Edit Selection View Go Run Terminal Help ... Untitled (Workspace) EP2 Project > BastionServerSharedDH.py & derive_fernet_key
1 def start_server():
2     context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
3     context.load_cert_chain(certfile="bastion_cert.pem", keyfile="bastion_key.pem")
4
5     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6     server_socket.bind(("localhost", 4443))
7     server_socket.listen(5)
8     print("[Server] Listening on port 4443...")
9
10    while True:
11        client_sock, addr = server_socket.accept()
12        secure_sock = context.wrap_socket(client_sock, server_side=True)
13        threading.Thread(target=handle_client, args=(secure_sock, addr)).start()
14
15    if __name__ == "__main__":
16        start_server()
```

## Bastion Server V7 (BSV7)

This screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "UNTITLED WORKSPACE".
- Editor:** Displays the file `BastionServerFinal.py`. The code implements a secure communication server using Fernet and HKDF.
- Terminal:** Shows the command-line output of the application running in the background.
- Output:** Shows the application's log messages, including successful login and secure session establishment.
- Problems:** Shows no errors or warnings.
- Debug Console:** Shows the application's log messages.
- Terminal:** Shows the application's log messages.
- Ports:** Shows the application's log messages.
- Bottom Status Bar:** Shows the current file, line number (151), column (Col 1), spaces (Spaces 4), encoding (UTF-8), CR/LF (CR LF), Python (Python 3.10.0), and the date (16.05.2025).

This screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "UNTITLED WORKSPACE".
- Editor:** Displays the file `BastionServerFinal.py`. The code implements a secure communication server using Fernet and HKDF.
- Terminal:** Shows the command-line output of the application running in the background.
- Output:** Shows the application's log messages, including successful login and secure session establishment.
- Problems:** Shows no errors or warnings.
- Debug Console:** Shows the application's log messages.
- Terminal:** Shows the application's log messages.
- Ports:** Shows the application's log messages.
- Bottom Status Bar:** Shows the current file, line number (151), column (Col 1), spaces (Spaces 4), encoding (UTF-8), CR/LF (CR LF), Python (Python 3.10.0), and the date (16.05.2025).

The screenshot shows a PyCharm IDE interface with the following details:

- File Structure:** The left sidebar shows the project structure under "EXPLORER". Files include EP2 Project, UNTITLED (WORKSPACE), and several Python files: BastionClientFinal.py, BastionServerSharedDH.py, BastionServerFinal.py, EP2 Project, BastionChangelog.docx, Bastion\_Changelog.docx, Bastion\_Derj.pem, Bastion\_Client.py, Bastion\_Keypair.py, Bastion\_Server.log, Bastion\_Server.log, BastionServerV3.py, BastionServerV3.py, BastionClientDebug.py, BastionClientFinal.py, BastionClientSharedDH.py, BastionClientV1Basic.py, BastionClientV2adv.py, BastionClientV3.py, BastionClientV4.py, BastionClientV5.py, BastionServerDebug.py, BastionServerFinal.py, BastionServerV3.py, BastionServerV4.py, BastionServerV5.py, EP2\_client.debug.log, Ryan EP2 Proposal.docx, EP2\_server.debug.log, EP2 users.db.
- Code Editor:** The main window displays the content of BastionClientFinal.py. The code implements a Fernet key exchange and password authentication. It uses the socket module for communication and the crypt module for password hashing.
- Terminal:** The bottom-left terminal shows a session where a client connects to a server using the Fernet cipher. The client sends its public key and receives a shared key, then logs in successfully.
- Sidebar:** The right sidebar shows a tree view of the project structure and a "PROBLEMS" tab.
- Bottom Bar:** Shows the current file as BastionClientFinal.py, the line number as 153, column 1, and other system information like space usage and network status.

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like BastionClientFinal.py, BastionServerSharedDH.py, and BastionServerFinal.py.
- Editor:** Displays the code for BastionClientFinal.py, which includes imports for socket, struct, serialization, and base64, along with various functions for key exchange and message handling.
- Bottom Status Bar:** Shows the current file path as 'C:\Users\ryafor01248\Desktop\EP2 Project>', the line number 'Line 153, Col 1', and other system information.
- Bottom Right Corner:** Features a Python interpreter icon with the text 'Python' and 'Python 3.10.0'.

The screenshot shows a Windows desktop environment with a PyCharm IDE window open. The title bar reads "Untitled (Workspace)". The left sidebar displays the "EXPLORER" tab with a tree view of files and folders, including EP2 Project, BastionClientFinal.py, BastionServerSharedDH.py, and BastionServerFinal.py. The main code editor shows the content of BastionClientFinal.py. The status bar at the bottom indicates "In 153, Col 1" and "Python". The taskbar at the bottom has icons for File Explorer, Start, Task View, Search, and several pinned applications.

```
File Edit Selection View Go Run Terminal Help < > ⌘ Untitled (Workspace) ○
```

EXPLORER

UNTITLED (WORKSPACE)

- EP2 Project
  - BastionChangelog.docx
  - Bastion ChangeLog.docx
  - Bastion\_Cert.pem
  - Bastion\_Client.py
  - Bastion\_Keys.pem
  - Bastion\_ServerConfig
  - Bastion\_Server.py
  - BastionServerDebug.py
  - BastionClientFinal.py
  - BastionClientSharedDH.py
  - BastionClientIBasic.py
  - BastionClient2d.py
  - BastionClient3d.py
  - BastionClient4d.py
  - BastionClient5d.py
  - BastionClientDebug.py
  - BastionServerFinal.py
  - BastionServerSharedDH.py
  - BastionServerV1Basic.py
  - BastionServerV2d.py
  - BastionServerV3.py
  - BastionServerV4.py
  - BastionServerV5.py
- Ryan EP2 Proposal.docx
- EP2 server.debug.log
- EP2 server.log
- users.db

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Message ('quit' to exit): quit
PS C:\Users\ryafor@1248\Desktop\EP2\Project>
PS C:\Users\ryafor@1248\Desktop\EP2\Project>
PS C:\Users\ryafor@1248\Desktop\EP2\Project>
PS C:\Users\ryafor@1248\Desktop\EP2\Project> & C:/Users/ryafor@1248/AppData/Local/Programs/Python/3.13/python.exe "c:/Users/ryafor@1248/Desktop/IP2 Project/BastionClientFinal.py"
[Client] Connected securely to server.
[Client] Login successful.
Username: Ryan
Password: Nonoff
[Client] Login successful.
[Client] Secure session established.
Message ('quit' to exit):
[Client] Server: Message received securely.
Message ('quit' to exit): Hello
[Client] Server: Message received securely.
Message ('quit' to exit): hello
[Client] Server: Message received securely.
Message ('quit' to exit): quit
PS C:\Users\ryafor@1248\Desktop\EP2 Project>
```

> OUTLINE > TIMELINE > MYSQL

0 0 0 Connect

File Explorer Start Task View Search

In 153, Col 1 Spaces: 4 UFT-8 CR/LF {} Python 3.13.0 Previewer 13:22 16.05.2025

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure for "EP2 Project" under "UNTITLED (WORKSPACE)". Files include BastionClientFinal.py, BastionServerSharedDH.py, and BastionServerFinal.py.
- Code Editor:** Displays the content of BastionServerFinal.py. The code implements a Bastion Server using the socket library with SSL/TLS support. It uses a certificate file (bastion\_cert.pem) and a key file (bastion\_key.pem). The server listens on port 4443 and handles client connections by spawning a new thread per client. A message "Hello" is sent to each client.
- Terminal:** Shows the command-line output of running the server and client scripts. The server starts and listens on port 4443. The client connects to the server and sends the message "Hello".
- Sidebar:** Includes sections for OUTLINE, TIMELINE, and MYSQL.
- Bottom Bar:** Shows the status bar with "Line 153, Col 1" and other standard OS icons.

## Bastion Server V8 (BSV8)

This screenshot shows the code editor interface for the BastionSpy module. The file `BastionSpy.py` is open in the center pane. The code implements a secure communication server using Fernet and HMAC. It includes functions for loading and saving messages from a local JSON database, deriving Fernet keys from a shared DH secret, and managing user credentials. The code uses Python's standard library modules like `socket`, `ssl`, `threading`, `cryptography`, and `json`. The left sidebar shows the project structure and other files in the workspace.

```
EP2 Project > BastionSpy > ...
...
2  Bastion - Secure Mail Fortress
3
4  This is the server side of Bastion, for secure communication. It replaces insecure plain-text messaging
5  It provides TLS secured connections, user authentication by using bcrypt, Diffie-Hellman key exchange,
6  and encrypted messaging using Fernet plus HMAC
7
8  Author: Ryan Forster
9  ===
10 import socket
11 import ssl
12 import threading
13 import bcrypt
14 import json
15 import os
16 from cryptography.fernet import Fernet
17 from cryptography.hazmat.primitives.asymmetric import dh
18 from cryptography.hazmat.backends import default_backend
19 from cryptography.hazmat.primitives import serialization, hashes
20 from cryptography.hazmat.primitives.hkdf import HKDF
21 import base64
22
23 USER_DATABASE = "users.db"
24 MESSAGE_DATABASE = "messages.db"
25
26 def load_messages():
27     """
28         This is for loading stored messages for all users from local JSON file
29         Returns: dict where the keys are usernames and values are list of messages
30     """
31     if not os.path.exists(MESSAGE_DATABASE):
32         return {}
33     try:
34         with open(MESSAGE_DATABASE, "r") as file:
35             return json.load(file)
36     except json.JSONDecodeError:
37         print("[Server] Warning: messages.db is empty or corrupted. Resetting it.")
38         return {}
39
40 def save_messages(messages):
41     """
42         Saves the current message state to the local JSON file
43         Args: message (dict): Updated message database to write
44     """
45     with open(MESSAGE_DATABASE, "w") as file:
46         json.dump(messages, file, indent=2)
47
48 def derive_fernet_key(shared_key):
49     """
50         It derives a Fernet-compatible encryption key from a shared DH secret
51
52         Args: shared_key : The raw shared key from DH key exchange
53
54         Returns: 32-byte base64-encoded key that is suitable for Fernet
55     """
56     derived_key = HKDF(
57         algorithm=hashes.SHA256(),
58         length=32,
59         salt=b"bastion-session",
60         info="bastion-session",
61         backend=default_backend()
62     ).derive(shared_key)
63     return base64.urlsafe_b64encode(derived_key)
64
65 def load_users():
66     """
67         Load the user credential from a local JSON file
68
69         Returns: dict: Dictionary that maps usernames to hashed passwords
70     """
71     if not os.path.exists(USER_DATABASE):
72         return {}
73     try:
74         with open(USER_DATABASE, "r") as file:
75             return json.load(file)
76     except json.JSONDecodeError:
77         return {}
78
79 def authenticate_client(sock):
80     """
81         Authenticates connection client by using username and password
82
83         Args: sock (ssl.SSLSocket): TLS wrapped client socket
84
85         Returns: str or None: The username if login is successful; None otherwise
86     """
87     users = load_users()
88     sock.sendall("LOGIN".encode())
89     username = sock.recv(1024).decode().strip()
90     password = sock.recv(1024).decode().strip()
91
92     if username in users and bcrypt.checkpw(password.encode(), users[username].encode()):
93         sock.sendall("Login successful.".encode())
94         return username
95     else:
96         sock.sendall("Invalid username or password.".encode())
97         return None
```

This screenshot shows the code editor interface for the BastionSpy module. The file `BastionSpy.py` is open in the center pane. The code implements a secure communication server using Fernet and HMAC. It includes functions for loading and saving messages from a local JSON database, deriving Fernet keys from a shared DH secret, and managing user credentials. The code uses Python's standard library modules like `socket`, `ssl`, `threading`, `cryptography`, and `json`. The left sidebar shows the project structure and other files in the workspace.

```
EP2 Project > BastionSpy > ...
...
40     json.dump(messages, file, indent=2)
41
42     def derive_fernet_key(shared_key):
43         """
44             It derives a Fernet-compatible encryption key from a shared DH secret
45
46             Args: shared_key : The raw shared key from DH key exchange
47
48             Returns: 32-byte base64-encoded key that is suitable for Fernet
49         """
50         derived_key = HKDF(
51             algorithm=hashes.SHA256(),
52             length=32,
53             salt=b"bastion-session",
54             info="bastion-session",
55             backend=default_backend()
56         ).derive(shared_key)
57         return base64.urlsafe_b64encode(derived_key)
58
59     def load_users():
60         """
61             Load the user credential from a local JSON file
62
63             Returns: dict: Dictionary that maps usernames to hashed passwords
64         """
65         if not os.path.exists(USER_DATABASE):
66             return {}
67         try:
68             with open(USER_DATABASE, "r") as file:
69                 return json.load(file)
70         except json.JSONDecodeError:
71             return {}
72
73     def authenticate_client(sock):
74         """
75             Authenticates connection client by using username and password
76
77             Args: sock (ssl.SSLSocket): TLS wrapped client socket
78
79             Returns: str or None: The username if login is successful; None otherwise
80         """
81         users = load_users()
82         sock.sendall("LOGIN".encode())
83         username = sock.recv(1024).decode().strip()
84         password = sock.recv(1024).decode().strip()
85
86         if username in users and bcrypt.checkpw(password.encode(), users[username].encode()):
87             sock.sendall("Login successful.".encode())
88             return username
89         else:
90             sock.sendall("Invalid username or password.".encode())
91             return None
```

File Edit Selection View Go Run Terminal Help

Untitled (Workspace)

```

    ... BastonClientFinal.py BastonSpy x messages.db BastonC.py CNP-ENCR-LAB.yaml BastonServerFinal.py BastonClientV1Basic.py Release Notes: 1.100.2
    EP2 Project > BastonSpy > ...
    91     return None
    92
    93     def perform_dh_key_exchange(sock):
    94         """
    95             Does a secure DH key exchange and derives a Fernet cipher
    96
    97             Args: sock (ssl.SSLSocket): TLS connection to the client
    98
    99             Returns: Cipher object for secure messaging (fernet)
   100
   101            Parameters = dh.generate_parameters(generator=2, key_size=2048, backend=default_backend())
   102            param_bytes = parameters.parameter_bytes(
   103                encoding.serialization.Encoding.PEM,
   104                format=serialization.ParameterFormat.PKCS3
   105            )
   106            sock.sendall(len(param_bytes).to_bytes(4, 'big'))
   107            sock.sendall(param_bytes)
   108
   109            server_private_key = parameters.generate_private_key()
   110            server_public_key = server_private_key.public_key()
   111            server_public_bytes = server_public_key.public_bytes(
   112                serialization.serialization.Encoding.PEM,
   113                serialization.serialization.Format.SubjectPublicKeyInfo
   114            )
   115            sock.sendall(len(server_public_bytes).to_bytes(4, 'big'))
   116            sock.sendall(server_public_bytes)
   117
   118            client_public_length = int.from_bytes(sock.recv(4), 'big')
   119            client_public_bytes = sock.recv(client_public_length)
   120            client_public_key = serialization.load_pem_public_key(client_public_bytes, backend=default_backend())
   121
   122            shared_key = server_private_key.exchange(client_public_key)
   123            fernet_key = derive_fernet_key(shared_key)
   124            return Fernet(fernert_key)
   125
   126        def handle_client(secure_sock, addr):
   127            """
   128                It manages one client session: authenticates, establishes session key, handles encrypted messages
   129
   130                Args: secure_sock (ssl.SSLSocket): Secure client socket
   131                ...
   132                | addr (tuple): IP and Port of client
   133
   134            try:
   135                print("[Server] Connection from", addr)
   136                user = authenticate_client(secure_sock)
   137                if not user:
   138                    secure_sock.close()
   139                    return
   140
   141                shared_key = server_private_key.exchange(client_public_key)
   142                fernet_key = derive_fernet_key(shared_key)
   143                return Fernet(fernert_key)
   144
   145            def handle_client(secure_sock, addr):
   146                cipher = perform_dh_key_exchange(secure_sock)
   147                print("[Server] Secure session established with", user)
   148
   149                messages = load_messages() # This loads inbox database at the start of session
   150
   151                while True:
   152                    encrypted_msg = secure_sock.recv(4096)
   153                    if not encrypted_msg:
   154                        break
   155                    try:
   156                        msg = cipher.decrypt(encrypted_msg).decode()
   157                        print(f"[{user}] ({msg})")
   158
   159                        if msg.startswith("SENDTO:"):
   160                            try:
   161                                parts = msg.split(":", 2)
   162                                recipient = parts[1].strip()
   163                                message = parts[2].strip()
   164
   165                                if recipient not in messages:
   166                                    messages[recipient] = []
   167                                messages[recipient].append(f"From {user}: {message}")
   168                                save_messages(messages)
   169                                response = "Message sent to " + recipient
   170
   171                            except Exception:
   172                                response = "Invalid SENDTO format. Use SENDTO:username:message"
   173
   174                        elif msg == "INBOX":
   175                            inbox = messages.get(user, [])
   176                            if inbox:
   177                                response = "\n".join(inbox)
   178                                messages[user] = [] # Clear the inbox after viewing
   179                                save_messages(messages)
   180                            else:
   181                                response = "Inbox is empty."
   182
   183                        response = cipher.encrypt(response.encode())
   184
   185                    except Exception as e:
   186                        print("[Error decrypting message]", e)
   187
   188                except Exception as e:
   189                    print("[Server error]", e)
   190
   191            finally:
   192                secure_sock.close()
   193
   194        def start_server():
   195            """
   196                Starts the Bastion Server, it listens for TLS connections and makes a thread per client
   197
   198                It uses certificate-based TLS for transport security
   199
   200                context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
   201                context.load_cert_chain(certfile="bastion_cert.pem", keyfile="bastion_key.pem")
   202
   203                server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   204                server_socket.bind(("localhost", 4443))
   205                server_socket.listen(5)
   206                print("[Server] Listening on port 4443...")
   207
   208                while True:
   209                    client_sock, addr = server_socket.accept()
   210                    secure_sock = context.wrap_socket(client_sock, server_side=True)
   211                    threading.Thread(target=handle_client, args=(secure_sock, addr)).start()
   212
   213
   214        if __name__ == "__main__":
   215            start_server()
   
```

OUTLINE TIMELINE MySQL Temperaturen... 1 morgen

File Edit Selection View Go Run Terminal Help

Untitled (Workspace)

File Edit Selection View Go Run Terminal Help

Untitled (Workspace)

```

    ... BastonClientFinal.py BastonSpy x messages.db BastonC.py CNP-ENCR-LAB.yaml BastonServerFinal.py BastonClientV1Basic.py Release Notes: 1.100.2
    EP2 Project > BastonSpy > ...
    128     def handle_client(secure_sock, addr):
    129
    130         cipher = perform_dh_key_exchange(secure_sock)
    131         print("[Server] Secure session established with", user)
    132
    133         messages = load_messages() # This loads inbox database at the start of session
    134
    135         while True:
    136             encrypted_msg = secure_sock.recv(4096)
    137             if not encrypted_msg:
    138                 break
    139             try:
    140                 msg = cipher.decrypt(encrypted_msg).decode()
    141                 print(f"[{user}] ({msg})")
    142
    143                 if msg.startswith("SENDTO:"):
    144                     try:
    145                         parts = msg.split(":", 2)
    146                         recipient = parts[1].strip()
    147                         message = parts[2].strip()
    148
    149                         if recipient not in messages:
    150                             messages[recipient] = []
    151                         messages[recipient].append(f"From {user}: {message}")
    152                         save_messages(messages)
    153                         response = "Message sent to " + recipient
    154
    155                     except Exception:
    156                         response = "Invalid SENDTO format. Use SENDTO:username:message"
    157
    158                 elif msg == "INBOX":
    159                     inbox = messages.get(user, [])
    160                     if inbox:
    161                         response = "\n".join(inbox)
    162                         messages[user] = [] # Clear the inbox after viewing
    163                         save_messages(messages)
    164                     else:
    165                         response = "Inbox is empty."
    166
    167                 response = cipher.encrypt(response.encode())
    168
    169             except Exception as e:
    170                 print("[Error decrypting message]", e)
    171
    172         except Exception as e:
    173             print("[Server error]", e)
    174
    175     finally:
    176         secure_sock.close()
    177
    178
    179     def start_server():
    180
    181         context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
    182         context.load_cert_chain(certfile="bastion_cert.pem", keyfile="bastion_key.pem")
    183
    184         server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    185         server_socket.bind(("localhost", 4443))
    186         server_socket.listen(5)
    187         print("[Server] Listening on port 4443...")
    188
    189         while True:
    190             client_sock, addr = server_socket.accept()
    191             secure_sock = context.wrap_socket(client_sock, server_side=True)
    192             threading.Thread(target=handle_client, args=(secure_sock, addr)).start()
    193
    194
    195     if __name__ == "__main__":
    196         start_server()
    
```

OUTLINE TIMELINE MySQL Temperaturen... 1 morgen

File Edit Selection View Go Run Terminal Help

Untitled (Workspace)

File Edit Selection View Go Run Terminal Help

Untitled (Workspace)

```

    ... BastonClientFinal.py BastonSpy x messages.db BastonC.py CNP-ENCR-LAB.yaml BastonServerFinal.py BastonClientV1Basic.py Release Notes: 1.100.2
    EP2 Project > BastonSpy > ...
    180     secure_sock.close()
    181
    182     def start_server():
    183
    184         context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
    185         context.load_cert_chain(certfile="bastion_cert.pem", keyfile="bastion_key.pem")
    186
    187         server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    188         server_socket.bind(("localhost", 4443))
    189         server_socket.listen(5)
    190         print("[Server] Listening on port 4443...")
    191
    192         while True:
    193             client_sock, addr = server_socket.accept()
    194             secure_sock = context.wrap_socket(client_sock, server_side=True)
    195             threading.Thread(target=handle_client, args=(secure_sock, addr)).start()
    196
    197
    198     if __name__ == "__main__":
    199         start_server()
    
```

OUTLINE TIMELINE MySQL Temperaturen... 1 morgen

File Edit Selection View Go Run Terminal Help

Untitled (Workspace)

## Bastion Server Final (BSF)

```
 BastionServer.py > ...
1  """
2  Bastion - Secure Mail Fortress
3
4  This is the server side of Bastion, for secure communication. It replaces insecure plain-text messaging
5  It provides TLS secured connections, user authentication by using bcrypt, Diffie-Hellman key exchange,
6  and encrypted messaging using Fernet plus HKDF
7
8  Author: Ryan Forster
9  """
10 import socket
11 import ssl
12 import threading
13 import bcrypt
14 import json
15 import os
16 from cryptography.fernet import Fernet
17 from cryptography.hazmat.primitives.asymmetric import dh
18 from cryptography.hazmat.backends import default_backend
19 from cryptography.hazmat.primitives import serialization, hashes
20 from cryptography.hazmat.primitives.kdf.hkdf import HKDF
21 import hmac
22 import hashlib
23 import base64
24 import time
25 from datetime import datetime, timedelta
26
27
28 USER_DATABASE = "users.db"
29 MESSAGE_DATABASE = "messages.db"
30 FAILED_LOG_FILE = "failed_logins.log"
31 LOGIN_ATTEMPTS = {} # This will track failed attempts per username/IP
32 LOCKED_USERS = {} # Temp blocks usernames
33 LOCKED_USERS_FILE = "locked_users.json"
34 SUCCESS_LOG_FILE = "successful_logins.log"
35
36 def load_messages():
37     """

```

```
❷ BastionClient.py ❸ BastionServerStart.py ❹ BastionServer.py X
❷ BastionServer.py > ...
79 def load_users():
80     """Load the user credential from a local JSON file
81
82     Returns: dict: Dictionary that maps usernames to hashed passwords
83     """
84
85     if not os.path.exists(USER_DATABASE):
86         return {}
87     with open(USER_DATABASE, "r") as file:
88         return json.load(file)
89
90 def save_locked_users():
91     """Saves the currently locked users and also their unlock times.
92
93     to_save = {user: unlock_time.isoformat() for user, unlock_time in LOCKED_USERS.items()}
94     with open(LOCKED_USERS_FILE, "w") as file:
95         json.dump(to_save, file)
96
97 def load_locked_users():
98     """Loads locked users from disk at server start
99
100    if os.path.exists(LOCKED_USERS_FILE):
101        try:
102            with open(LOCKED_USERS_FILE, "r") as file:
103                raw = json.load(file)
104                for user, timestr in raw.items():
105                    dt = datetime.fromisoformat(timestr)
106                    if dt > datetime.now(): #still locked
107                        LOCKED_USERS[user] = dt
108        except Exception as e:
109            print("[Server] Warning: Failed to load locked users:", e)
110
111    def log_failed_login(username, ip, reason="FAILED"):
112        """Logs any failed login attempt with timestamps and IP address.
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
```

Ln 22, Col 15 Spaces: 4 UTF-8 CRLF ⚙ Python 3.13.0 20:17  
ENG NO 30/05/2025

```
❷ BastionClient.py ❸ BastionServerStart.py ❹ BastionServer.py X
❷ BastionServer.py > ⌂ load_locked_users
113 def log_failed_login(username, ip, reason="FAILED"):
114     """Logs any failed login attempt with timestamps and IP address.
115
116     with open(FAILED_LOG_FILE, "a") as log:
117         log.write(f"{datetime.now()} | {ip} | {username} | {reason}\n")
118
119 def log_sucessfull_login(username, ip):
120     """It logs sucessfull login attempts with timestamp and IP
121
122     with open(SUCCESS_LOG_FILE, "a") as log:
123         log.write(f"{datetime.now()} | {ip} | {username} | SUCESS\n")
124
125 def authenticate_client(sock):
126     """Authenticates connection client by using username and password
127
128     Args: sock (ssl.SSLocket): TLS wrapped client socket
129
130     Returns: str or None: The username if login is sucessfull; None otherwise"""
131
132     users = load_users()
133     sock.sendall("LOGIN:".encode())
134     username = sock.recv(1024).decode().strip()
135     password = sock.recv(1024).decode().strip()
136     client_ip = sock.getpeername()[0]
137
138     # Checks if user is blocked
139     if username in LOCKED_USERS and datetime.now() < LOCKED_USERS[username]:
140         sock.sendall("Too many failed attempts. Try again later.".encode())
141         log_failed_login(username, client_ip, reason="LOCKED")
142         return None
143
144     # Checks for the credentials
145     if username in users and bcrypt.checkpw(password.encode(), users[username].encode()):
146         log_sucessfull_login(username, client_ip)
147         LOGIN_ATTEMPTS[username] = 0 # Reset attempts on sucess
148         sock.sendall("Login successful.".encode())
149
150
```

Ln 103, Col 13 Spaces: 4 UTF-8 CRLF ⚙ Python 3.13.0 20:18  
ENG NO 30/05/2025

```
152     log_failed_login(username, client_ip)
153     # Count failed attempts
154     LOGIN_ATTEMPTS[username] = LOGIN_ATTEMPTS.get(username, 0) + 1
155
156     # Lock account after 3 failures for 30 seconds
157     if LOGIN_ATTEMPTS[username] >= 3:
158         LOCKED_USERS[username] = datetime.now() + timedelta(seconds=30)
159         save_locked_users()
160         sock.sendall("Too many failed attempts. User locked for 30 seconds.".encode())
161     else:
162         sock.sendall("Invalid username or password.".encode())
163         time.sleep(2) # A small delay to slow down brute forcing
164
165     return None
166
167 def perform_dh_key_exchange(sock):
168     """
169     Does a secure DH key exchange and derives a Fernet cipher
170
171     Args: sock (sslSSLocket): TLS connection to the client
172
173     Returns: Cipher object for secure messaging (fernet)
174     """
175     parameters = dh.generate_parameters(generator=2, key_size=2048, backend=default_backend())
176     param_bytes = parameters.parameter_bytes(
177         encoding=serialization.Encoding.PEM,
178         format=serialization.ParameterFormat.PKCS3
179     )
180     sock.sendall(len(param_bytes).to_bytes(4, 'big'))
181     sock.sendall(param_bytes)
182
183     server_private_key = parameters.generate_private_key()
184     server_public_key = server_private_key.public_key()
185     server_public_bytes = server_public_key.public_bytes(
186         serialization.Encoding.PEM,
187         serialization.PublicFormat.SubjectPublicKeyInfo
188     )
189
190     def verify_hmac(message, received_hmac, key):
191         expected = hmac.new(key, message, hashlib.sha256).digest()
192         return hmac.compare_digest(received_hmac, expected)
193
194     def handle_client(secure_sock, addr):
195         """
196             It manages one client session: authenticates, establishes session key, handles encrypted messages
197
198         Args: secure_sock (sslSSLocket): Secure client socket
199             addr (tuple): IP and Port of client
200
201         try:
202             print("[Server] Connection from", addr)
203             user = authenticate_client(secure_sock)
204             if not user:
205                 secure_sock.close()
206                 return
207
208             cipher, hmac_key = perform_dh_key_exchange(secure_sock)
209             print("[Server] Secure session established with", user)
210
211             messages = load_messages() # This loads inbox database at the start of session
212
213             while True:
214                 encrypted_msg = secure_sock.recv(4096)
215                 if not encrypted msg:
```

Ln 103, Col 13 Spaces: 4 UTF-8 CRLF {} Python 3.13.0 20:18  
ENG NO 30/05/2025

```
196                 encrypted_msg = secure_sock.recv(4096)
197                 if not encrypted msg:
198                     continue
199
200                 decrypted_msg = cipher.decrypt(encrypted_msg)
201                 decrypted_msg = decrypted_msg.decode('utf-8')
202
203                 if decrypted_msg == "exit":
204                     secure_sock.close()
205                     break
206
207                 if decrypted_msg in messages:
208                     messages[decrypted_msg] += 1
209                 else:
210                     messages[decrypted_msg] = 1
211
212             save_messages(messages)
213
214             print("[Server] Session ended with", user)
215
216             break
217
218     save_locked_users()
219
220     print("[Server] Server is now running")
221
222     while True:
223         command = input("Enter command: ")
224
225         if command == "load messages":
226             messages = load_messages()
227             print("Messages loaded successfully")
228
229         elif command == "save messages":
230             save_messages(messages)
231             print("Messages saved successfully")
232
233         elif command == "exit":
234             break
235
236         else:
237             print("Unknown command. Please enter a valid command")
```

Ln 180, Col 30 Spaces: 4 UTF-8 CRLF {} Python 3.13.0 20:18  
ENG NO 30/05/2025



```
BastionClient.py BastionServerStart.py BastionServer.py
271     def start_server():
272         print(f"[Server] TLS handshake failed from {addr}: {e}")
273         client_sock.close() # Close unwrapped socket
274
275     if __name__ == "__main__":
276         start_server()
```

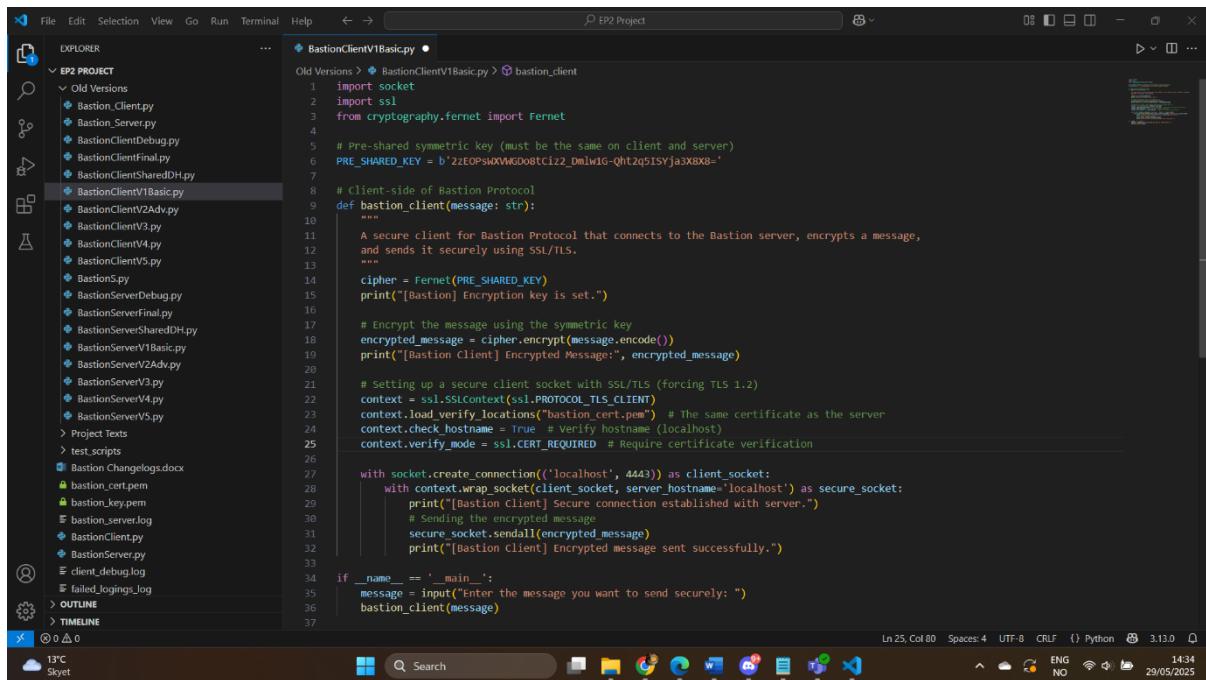
## BastionServerStart (BSS)

```
BastionClient.py BastionServerStart.py BastionServer.py
# BastionServerStart.py > register_user
1 """
2 Bastion - Secure Mail Fortress
3
4 This is the server side of Bastion, running this code starts the server or registers a new user to the users.db file.
5 By writing "register" a new menu prompts where username and password need to be input.
6 When done adding users, run the code again and write "server" to start the server, it will run the code from the
7 BastionServer.py file.
8
9 Author: Ryan Forster
10 """
11 import bcrypt
12 import json
13 import getpass
14
15 USER_DATABASE = "users.db"
16
17 def register_user():
18     users = {}
19     try:
20         with open(USER_DATABASE, "r") as f:
21             users = json.load(f)
22     except:
23         pass
24
25     print("==== Register New User ===")
26     username = input("Enter new username: ").strip()
27     if username in users:
28         print("Username already exists.")
29         return
30
31     while True:
32         password = getpass.getpass("Enter password: ").strip()
33         confirm = getpass.getpass("Confirm password: ").strip()
34         if password == confirm:
35             break
36         print("Passwords do not match. Try again.")
```

Ln 19, Col 9 Spaces: 4 UTF-8 LF {} Python 3.13.0 20:19  
ENG NO 30/05/2025

```
BastionClient.py BastionServerStart.py BastionServer.py
# BastionServerStart.py > register_user
17 def register_user():
18     print("Passwords do not match. Try again.")
19
20     hashed = bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode()
21     users[username] = hashed
22
23     with open(USER_DATABASE, "w") as f:
24         json.dump(users, f)
25     print("User registered successfully.")
26
27     if __name__ == "__main__":
28         mode = input("Start server or register user? (server/register): ").strip().lower()
29         if mode == "register":
30             register_user()
31         else:
32             from BastionServer import start_server
33             start_server()
```

## Bastion Client V1 (BCV1)

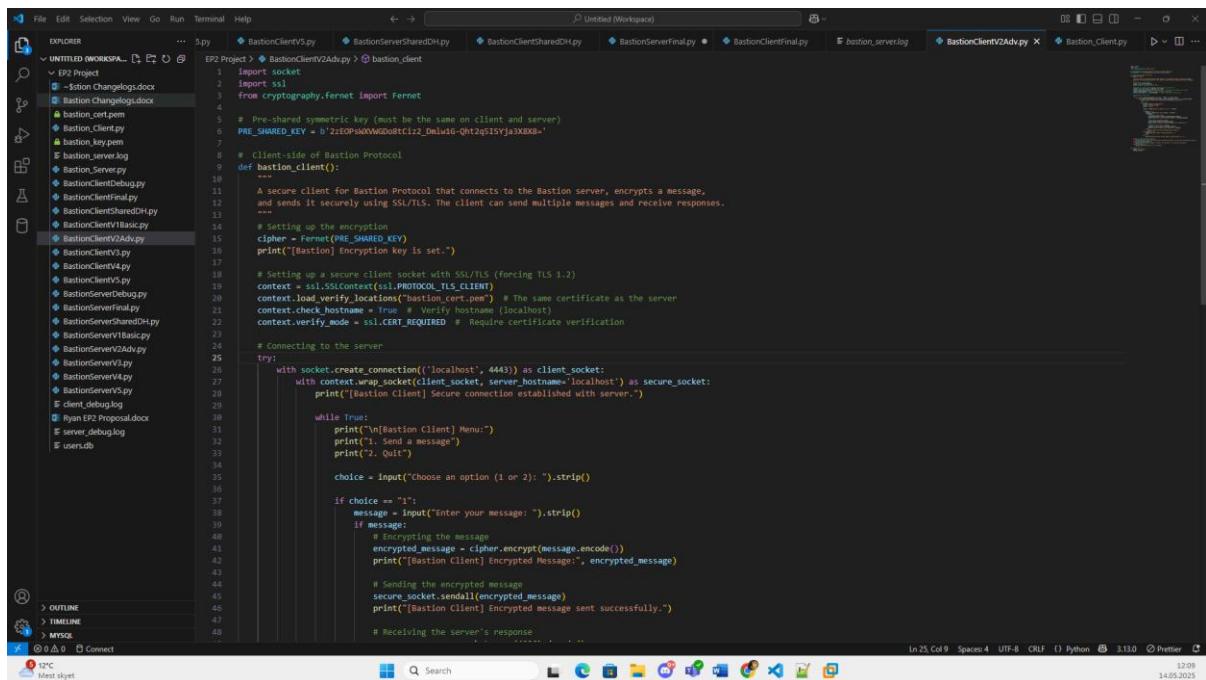


```
File Edit Selection View Go Run Terminal Help < > EP2 Project ... Explorer Old Versions Bastion.Client.py Bastion_Server.py BastionClientDebug.py BastionClientFinal.py BastionClientSharedDH.py BastionClientV1Basic.py BastionClientV2Adv.py BastionClientV3.py BastionClientV4.py BastionClientV5.py BastionS.py BastionServerDebug.py BastionServerFinal.py BastionServerShared.py BastionServerSharedDH.py BastionServerV1Basic.py BastionServerV2Adv.py BastionServerV3.py BastionServerV4.py BastionServerV5.py Project Text test_scripts Bastion_Changelog.docx bastion.cert.pem bastion.key.pem bastion_server.log BastionClient.py BastionServer.py client_debug.log failed_logins.log OUTLINE TIMELINE 13°C SkyNet
```

```
BastionClientV1Basic.py
Old Versions > BastionClientV1Basic.py > bastion_client
1 import socket
2 import ssl
3 from cryptography.fernet import Fernet
4
5 # Pre-shared symmetric key (must be the same on client and server)
6 PRE_SHARED_KEY = b'2EOPsXWMD08tC1z2_DmlwIG-Qhtq5ISYja3XBX8='
7
8 # Client-side of Bastion Protocol
9 def bastion_client(message: str):
10     """
11         A secure client for Bastion Protocol that connects to the Bastion server, encrypts a message,
12         and sends it securely using SSL/TLS.
13     """
14     cipher = Fernet(PRE_SHARED_KEY)
15     print("[Bastion Client] Encryption key is set.")
16
17     # Encrypt the message using the symmetric key
18     encrypted_message = cipher.encrypt(message.encode())
19     print("[Bastion Client] Encrypted Message:", encrypted_message)
20
21     # Setting up a secure client socket with SSL/TLS (forcing TLS 1.2)
22     context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
23     context.load_verify_locations("bastion.cert.pem") # The same certificate as the server
24     context.check_hostname = True # Verify hostname (localhost)
25     context.verify_mode = ssl.CERT_REQUIRED # Require certificate verification
26
27     with socket.create_connection(("localhost", 4443)) as client_socket:
28         with context.wrap_socket(client_socket, server_hostname="localhost") as secure_socket:
29             print("[Bastion Client] Secure connection established with server.")
30             # Sending the encrypted message
31             secure_socket.sendall(encrypted_message)
32             print("[Bastion Client] Encrypted message sent successfully.")
33
34     if __name__ == '__main__':
35         message = input("Enter the message you want to send securely: ")
36         bastion_client(message)
```

Ln 25, Col 80 Spaces:4 UTF-8 CRLF () Python ENG NO 14:34 29/05/2025

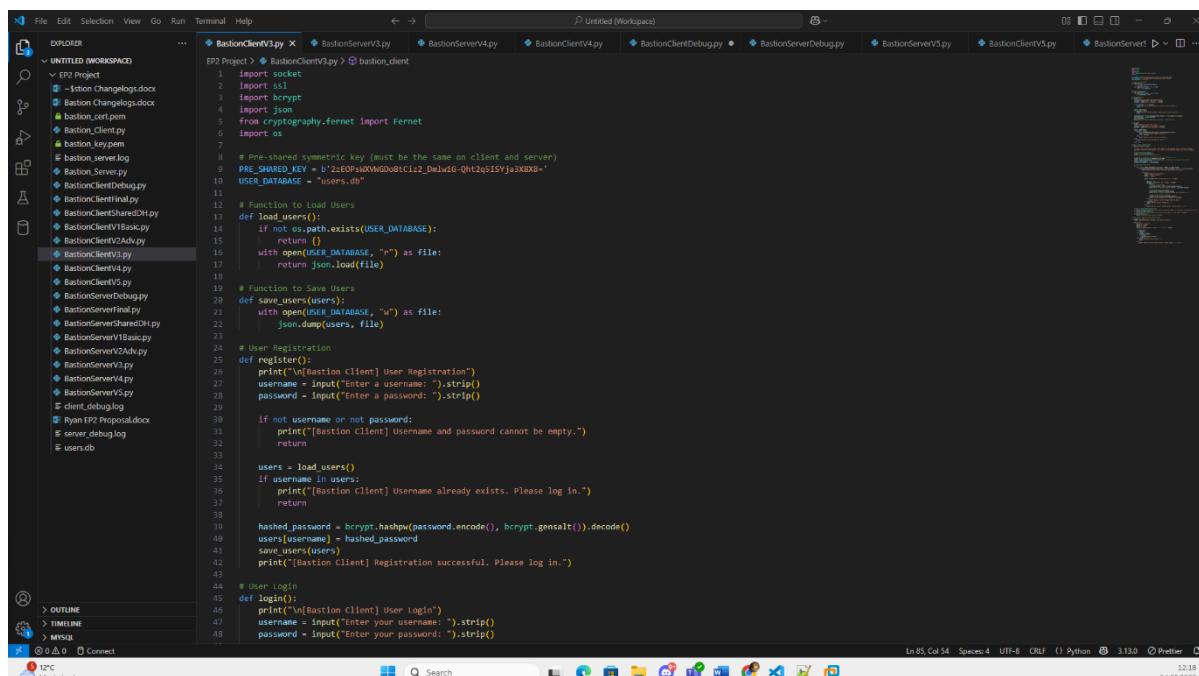
## Bastion Client V2 (BCV2)



```
File Edit Selection View Go Run Terminal Help < > Untitled (Workspace) ... Explorer EP2 Project Bastion_Changelog.docx BastionClientV2Adv.py bastion_client
1 import socket
2 import ssl
3 from cryptography.fernet import Fernet
4
5 # Pre-shared symmetric key (must be the same on client and server)
6 PRE_SHARED_KEY = b'2EOPsXWMD08tC1z2_DmlwIG-Qhtq5ISYja3XBX8='
7
8 # Client-side of Bastion Protocol
9 def bastion_client():
10     """
11         A secure client for Bastion Protocol that connects to the Bastion server, encrypts a message,
12         and sends it securely using SSL/TLS. The client can send multiple messages and receive responses.
13     """
14     # Setting up the encryption
15     cipher = Fernet(PRE_SHARED_KEY)
16     print("[Bastion Client] Encryption key is set.")
17
18     # Setting up a secure client socket with SSL/TLS (forcing TLS 1.2)
19     context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
20     context.load_verify_locations("bastion.cert.pem") # The same certificate as the server
21     context.check_hostname = True # Verify hostname (localhost)
22     context.verify_mode = ssl.CERT_REQUIRED # Require certificate verification
23
24     # Connecting to the server
25     try:
26         with socket.create_connection(("localhost", 4443)) as client_socket:
27             with context.wrap_socket(client_socket, server_hostname="localhost") as secure_socket:
28                 print("[Bastion Client] Secure connection established with server.")
29
30                 while True:
31                     print("\n[Bastion Client] Menu:")
32                     print("1. Send a message")
33                     print("2. Quit")
34
35                     choice = input("Choose an option (1 or 2): ").strip()
36
37                     if choice == "1":
38                         message = input("Enter your message: ").strip()
39                     if message:
40                         # Encrypting the message
41                         encrypted_message = cipher.encrypt(message.encode())
42                         print("[Bastion Client] Encrypted Message:", encrypted_message)
43
44                         # Sending the encrypted message
45                         secure_socket.sendall(encrypted_message)
46                         print("[Bastion Client] Encrypted message sent successfully.")
47
48                         # Receiving the server's response
49                         response = secure_socket.recv(1024).decode()
50                         print("[Bastion Client] Received Response:", response)
```

Ln 25, Col 9 Spaces:4 UTF-8 CRLF () Python ENG NO 12:09 14/05/2025

## Bastion Client V3 (BCV3)



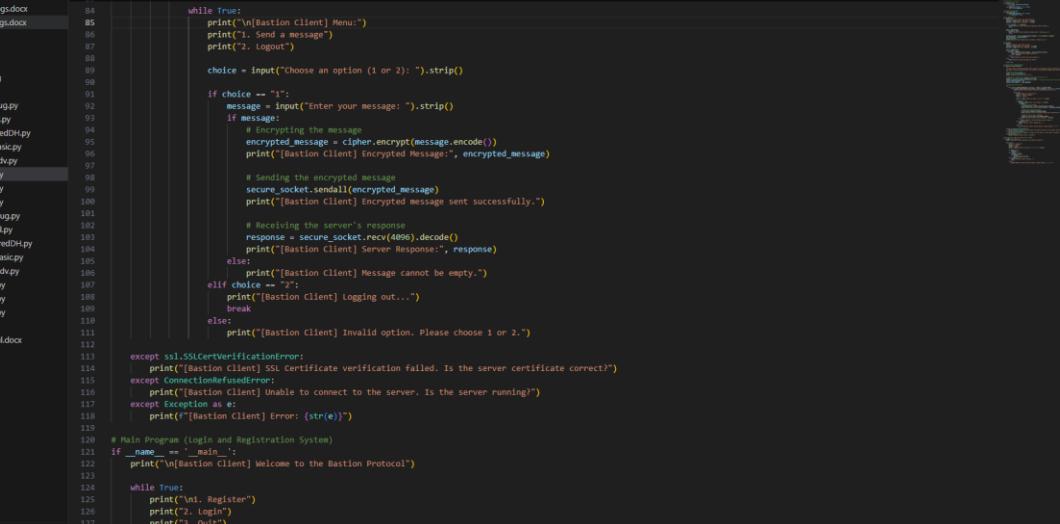
The terminal window displays the following output:

```
On branch main
Your branch is up-to-date with 'origin/main'.
Changes to be committed:
  (use "git commit -a" to include in current commit)

    modified:   .gitignore
    modified:   README.md
```

The screenshot shows a code editor with multiple tabs open, primarily focused on a file named `BastionClient3.py`. The code is a Python script for a Bastion Client, version 3. It includes imports for `socket`, `ssl`, `contextlib`, `select`, and `hashlib`. The script defines a `User` class with methods for `login` and `check_password`. It also includes a `SecureClient` class with methods for `connect`, `send_message`, and `receive_message`. The `main` function handles command-line arguments and starts the client. Other files visible in the sidebar include `BastionServerV3.py`, `BastionServerV4.py`, `BastionClientV4.py`, `BastionClientDebug.py`, `BastionServerDebug.py`, `BastionServerV5.py`, `BastionClientV5.py`, and `BastionServerV6.py`. The bottom status bar shows the current file is `BastionClient3.py` at line 85, column 54, with 1300 characters and 130 lines of code.

```
File Edit Selection View Go Run Terminal Help ← → Untitled (Workspace) BastionClient3.py BastionServerV3.py BastionServerV4.py BastionClientV4.py BastionClientDebug.py BastionServerDebug.py BastionServerV5.py BastionClientV5.py BastionServerV6.py ... EP2 Project BastionChangelog.docx bastion_cert.pem bastion_client.py bastion_key.pem bastion_server.log Bastion_Server.py BastionClientDebug.py BastionClientFinal.py BastionClientSharedDH.py BastionClientTlsiscy.py BastionClientV2dvp.py BastionClientV3.py BastionClientV4.py BastionClientV5.py BastionServerDebug.py BastionServerFinal.py BastionServerSharedDH.py BastionServerTlsiscy.py BastionServerV2dvp.py BastionServerV3.py BastionServerV4.py BastionServerV5.py BastionServerV6.py ... Ryan EP2 Proposal.docx ... server.debug.log users.db ... # User Login 43 44 def login(): 45     print("\n[Bastion Client] User Login") 46     username = input("Enter your username: ").strip() 47     password = input("Enter your password: ").strip() 48 49     users = load_users() 50     if username in users: 51         if bcrypt.checkpw(password.encode(), users[username].encode()): 52             print("[Bastion Client] login successful.") 53             return username 54         else: 55             print("[Bastion Client] Incorrect password.") 56     else: 57         print("[Bastion Client] Username not found.") 58 59     return None 60 61 # Secure Client (Authenticated) 62 def bastion_client(username): 63     """ 64     A secure client for Bastion Protocol that connects to the Bastion server, encrypts a message, 65     and sends it securely using SSL/TLS. The client can send multiple messages and receive responses. 66     """ 67 68     # Setting up the encryption 69     cipher = Fernet(PUBLIC_KEY) 70     print("[Bastion] Encryption key is set.") 71 72     # Setting up a secure client socket with SSL/TLS (forcing TLS 1.2) 73     context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT) 74     context.load_verify_locations('bastion_cert.pem') # The same certificate as the server 75     context.check_hostname = True 76     context.verify_mode = ssl.CERT_REQUIRED 77 78     # Connecting to the server 79     try: 80         with socket.create_connection(("localhost", 4443)) as client_socket: 81             with context.wrap_socket(client_socket, server_hostname='localhost') as secure_socket: 82                 print("[Bastion Client] Secure connection established with server.") 83 84                 while True: 85                     print("\n[Bastion Client] Menu:") 86                     print("1. Send a message") 87                     print("2. Logout") 88 89                     choice = input("Choose an option (1 or 2): ").strip() 90 91                     if choice == "1": 92                         message = input("Enter your message: ") 93                         encrypted_message = cipher.encrypt(message.encode()) 94                         secure_socket.write(encrypted_message) 95 96                     elif choice == "2": 97                         break 98 99     except Exception as e: 100         print(f"An error occurred: {e}") 101 102 if __name__ == "__main__": 103     bastion_client("user")
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER ... BastionClientV3.py BastonServer3.py BastionServer4.py BastionClientV4.py BastionClientDebug.py BastonServerDebug.py BastionServer5.py BastionClient5.py BastionServer6.py
UNTITLED WORKSPACE... EP2 Project BastionChangelog.docx
BastionChangelog.docx
BastionClient.py
BastionClientV3.py
BastionClientKeygen.py
BastionServer.log
BastionServerV3.py
BastionServerV4.py
BastionServerDebug.py
BastionClientFinal.py
BastionClientSharedH.py
BastionClientVBasic.py
BastionClient2adv.py
BastionClient3.py
BastionClient4.py
BastionClient5.py
BastionServerFinal.py
BastionServerDebug.py
BastionClientSharedH.py
BastionServerVBBasic.py
BastionServer2adv.py
BastionServerV3.py
BastionServerV4.py
BastionServerV5.py
client_debug.log
Ryan EP2 Proposal.docx
E server_debug.log
E users.db
I2C MySQL
OUTLINE TIMELINE Connect
BastionClientV3.py X BastonClientV4.py > Bastion_client
EP2 Project BastionClientV4.py > Bastion_client
63     def bastion_client(username):
64         while True:
65             print("\n[Bastion Client] Menu:")
66             print("1. Log in")
67             print("2. Logout")
68
69             choice = input("Choose an option (1 or 2): ").strip()
70
71             if choice == "1":
72                 message = input("Enter your message: ").strip()
73                 if message:
74                     # Encrypting the message
75                     encrypted_message = cipher.encrypt(message.encode())
76                     print("[Bastion Client] Encrypted Message:", encrypted_message)
77
78                     # Sending the encrypted message
79                     secure_socket.sendall(encrypted_message)
80                     print("[Bastion Client] Encrypted message sent successfully.")
81
82                     # Receiving the server's response
83                     response = secure_socket.recv(4096).decode()
84                     print("[Bastion Client] Server Response:", response)
85
86                     print("[Bastion Client] Message cannot be empty.")
87                 elif choice == "2":
88                     print("[Bastion Client] Logging out...")
89                     break
90
91             else:
92                 print("[Bastion Client] Invalid option. Please choose 1 or 2.")
93
94             except ssl.SSLCertVerificationError:
95                 print("[Bastion Client] SSL Certificate verification failed. Is the server certificate correct?")
96             except ConnectionRefusedError:
97                 print("[Bastion Client] Unable to connect to the server. Is the server running?")
98             except Exception as e:
99                 print("[Bastion Client] Error: (str(e))")
100
101     # Main Program (Login and Registration System)
102     if __name__ == '__main__':
103         print("[Bastion Client] Welcome to the Bastion Protocol")
104
105         while True:
106             print("\n1. Register")
107             print("2. Login")
108             print("3. Quit")
109             choice = input("Choose an option (1, 2, or 3): ").strip()
110
111             if choice == "1":
```

## Bastion Client V4 (BCV4)

The screenshot shows the PyCharm IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Toolbars:** Standard toolbar with icons for file operations.
- SIDE BAR:** Explorer (left) showing the project structure and files, including EP2 Project, BastionClientV3.py, BastionServerV3.py, BastionServerV4.py, BastionClientV4.py, BastionClientDebug.py, BastionClientFinal.py, BastionClientSharedH.py, BastionClientVBasic.py, BastionClientV2Adv.py, BastionClientV3.py, BastionClientV4.py, BastionClientV5.py, BastionServerDebug.py, BastionServerFinal.py, BastionServerSharedH.py, BastionServerVBasic.py, BastionServerV2Adv.py, BastionServerV3.py, BastionServerV4.py, BastionServerV5.py, and users.db. A file named "Ryan EP2 Proposal.docx" is also listed.
- EDITOR:** The main code editor displays the content of the BastionClientV4.py file. The code implements a secure client for the Bastion Protocol, utilizing SSL/TLS (TLS 1.2), Fernet encryption, and the Diffie-Hellman key exchange. It handles user authentication via a local socket connection, sends and receives messages in JSON format, and manages session keys.
- STATUS BAR:** Shows the current file is BastionClientV4.py, line 35, column 58. It also indicates 3130 lines of code, 1330 characters, and the Python language is selected.
- Bottom Icons:** Includes icons for file operations like Open, Save, and Close, as well as other standard OS-like icons.

```
File Edit Selection View Go Run Terminal Help Untitled (Workspace) BastonClientV3.py BastonServerV3.py BastonServerV4.py BastonClientV4.py BastonClientDebug.py BastonServerDebug.py BastonServerV5.py BastonClientV5.py BastonServer.py ... EP2 Project > BastonClientV4.py > bastion_client.py

4 def bastion_client():
5     print("[Bastion Client] [response]")
6
7     choice = input("Enter choice: ")
8
9     if choice == "1":
10         username = input("Enter username: ").strip()
11         password = input("Enter password: ").strip()
12
13         secure_socket.sendall(password.encode())
14         response = secure_socket.recv(1024).decode()
15         print("[Bastion Client] [response]")
16
17         if "Login successful" in response:
18             print("[Bastion Client] Welcome, [username].")
19             secure_communication(secure_socket, username)
20         else:
21             print("[Bastion Client] login failed.")
22
23     elif choice == "2":
24         print("[Bastion Client] Exiting...")
25         break
26     else:
27         print("[Bastion Client] Invalid option. Please choose 1, 2, or 3.")
28
29 except ssl.SSLCertVerificationError:
30     print("[Bastion Client] SSL Certificate verification failed. Is the server certificate correct?")
31 except ConnectionRefusedError:
32     print("[Bastion Client] Unable to connect to the server. Is the server running?")
33 except Exception as e:
34     print("[Bastion Client] Error: [str(e)]")
35
36 # Secure Communication After Login
37 def secure_communication(secure_socket, username):
38
39     """
40     Manages secure communication with the server using a dynamically generated session key (DHE).
41     """
42
43     print("\n[Bastion Client] Establishing a secure session...")
44
45     # Diffie-Hellman Key Exchange (DHE)
46     parameters = dh.generate_parameters(generator=2, key_size=2048)
47     client_private_key = parameters.generate_private_key()
48     client_public_key = client_private_key.public_key()
49
50     # Send Client Public Key to Server
51     secure_socket.sendall(client_public_key.public_bytes(Encoding.PEM, PublicFormat.SubjectPublicKeyInfo))
52
53     # Receive Server Public Key
54     server_public_bytes = secure_socket.recv(4096)
55
56     # Generate a shared session key
57     shared_key = client_private_key.exchange(server_public_key)
58     session_key = Fernet.generate_key()
59     cipher = Fernet(session_key)
60
61     print("[Bastion Client] Secure session established using DHE.")
62
63     while True:
64         print("\n[Bastion Client] Menu:")
65         print("1. Send a message")
66         print("2. View Inbox")
67         print("3. Logout")
68
69         choice = input("Choose an option (1, 2, or 3): ").strip()
70
71         if choice == "1":
72             message = input("Enter your message: ").strip()
73             if message:
74                 # Encrypting the message with the session key
75                 encrypted_message = cipher.encrypt(message.encode())
76                 secure_socket.sendall(encrypted_message)
77                 print("[Bastion Client] Encrypted message sent.")
78
79             # Receiving the server's response
80             response = secure_socket.recv(4096).decode()
81             print("[Bastion Client] Server Response: ", response)
82
83         elif choice == "2":
84             # Requesting the inbox from the server
85             secure_socket.sendall("INBOX".encode())
86             inbox = secure_socket.recv(4096).decode()
87             print("\n[Bastion Client] Inbox for [username]:\nInbox")
88
89         elif choice == "3":
90             print("[Bastion Client] Logging out...")
91             break
92
93         else:
94             print("[Bastion Client] Invalid option. Please choose 1, 2, or 3.")

In 35, Col 58 Spaces 4 - UTF-8 CRLF ⓘ Python 3.11.0 ⌂ Prettier ⓘ 12:21 14.05.2025
```

```
File Edit Selection View Go Run Terminal Help Untitled (Workspace) BastonClientV3.py BastonServerV3.py BastonServerV4.py BastonClientV4.py BastonClientDebug.py BastonServerDebug.py BastonServerV5.py BastonClientV5.py BastonServer.py ... EP2 Project > BastonClientV4.py > bastion_client.py

65         # Requesting the inbox from the server
66         secure_socket.sendall("INBOX".encode())
67         inbox = secure_socket.recv(4096).decode()
68         print("\n[Bastion Client] Inbox for [username]:\nInbox")
69
70     elif choice == "3":
71         print("[Bastion Client] Logging out...")
72         break
73
74     else:
75         print("[Bastion Client] Invalid option. Please choose 1, 2, or 3.")

In 35, Col 58 Spaces 4 - UTF-8 CRLF ⓘ Python 3.11.0 ⌂ Prettier ⓘ 12:23 14.05.2025
```

## Bastion Client V5 (BCV5)

```
File Edit Selection View Go Run Terminal Help < > Untitled (Workspace) Explorer ... BastionClientV3.py BastionServerV3.py BastionServerV4.py BastionClientV4.py BastionClientDebug.py BastionServerDebug.py BastionServerV5.py BastionClientV5.py BastionServerV5.py

EXPLORER ... UNTITLED (WORKSPACE) EP2 Project ... BastionClientV3.py > Bastion_client
29 def bastion_client():
30     print("[Client]", login_status)
31     if "successful" not in login_status:
32         return
33
34     cipher = perform_db_key_exchange(secure_sock)
35     print("[Client] Secure session established.")
36
37     while True:
38         msg = input("Message ('quit' to exit): ").strip()
39         if msg.lower() == "quit":
40             break
41         encrypted = cipher.encrypt(msg.encode())
42         secure_sock.sendall(encrypted)
43         response = secure_sock.recv(4096)
44         print("[Client] Server:", cipher.decrypt(response).decode())
45
46     if __name__ == "__main__":
47         bastion_client()
```

Bastion Client V6 (BCV6)

```
File Edit Selection View Go Run Terminal Help < - > Untitled (Workspace) BastionServerV4.py BastionClientV4.py BastionClientDebug.py BastionServerDebug.py BastionServerV5.py BastionClientV5.py BastionServerSharedDH.py BastionClientSharedDH.py

... EP2 Project BastionChangelog.docx Bastion_cert.pem Bastion_Client.py Bastion_Server.py Bastion_ServerFinal.py BastionClientV4.py BastionClientSharedDH.py BastionClientV5Basic.py BastionServerV2Adv.py BastionServerV3.py BastionServerV4.py BastionServerV5.py Ryan EP2 Proposal.docx E_client_debug.log E_server_debug.log E_users.db

EXPLORER BastionServerV4.py BastionClientV4.py BastionClientSharedDH.py > perform_db_key_exchange

1 import socket
2 import ssl
3 from cryptography.hazmat import Fernet
4 from cryptography.hazmat.primitives.asymmetric import dh
5 from cryptography.hazmat.primitives import serialization, hashes
6 from cryptography.hazmat.backends import default_backend
7 from cryptography.hazmat.primitives.kdf.hkdf import HKDF
8 import base64
9 import os
10
11 def derive_fernet_key(shared_key):
12     derived_key = HKDF(
13         algorithm=hashes.SHA256(),
14         length=32,
15         salt=b'long',
16         info=b'bastion-session',
17         backend=default_backend()
18     ).derive(shared_key)
19     return base64.urlsafe_b64encode(derived_key)
20
21 def receive_block(sock):
22     length = int.from_bytes(sock.recv(4), 'big')
23     return sock.recv(length)
24
25 def perform_db_key_exchange(sock):
26     param_bytes = receive_block(sock)
27     parameters = serialization.load_pem_parameters(param_bytes, backend=default_backend())
28
29     private_key = parameters.generate_private_key()
30     public_key = private_key.public_key()
31
32     server_public_bytes = receive_block(sock)
33     server_public_key = serialization.load_pem_public_key(server_public_bytes, backend=default_backend())
34
35     client_public_bytes = public_key.public_bytes(
36         serialization.Encoding.PEM,
37         serialization.PublicFormat.SubjectPublicKeyInfo
38     )
39     sock.sendall(len(client_public_bytes).to_bytes(4, 'big'))
40     sock.sendall(client_public_bytes)
41
42     shared_key = private_key.exchange(server_public_key)
43     fernet_key = derive_fernet_key(shared_key)
44     return Fernet(fernet_key)
45
46 def bastion_client():
47     context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
48     context.load_verify_locations("bastion_cert.pem")
```

```
File Edit Selection View Go Run Terminal Help < - > Untitled (Workspace) BastionServerV4.py BastionClientV4.py BastionClientDebug.py BastionServerDebug.py BastionServerV5.py BastionClientV5.py BastionServerSharedDH.py BastionClientSharedDH.py

EXPLORER ... EP2 Project BastionClientSharedDH.py > perform_dh_key_exchange

untitled (workspace) ... EP2 Project BastionClientSharedDH.py > perform_dh_key_exchange

46     def bastion_client():
47         context = ssl.create_default_context(ssl.PROTTOCOL_TLS_CLIENT)
48         context.load_verify_locations('bastion_cert.pem')
49         context.check_hostname = True
50         context.verify_mode = ssl.CERT_REQUIRED
51
52         with socket.create_connection(('localhost', 4443)) as sock:
53             with context.wrap_socket(sock, server_hostname='localhost') as secure_sock:
54                 print("Client connected securely to server.")
55
56                 response = secure_sock.recv(1024).decode()
57                 print("Client", response)
58                 username = input("Username: ").strip()
59                 password = input("Password: ").strip()
60                 secure_sock.sendall(username.encode())
61                 secure_sock.sendall(password.encode())
62
63                 login_status = secure_sock.recv(1024).decode()
64                 print("Client", login_status)
65                 if "successful" not in login_status:
66                     return
67
68                 cipher = perform_dh_key_exchange(secure_sock)
69                 print("Client Secure session established.")
70
71                 while True:
72                     msg = input("Message ('quit' to exit): ").strip()
73                     if msg.lower() == "quit":
74                         break
75
76                     encrypted = cipher.encrypt(msg.encode())
77                     secure_sock.sendall(encrypted)
78                     response = secure_sock.recv(4096)
79                     print("Client Server:", cipher.decrypt(response).decode())
80
81             if __name__ == "__main__":
82                 bastion_client()
```

## Bastion Client V7 (BCV7)

The screenshot shows a Python development environment with the following details:

- File Explorer (EXPLORER):** Shows the project structure for EP2 Project, including files like BastionChangelog.docx, BastionClient.py, BastionClientDebug.py, BastionClientFinal.py, BastionClientSharedDH.py, BastionClientShared4.py, BastionClientShared5.py, BastionClientV1.py, BastionClientV2.py, BastionClientV3.py, BastionClientV4.py, BastionClientV5.py, BastionServerShared.py, BastionServerFinal.py, BastionServerShared4.py, BastionServerShared5.py, BastionServerV1Basic.py, BastionServerV2Ady.py, BastionServerV3.py, BastionServerV4.py, BastionServerV5.py, client.debug.log, Ryan EP2 Proposal.docx, server.debug.log, and users.db.
- Code Editor (UNTITLED (WORKSPACE)):** The current file is BastionClientFinal.py. The code implements a client-side part for Bastion, a secure communication system using Fernet and HKDF. It includes imports for socket, ssl, cryptography.fernet, cryptography.hazmat.primitives.asymmetric, cryptography.hazmat.primitives.serialization, cryptography.hazmat.backends, and cryptography.hazmat.primitives.kdf.hkdf. The code defines a derive\_fernet\_key function that takes a shared key and derives a Fernet-compatible key using HKDF from the shared DH secret. It also includes a main loop for user interaction.
- Terminal:** The terminal shows the execution of the client script. It connects to the server, logs in as 'Ryan', and exchanges messages with the server.
- Bottom Status Bar:** Shows file paths (C:\Users\ryafor01248\Desktop\EP2 Project), the Python interpreter (Python 3.13.0), and other system information (Un� 60, Col 38, Spaces: 4, UFT-8, CRU).

The screenshot shows a Python development environment with the following details:

- File Explorer:** Shows the project structure for "EP2 Project" with files like `BastionClientFinal.py`, `BastionServerSharedDH.py`, and `BastionServerFinal.py`.
- Code Editor:** Displays the `BastionClientFinal.py` file containing code for a secure communication protocol using Diffie-Hellman key exchange and Fernet encryption.
- Terminal:** Shows the command-line interface with the Python interpreter running the client script. The terminal output includes messages from the server indicating secure session establishment and the client responding with "Hello".
- Status Bar:** Shows the current file is `BastionClientFinal.py`, release notes are at 1.100.2, and the Python version is 3.11.0.

The screenshot shows a Windows desktop environment with several open windows. The most prominent is a terminal window titled 'Terminal' which contains Python code for a secure communication protocol. Below the terminal is a file explorer window titled 'EXPLORER' showing a directory structure for an EP2 Project. Other windows visible include a browser tab for 'Untitled (Workspace)', a taskbar with various icons, and a system tray at the bottom.

```
File Edit Selection View Go Run Terminal Help
... BastionClientFinal.py BastionServerSharedDH.py BastionServerFinal.py Release Notes 1.1002
EP2 Project > BastionClientFinal.py > perform_db_key_exchange
54 def perform_db_key_exchange(sock):
55     """
56     Returns: A cipher for encrypted communication
57     """
58     param_bytes = receive_block(sock)
59     parameters = serialization.load_pem_parameters(param_bytes, backend=default_backend())
60
61     private_key = parameters.generate_private_key()
62     public_key = private_key.public_key()
63
64     server_public_bytes = receive_block(sock)
65     server_public_key = serialization.load_pem_public_key(server_public_bytes, backend=default_backend())
66
67     client_public_bytes = public_key.public_bytes(
68         serialization.Encoding.X509,
69         serialization.PublicFormat.SubjectPublicKeyInfo
70     )
71
72     sock.sendall(len(client_public_bytes).to_bytes(4, 'big'))
73     sock.sendall(client_public_bytes)
74
75     shared_key = private_key.exchange(server_public_key)
76     fernet_key = derive_fernet_key(shared_key)
77
78     return Fernet(fernet_key)
79
80 def bastion_client():
81     """
82     Connects to the Bastion Server, it logs in safely, does the key exchange and,
83     allows the user to send encrypted messages over a secured channel.
84     """
85
86     context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
87
88     [Client] Server: Message received securely.
89     Message ('quit' to exit): Yes
90     [Client] Server: Message received securely.
91     Message ('quit' to exit): quit
92     PS C:\Users\ryafor01248\Desktop\EP2 Project>
93     PS C:\Users\ryafor01248\Desktop\EP2 Project>
94     PS C:\Users\ryafor01248\Desktop\EP2 Project>
95     PS C:\Users\ryafor01248\Desktop\EP2 Project & C:\Users\ryafor01248\AppData\Local\Programs\Python\Python33\python.exe "c:/Users/ryafor01248/Desktop/EP2 Project/BastionClientFinal.py"
96     [Client] Connected securely to server.
97     [Client] LOGIN:
98     Username: Ryan
99     Password: Ryaforoff
100    [Client] Login successful.
101    [Client] Secure session established.
102    Message ('quit' to exit):
103    [Client] Server: Message received securely.
104    Message ('quit' to exit): Hello
105    [Client] Server: Message received securely.
106    Message ('quit' to exit): hello
107
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+ Python: Bas...
+ Python: Bas...
+ Python: Bas...
```

File Edit Selection View Go Run Terminal Help

EXPLORER

UNTITLED (WORKSPACE)

EP2 Project

- EP2 ChangeLogs.docx
- EP2 Configuration
- EP2 ClientApp
- EP2 Client.py
- EP2 ClientServerLog
- EP2 Server.py
- EP2 ServerDebug.py
- EP2 ClientFinal.py
- EP2 ClientSharedDH.py
- EP2 ClientVBasic.py
- EP2 ClientV2Adv.py
- EP2 ClientV3.py
- EP2 ClientV4.py
- EP2 ClientV5.py
- EP2 ServerDebug.py
- EP2 ServerFinal.py
- EP2 ClientSharedDH.py
- EP2 ServerVBasic.py
- EP2 ServerV2Adv.py
- EP2 ServerV3.py
- EP2 ServerV4.py
- EP2 ServerV5.py
- EP client\_debug.log
- Ryan EP2 Proposal.docx
- EP server\_debug.log
- Users.xls

BastionClientFinal.py

```
def bastion_client():
    """Connects to the Bastion Server. It logs in safely, does the key exchange and, allows the user to send encrypted messages over a secured channel.

    """
    context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
    context.verify_mode = ssl.CERT_REQUIRED
    context.load_verify_locations("bastion_cert.pem")
    context.check_hostname = True
    context.verify_mode = ssl.CERT_REQUIRED

    with socket.create_connection(("localhost", 4443)) as sock:
        with context.wrap_socket(sock, server_hostname="localhost") as secure_sock:
            print("[Client] Connected securely to server.")

            response = secure_sock.recv(1024).decode()
            print("[Client]", response)
            username = input("Username: ").strip()
            password = input("Password: ").strip()
            secure_sock.sendall(username.encode())
            secure_sock.sendall(password.encode())

            login_status = secure_sock.recv(1024).decode()
            print("[Client]", login_status)
            if "successful" not in login_status:
                return

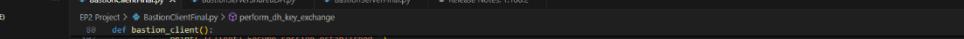
            cipher = perform_db_key_exchange(secure_sock)
            print("[Client] Secure session established.")

    [Client] Server: Message received securely.
    Hello
    [Client] Server: Message received securely.
    Message ('quit' to exit): quit
    PS C:\Users\ryafor01248\Desktop\EP2 Project>
    PS C:\Users\ryafor01248\Desktop\EP2 Project>
    PS C:\Users\ryafor01248\Desktop\EP2 Project>
    PS C:\Users\ryafor01248\Desktop\EP2 Project & C:/Users/ryafor01248/appData/local/Programs/Python/Python313/python.exe "C:/Users/ryafor01248/Desktop/EP2 Project/BastionClientFinal.py"
    [Client] connected securely to server.
    [Client]
    Username: Ryan
    Password: Noroff
    [Client] Login successful.
    [Client] Secure session established.
    Message ('quit' to exit):
    [Client] Server: Message received securely.
    Message ('quit' to exit): Hello
    [Client] Server: Message received securely.
    Message ('quit' to exit): Hello
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

[Client] Server: Message received securely.  
Hello  
[Client] Server: Message received securely.  
Message ('quit' to exit): quit  
PS C:\Users\ryafor01248\Desktop\EP2 Project>  
PS C:\Users\ryafor01248\Desktop\EP2 Project>  
PS C:\Users\ryafor01248\Desktop\EP2 Project>  
PS C:\Users\ryafor01248\Desktop\EP2 Project & C:/Users/ryafor01248/appData/local/Programs/Python/Python313/python.exe "C:/Users/ryafor01248/Desktop/EP2 Project/BastionClientFinal.py"  
[Client] connected securely to server.  
[Client]  
Username: Ryan  
Password: Noroff  
[Client] Login successful.  
[Client] Secure session established.  
Message ('quit' to exit):  
[Client] Server: Message received securely.  
Message ('quit' to exit): Hello  
[Client] Server: Message received securely.  
Message ('quit' to exit): Hello

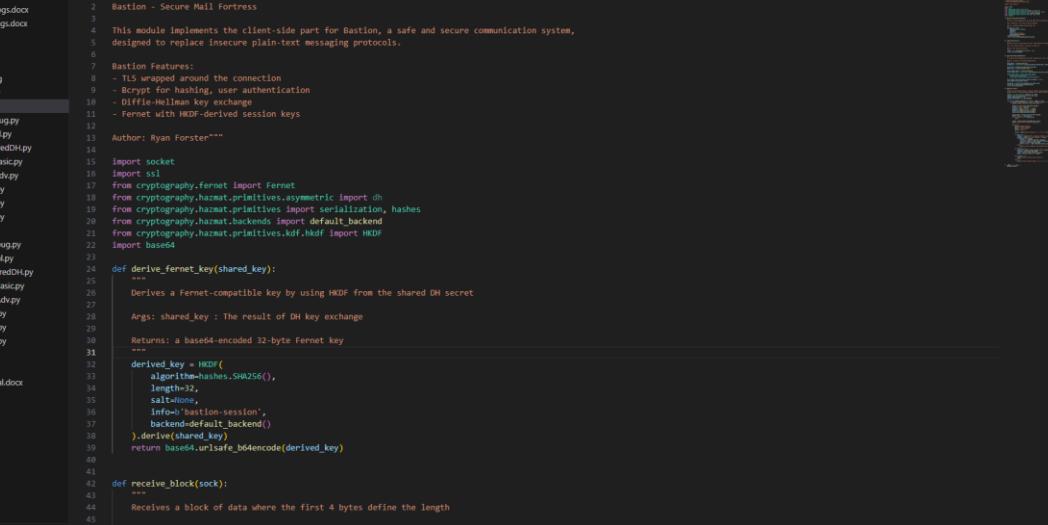
In 60. Col 38 Spaces: 4 UTF-8 CR/LF () Python 3.13.0 Prettier



The screenshot shows a Windows desktop environment. In the foreground, there is a terminal window titled "Untitled (Workspace)" containing Python code. The code is related to a Bastion Client application, specifically handling session establishment and secure communication using encryption and sockets. Below the terminal, the file explorer shows the project structure under "EP2 Project". The project includes files like "BastionClient.py", "BastionClientFinal.py", "BastionClientSharedDH.py", "BastionClientV1Basic.py", "BastionClientV2Adv.py", and "BastionClientV3.py". There are also configuration files such as "bastion\_client.ini" and "bastion\_server.ini", and logs like "bastion\_client.log" and "bastion\_server.log". A "CHANGELOG.md" file is present in the root directory.

```
File Edit Selection View Go Run Terminal Help ← → Untitled (Workspace)
EXPLORER ... BastionClientFinal.py BastionClientSharedDH.py BastionServerFinal.py Release Notes: 1.100.2
EP2 Project BastionClientFinal.py perform_dh_key_exchange
bastion_client.py
108     def bastion_client():
109         print("[Client] secure session established... ")
110
111         while True:
112             msg = input("Message (quit to exit): ").strip()
113             if msg == "quit":
114                 break
115             encrypted = cipher.encrypt(msg.encode())
116             secure_sock.sendall(encrypted)
117             response = secure_sock.recv(4096)
118             print(f"[Client] Server: {cipher.decrypt(response).decode()}")
119
120     if __name__ == "__main__":
121         bastion_client()
BastionClientSharedDH.py
BastionClientV1Basic.py
BastionClientV2Adv.py
BastionClientV3.py
```

Bastion Client V8 (BCV8)



```
EP2 Project > BastionCpy > derive_fernet_key
1
2   Bastion - Secure Mail Fortress
3
4   This module implements the client-side part for Bastion, a safe and secure communication system,
5   designed to replace insecure plain-text messaging protocols.
6
7   Bastion Features:
8     - TLS wrapped around the connection
9     - Bcrypt for hashing, user authentication
10    - Diffie-Hellman key exchange
11    - Fernet with HKDF-derived session keys
12
13    Author: Ryan Forster"""
14
15    import socket
16    import ssl
17    from cryptography.fernet import Fernet
18    from cryptography.hazmat.primitives.asymmetric import dh
19    from cryptography.hazmat.primitives import serialization, hashes
20    from cryptography.hazmat.backends import default_backend
21    from cryptography.hazmat.primitives.kdf.hkdf import HKDF
22    import base64
23
24    def derive_fernet_key(shared_key):
25        """
26            Derives a Fernet-compatible key by using HKDF from the shared DH secret
27
28        Args: shared_key : The result of DH key exchange
29
30        Returns: a base64-encoded 32-byte Fernet key
31        """
32
33        derived_key = HKDF(
34            algorithm=hashes.SHA256(),
35            length=32,
36            salt=None,
37            info=b'bastion-session',
38            backend=default_backend()
39        ).derive(shared_key)
40        return base64.urlsafe_b64encode(derived_key)
41
42    def receive_block(sock):
43        """
44            Receives a block of data where the first 4 bytes define the length
45
46        Args: sock (socket.socket): The socket to read from
47
48        Returns: The received data block
49        """
50
```

The screenshot shows a Windows desktop environment with a Python development setup. The main window is a code editor displaying a file named 'BastionClient.py'. The code implements a TLS client, specifically a Bastion Client, using the `ssl` module. It includes functions for receiving blocks of data, performing DH key exchange, generating private keys, and sending encrypted messages. The code also handles SSL context creation and socket wrapping. The status bar at the bottom indicates the file has 37 lines, 1130 characters, and is using Python syntax. A terminal window at the bottom shows a command prompt. The taskbar at the bottom has icons for various applications like File Explorer, Task View, and Start.

```
File Edit Selection View Go Run Terminal Help ← → ⌘ Untitled (Workspace) BastionClientFinal.py BastionS.py messages.db BastionCopy CCNP-ENCLR-LAB.yaml BastionServerFinal.py BastionClientV1Basic.py Release Notes: 1.100.2
```

```
EP2 Project BastionChangelog.docx Bastion_Cert.pem Bastion_Client.py Bastion_Key.py Bastion_Server.log Bastion.py BastionCopy BastionClientDebug.py BastionClientFinal.py BastionClientShared.py BastionClientV1Basic.py BastionClient2adv.py BastionClientV3.py BastionClientV4.py BastionClientV5.py BastionS.py BastionServerDebug.py BastionServerFinal.py BastionServerShared.py BastionServerV1Basic.py BastionServerV2adv.py BastionServerV3.py BastionServerV4.py BastionServerV5.py client_debug.log messages.db Ryan EP2 Proposal.docx server_debug.log users.db
```

```
... BastionClientFinal.py BastionS.py messages.db BastionCopy CCNP-ENCLR-LAB.yaml BastionServerFinal.py BastionClientV1Basic.py Release Notes: 1.100.2
```

```
EP2 Project BastionChangelog.docx Bastion_Cert.pem Bastion_Client.py Bastion_Key.py Bastion_Server.log Bastion.py BastionCopy BastionClientDebug.py BastionClientFinal.py BastionClientShared.py BastionClientV1Basic.py BastionClient2adv.py BastionClientV3.py BastionClientV4.py BastionClientV5.py BastionS.py BastionServerDebug.py BastionServerFinal.py BastionServerShared.py BastionServerV1Basic.py BastionServerV2adv.py BastionServerV3.py BastionServerV4.py BastionServerV5.py client_debug.log messages.db Ryan EP2 Proposal.docx server_debug.log users.db
```

```
... BastionClientFinal.py BastionS.py messages.db BastionCopy CCNP-ENCLR-LAB.yaml BastionServerFinal.py BastionClientV1Basic.py Release Notes: 1.100.2
```

```
42     def receive_block(sock):
```

```
43         Returns: The received data block
```

```
44         ---
```

```
45         length = int.from_bytes(sock.recv(4), 'big')
```

```
46         return sock.recv(length)
```

```
47
```

```
48     def perform_dh_key_exchange(sock):
```

```
49         """
```

```
50             It receives the DH parameters and server's publick key, then it sends the clients public key
```

```
51         Returns: A cipher for encrypted communication
```

```
52         ---
```

```
53         param_bytes = receive_block(sock)
```

```
54         parameters = serialization.load_pem_parameters(param_bytes, backend=default_backend())
```

```
55
```

```
56         private_key = parameters.generate_private_key()
```

```
57         public_key = private_key.public_key()
```

```
58
```

```
59         server_public_bytes = receive_block(sock)
```

```
60         server_public_key = serialization.load_pem_public_key(server_public_bytes, backend=default_backend())
```

```
61
```

```
62         client_public_bytes = public_key.public_bytes(
```

```
63             serialization.Encoding.PEM,
```

```
64             serialization.PublicFormat.SubjectPublicKeyInfo
```

```
65         )
```

```
66         (variable) client_public_bytes: bytes
```

```
67     
```

```
68     sock.sendall(client_public_bytes.to_bytes(4, 'big'))
```

```
69     sock.sendall(client_public_bytes)
```

```
70
```

```
71     shared_key = private_key.exchange(server_public_key)
```

```
72     fernet_key = derive.fernet_key(shared_key)
```

```
73     return Fernet(fernet_key)
```

```
74
```

```
75     def bastion_client():
```

```
76         """
```

```
77             Connects to the Bastion Server, it logs in safely, does the key exchange and,
```

```
78             allows the user to send encrypted messages over a secured channel.
```

```
79             ---
```

```
80             context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
```

```
81             context.load_verify_locations("bastion_cert.pem")
```

```
82             context.check_hostname = True
```

```
83             context.verify_mode = ssl.CERT_REQUIRED
```

```
84
```

```
85             with socket.create_connection(("localhost", 4443)) as sock:
```

```
86                 with context.wrap_socket(sock, server_hostname="localhost") as secure_sock:
```

```
87                     print("(client) Connected securely to server.")
```

```
88
```

```
89
```

```
90
```

```
91
```

```
92
```

```
93
```

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure for "EP2 Project > BastionCopy". Files include EP2 Project, BastionChangelog.log, BastionChangelog.docx, Bastion\_client.py, Bastion\_keygen.py, Bastion\_server.log, Bastion\_Server.py, Bastion.py, BastionClientDebug.py, BastionClientInbox.py, BastionClientShared.py, BastionClientV1Basic.py, BastionClientV2Adv.py, BastionClientV3.py, BastionClientV4.py, BastionClientV5.py, BastionServerFinal.py, BastionServerShared.py, BastionServerV1Basic.py, BastionServerV2Adv.py, BastionServerV3.py, BastionServerV4.py, BastionServerV5.py, and users.db.
- Code Editor (Center):** The main editor window displays the content of Bastion.py. The code implements a secure socket connection, handles user authentication, performs a DH key exchange, and provides a menu for sending messages or viewing the inbox. It also handles file operations like reading from messages.db and writing to users.db.
- Terminal (Bottom Left):** Shows the command "Advanced on bra...".
- Status Bar (Bottom Right):** Displays "Line 81, Col 8" and "Release Notes: 1.100.2".

```
EP2 Project > BastionCopy > bastion_client
  88     def bastion_client():
  89         pass
  90         break
  91     else:
  92         print("[Client] Invalid choice. Try 1, 2, or 3.")
  93
  94     if __name__ == "__main__":
  95         bastion_client()
```

## Bastion Client Final (BCF)

```
  BastionClient.py  BastionServerStart.py  BastionServer.py
1  """
2  Bastion - Secure Mail Fortress
3
4  This module implements the client-side part for Bastion, a safe and secure communication system,
5  designed to replace insecure plain-text messaging protocols.
6
7  Bastion Features:
8  - TLS wrapped around the connection
9  - Bcrypt for hashing, user authentication
10 - Diffie-Hellman key exchange
11 - Fernet with HKDF-derived session keys
12
13 Author: Ryan Forster"""
14
15 import socket
16 import ssl
17 from cryptography.fernet import Fernet
18 from cryptography.hazmat.primitives.asymmetric import dh
19 from cryptography.hazmat.primitives import serialization, hashes
20 from cryptography.hazmat.backends import default_backend
21 from cryptography.hazmat.primitives.kdf.hkdf import HKDF
22 import hmac
23 import hashlib
24 import base64
25 import getpass
26
27 def derive_keys(shared_key):
28     """
29     Derives a Fernet-compatible key by using HKDF from the shared DH secret
30     It also does it for HMAC.
31
32     Args: shared_key : The result of DH key exchange
33
34     Returns: a base64-encoded 32-byte Fernet key
35     """
36     hkdf = HKDF(
37         algorithm=hashes.SHA256(),
38         length=32, # 32 bits for Fernet and 32 for HMAC
39         salt=None,
40         info=b'bastion-session',
41         backend=default_backend()
42     ).derive(shared_key)
43     return base64.urlsafe_b64encode(hkdf[:32]), hkdf[32:]
44
45 def receive_block(sock):
46     """
47     Receives a block of data where the first 4 bytes define the length
48
49     Args: sock (socket.socket): The socket to read from
50
51     Returns: The received data block
52     """
53     length = int.from_bytes(sock.recv(4), 'big')
54     return sock.recv(length)
55
56 def perform_dh_key_exchange(sock):
57     """
58     Performs DH key exchange, returns Fernet cipher and HMAC key.
59     """
60     param_bytes = receive_block(sock)
61     parameters = serialization.load_pem_parameters(param_bytes, backend=default_backend())
62     private_key = parameters.generate_private_key()
63     public_key = private_key.public_key()
64
65     server_public_bytes = receive_block(sock)
66     server_public_key = serialization.load_pem_public_key(server_public_bytes, backend=default_backend())
67
68     client_public_bytes = public_key.public_bytes(
69         serialization.Encoding.PEM,
70         serialization.PublicFormat.SubjectPublicKeyInfo
71     )

```

Ln 153, Col 1 Spaces: 4 UTF-8 LF () Python 3.13.0 20:26  
ENG NO 30/05/2025

```
  BastionClient.py  BastionServerStart.py  BastionServer.py
27     def derive_keys(shared_key):
28         hkdf = HKDF(
29             algorithm=hashes.SHA256(),
30             length=32, # 32 bits for Fernet and 32 for HMAC
31             salt=None,
32             info=b'bastion-session',
33             backend=default_backend()
34         ).derive(shared_key)
35         return base64.urlsafe_b64encode(hkdf[:32]), hkdf[32:]
36
37     def receive_block(sock):
38         """
39         Receives a block of data where the first 4 bytes define the length
40
41         Args: sock (socket.socket): The socket to read from
42
43         Returns: The received data block
44         """
45         length = int.from_bytes(sock.recv(4), 'big')
46         return sock.recv(length)
47
48     def perform_dh_key_exchange(sock):
49         """
50         Performs DH key exchange, returns Fernet cipher and HMAC key.
51         """
52         param_bytes = receive_block(sock)
53         parameters = serialization.load_pem_parameters(param_bytes, backend=default_backend())
54         private_key = parameters.generate_private_key()
55         public_key = private_key.public_key()
56
57         server_public_bytes = receive_block(sock)
58         server_public_key = serialization.load_pem_public_key(server_public_bytes, backend=default_backend())
59
60         client_public_bytes = public_key.public_bytes(
61             serialization.Encoding.PEM,
62             serialization.PublicFormat.SubjectPublicKeyInfo
63         )
64
65         server_public_bytes = receive_block(sock)
66         server_public_key = serialization.load_pem_public_key(server_public_bytes, backend=default_backend())
67
68         client_public_bytes = public_key.public_bytes(
69             serialization.Encoding.PEM,
70             serialization.PublicFormat.SubjectPublicKeyInfo
71         )

```

Ln 153, Col 1 Spaces: 4 UTF-8 LF () Python 3.13.0 20:26  
ENG NO 30/05/2025

```
  BastionClient.py X  BastionServerStart.py  BastionServer.py ...
  BastionClient.py > ⚡ perform_dh_key_exchange
56  def perform_dh_key_exchange(sock):
57      # Select a random prime number and a generator
58      # ... (code omitted)
59
60      shared_key = private_key.exchange(server_public_key)
61      fernet_key, hmac_key = derive_keys(shared_key)
62
63      return Fernet(fernet_key), hmac_key
64
65  def attach_hmac(message_bytes, key):
66      """
67          This creates an HMAC of the message and prepends it to the encrypted message
68      """
69
70      mac = hmac.new(key, message_bytes, hashlib.sha256).digest()
71
72      return mac + message_bytes
73
74
75  def bastion_client():
76      """
77          Connects to the Bastion Server, it logs in safely, does the key exchange and,
78          allows the user to send encrypted messages over a secured channel.
79      """
80
81      context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
82      context.load_verify_locations("bastion_cert.pem")
83      context.check_hostname = True
84      context.verify_mode = ssl.CERT_REQUIRED
85
86      server_ip = input("Enter the IP address of the Bastion server (leave empty for localhost): ").strip()
87      if not server_ip:
88          server_ip = "localhost"
89
90      try:
91          with socket.create_connection((server_ip, 4443)) as sock:
92              with context.wrap_socket(sock, server_hostname=server_ip) as secure_sock:
93                  print("[Client] connected securely to server.")
94                  response = secure_sock.recv(1024).decode()
95                  print("[Client]", response)
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
```

Ln 69, Col 36 Spaces: 4 UTF-8 LF () Python 3.13.0 2027 ENG NO 30/05/2025

```
  BastionClient.py X  BastionServerStart.py  BastionServer.py ...
  BastionClient.py > bastion_client
86  def bastion_client():
87      with context.wrap_socket(sock, server_hostname=server_ip) as secure_sock:
88          print("[Client] Connected securely to server.")
89          response = secure_sock.recv(1024).decode()
90          print("[Client]", response)
91          username = input("Username: ").strip()
92          password = getpass.getpass("Password: ").strip()
93          secure_sock.sendall(username.encode())
94          secure_sock.sendall(password.encode())
95
96          login_status = secure_sock.recv(1024).decode()
97          print("[Client]", login_status)
98          if "successful" not in login_status:
99              return
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
```

Ln 101, Col 66 Spaces: 4 UTF-8 LF () Python 3.13.0 2027 ENG NO 30/05/2025

```

BastionClient.py X BastionServerStart.py BastionServer.py
BastionClient.py > bastion_client
86     def bastion_client():
87         print("[Client] Recipient and message cannot be empty. ")
88
89         elif choice == "2":
90             encrypted = cipher.encrypt("INBOX".encode())
91             secure_sock.sendall(attach_hmac(encrypted, hmac_key))
92             response = secure_sock.recv(4096)
93             inbox = cipher.decrypt(response).decode()
94             print("\n[Client] Inbox:\n" + inbox)
95
96         elif choice == "3":
97             print("[Client] Exiting secure session.")
98             break
99
100        else:
101            print("[Client] Invalid choice. Try 1, 2, or 3.")
102
103    except Exception as e:
104        print(f"[Client] Connection failed: {e}")
105
106 if __name__ == "__main__":
107     bastion_client()
108
109

```

## 8.2 Bastion Testing

### Bastion Client Certificate (False) (BCCF)

```

return Fernet(fernet_key)

def bastion_client():
    """
    Connects to the Bastion Server, it logs in safely, does the key exchange and,
    allows the user to send encrypted messages over a secured channel.
    """
    context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
    context.load_verify_locations("bastion_cert.pem")
    context.check_hostname = False
    context.verify_mode = ssl.CERT_REQUIRED

    with socket.create_connection(('localhost', 4443)) as sock:
        with context.wrap_socket(sock, server_hostname='localhost') as secure_sock:
            print("[Client] Connected securely to server.")

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```

### Bastion Client Certificate (TRUE) (BCCT)

```

return Fernet(fernet_key)

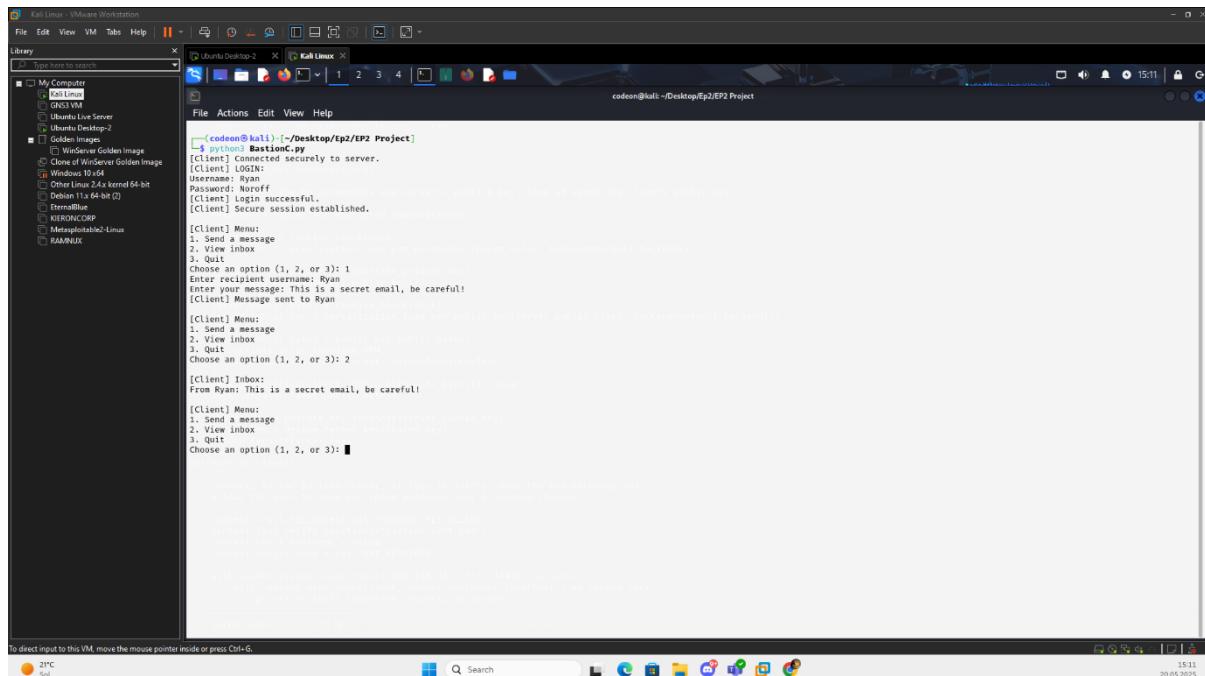
def bastion_client():
    """
    Connects to the Bastion Server, it logs in safely, does the key exchange and,
    allows the user to send encrypted messages over a secured channel.
    """
    context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
    context.load_verify_locations("bastion_cert.pem")
    context.check_hostname = True
    context.verify_mode = ssl.CERT_REQUIRED

    with socket.create_connection(('localhost', 4443)) as sock:
        with context.wrap_socket(sock, server_hostname='localhost') as secure_sock:
            print("[Client] Connected securely to server.")

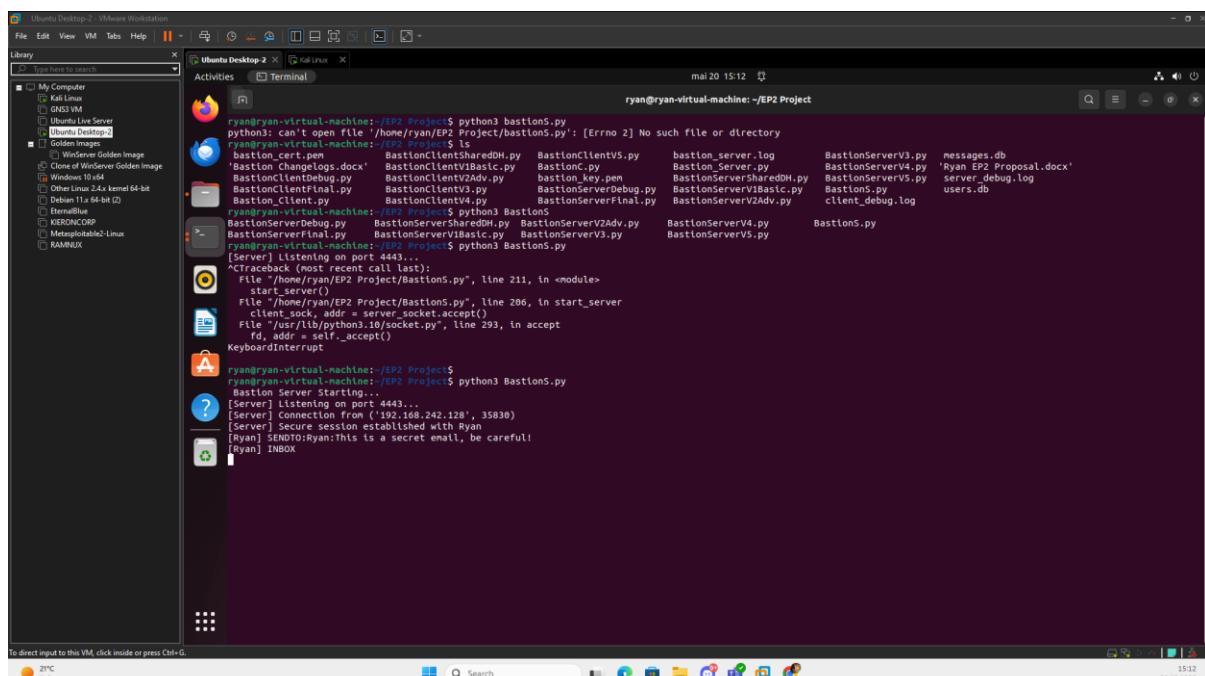
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```

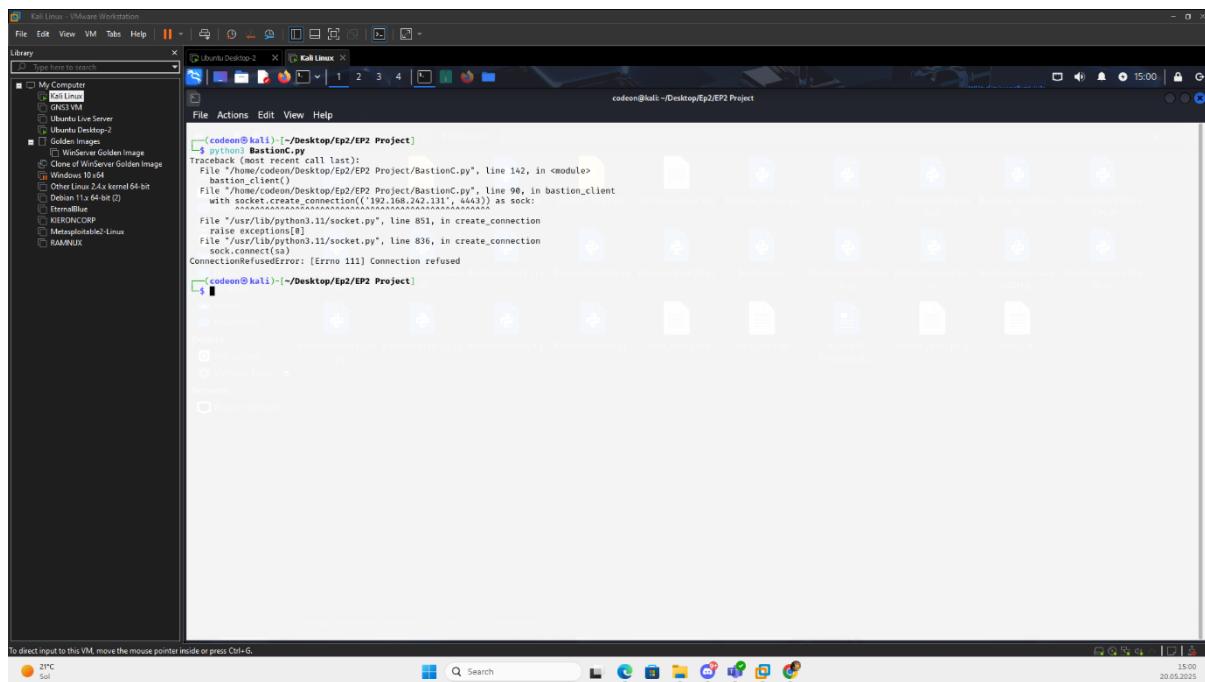
## Bastion Client Basic Functionality (BCBF1)



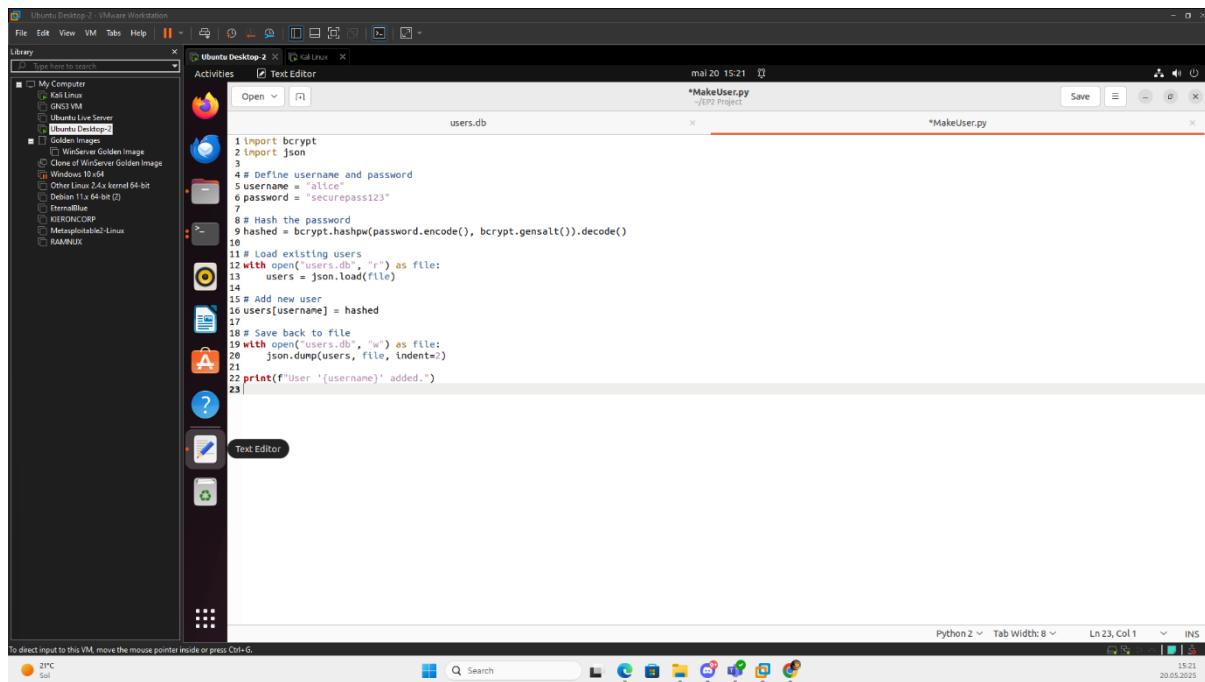
## Bastion Server Basic Functionality (BSBF)



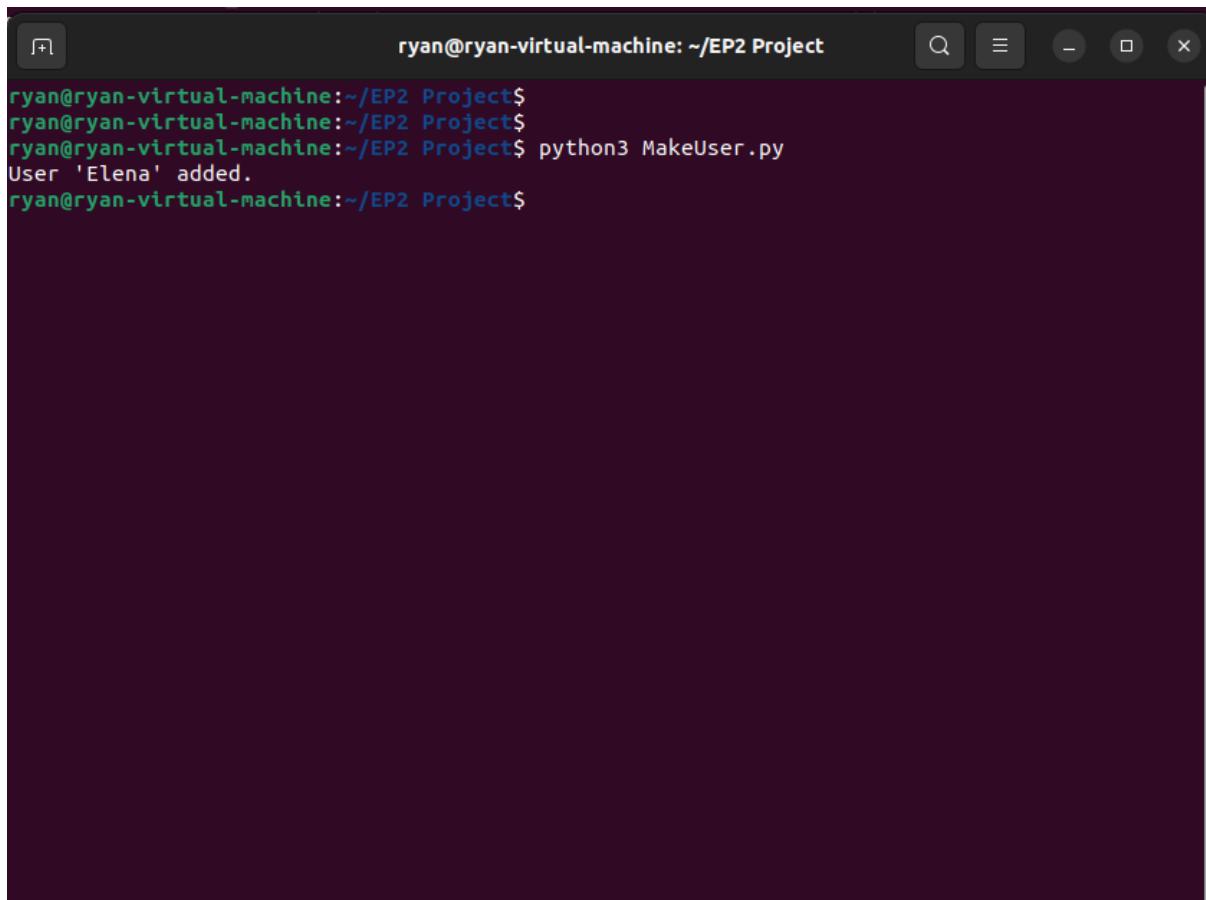
## Bastion Client Connection Refused (BCCR)



## Bastion Server Python Script to Add user (BSPS)

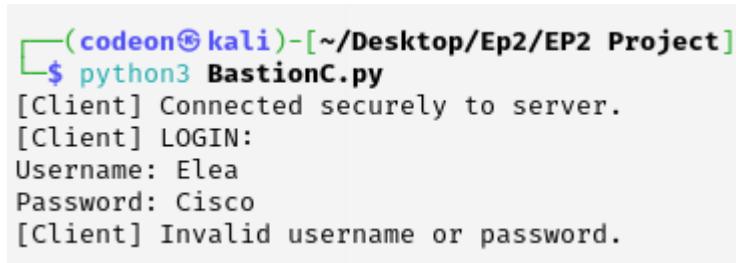


### Bastion Server Running Add User Script (BSRA)



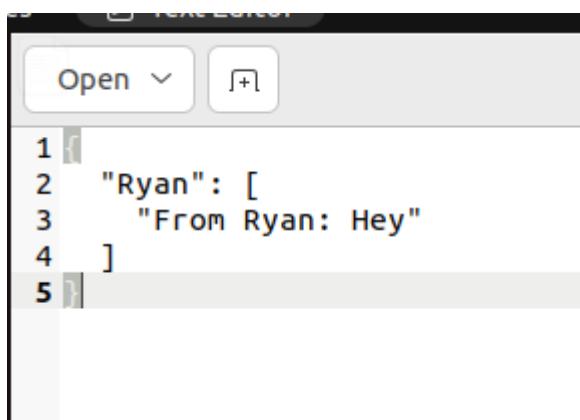
```
ryan@ryan-virtual-machine:~/EP2 Project$ python3 MakeUser.py
User 'Elena' added.
ryan@ryan-virtual-machine:~/EP2 Project$
```

### Bastion Client Wrong Username (BCWU)



```
(codeon㉿kali)-[~/Desktop/Ep2/EP2 Project]
$ python3 BastionC.py
[Client] Connected securely to server.
[Client] LOGIN:
Username: Elea
Password: Cisco
[Client] Invalid username or password.
```

### Bastion Server messages.db example (BSM)



```
1 {
2   "Ryan": [
3     "From Ryan: Hey"
4   ]
5 }
```

## Bastion Server Message Confirmed (BSMC)

[Ryan] INBOX  
[Ryan] SENDTO:Ryan:Hey  
[Server] Connection from ('192.168.242.128', 34502)  
[Server] Secure session established with Ryan  
[Ryan] SENDTO:Elena:Hey, how is it going?  
[ ]

## Bastion Client Brute Force Script (BCBF)

The screenshot shows a Kali Linux virtual machine interface. The terminal window displays a Python script named 'BruteForce Script.py' located at '/Desktop/Ep2/EP2 Project/BruteForce Script.py'. The script is a password cracker for a Windows 10 system. It imports socket, ssl, and time modules, defines variables for USERNAME ('ryan'), WORDLIST ('1234', 'password', 'admin', 'noroff', 'letmein', 'Noroff'), and sets up a socket connection to port 4443 on '192.168.242.131'. It handles password attempts by sending them to the server and checking the response for success. A sleep command is included to simulate real-world delays.

```
File Edit View VM Tabs Help Library Type here to search Kali Linux - /Desktop/Ep2/EP2 Project/BruteForce Script.py - Mousepad
File Edit Search View Document Help
import socket
import ssl
import time

USERNAME = "ryan"
WORDLIST = ["1234", "password", "admin", "noroff", "letmein", "Noroff"] # Replace with your test set

for pwd in WORDLIST:
    try:
        raw_sock = socket.create_connection(("192.168.242.131", 4443))
        context = ssl.create_default_context()
        context.check_hostname = False
        context.verify_mode = ssl.CERT_NONE

        with context.wrap_socket(raw_sock, server_hostname="localhost") as s:
            s.sendall(f"Tr1p3! {pwd}\n".encode())
            s.recv(1024) # LOGIN
            s.sendall(USERNAME.encode())
            s.sendall(pwd.encode())
            response = s.recv(1024).decode()
            print("[Server]", response)

            if "successful" in response.lower():
                print(f"[+] Password found: {pwd}")
                break
    except Exception as e:
        print(f"[-] Error: {e}")
    time.sleep(1) # slow down a bit to simulate real delay
```

## Bastion Client Brute Force Attack (BCBFA)

```
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certif
└──(codeon㉿kali)-[~/Desktop/Ep2/EP2 Project]
$ python3 BruteForce\ Script.py
[*] Trying password: 1234
[Server] Invalid username or password.
[*] Trying password: password
[Server] Invalid username or password.
[*] Trying password: admin
[Server] Too many failed attempts. User locked for 30 seconds.
[*] Trying password: noroff
[Server] Too many failed attempts. Try again later.
[*] Trying password: letmein
[Server] Too many failed attempts. Try again later.
[*] Trying password: Noroff
[Server] Too many failed attempts. Try again later.
```

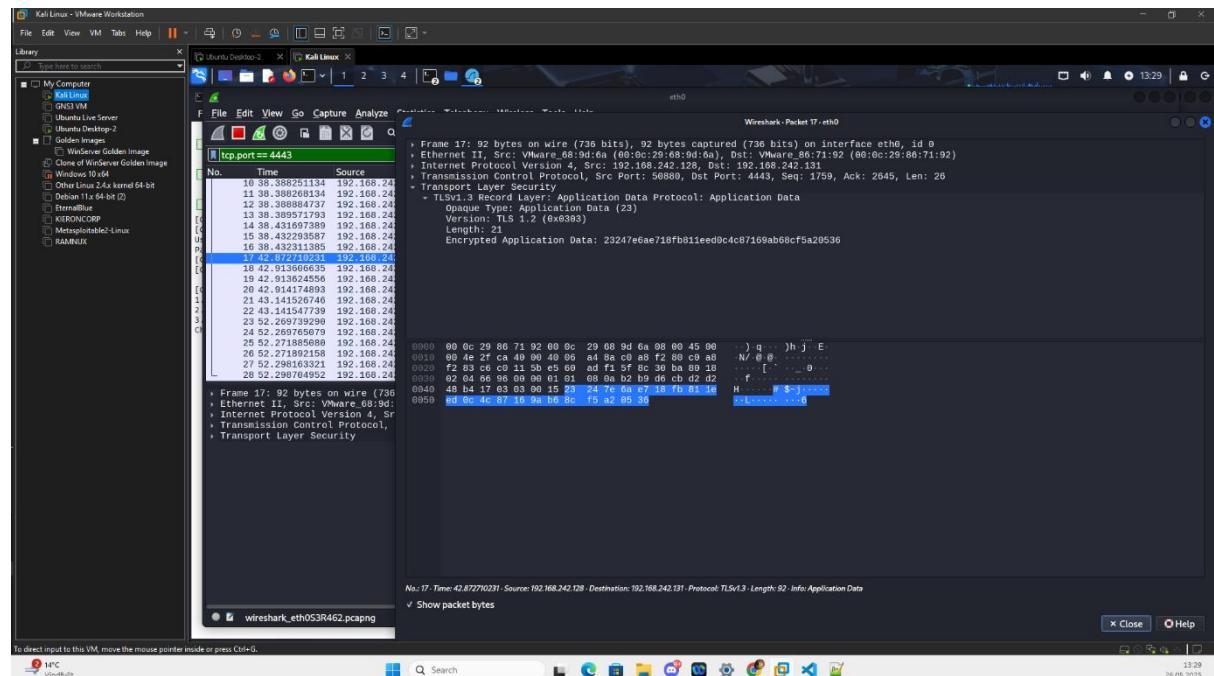
## Bastion Server Down after brute force (BSBD)

```
ryan@ryan-virtual-machine:~/EP2 Project$ python3 BastionServerFinalVersion.py
[Server] Listening on port 4443...
[Server] Loaded locked users: []
[Server] Connection from ('192.168.242.128', 54284)
[Server] Secure session established with Ryan
[Server] Connection from ('192.168.242.128', 59662)
[Server] Secure session established with Elena
Traceback (most recent call last):
  File "/home/ryan/EP2 Project/BastionServerFinalVersion.py", line 294, in <module>
    start_server()
  File "/home/ryan/EP2 Project/BastionServerFinalVersion.py", line 290, in start_server
    secure_sock = context.wrap_socket(client_sock, server_side=True)
  File "/usr/lib/python3.10/ssl.py", line 513, in wrap_socket
    return self.sslsocket_class._create(
  File "/usr/lib/python3.10/ssl.py", line 1100, in _create
    self.do_handshake()
  File "/usr/lib/python3.10/ssl.py", line 1371, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLError: [SSL: TLSV1_ALERT_UNKNOWN_CA] tlsv1 alert unknown ca (_ssl.c:1007)
ryan@ryan-virtual-machine:~/EP2 Project$
```

## Bastion Server After Second Brute Force, survived (BSBD2)

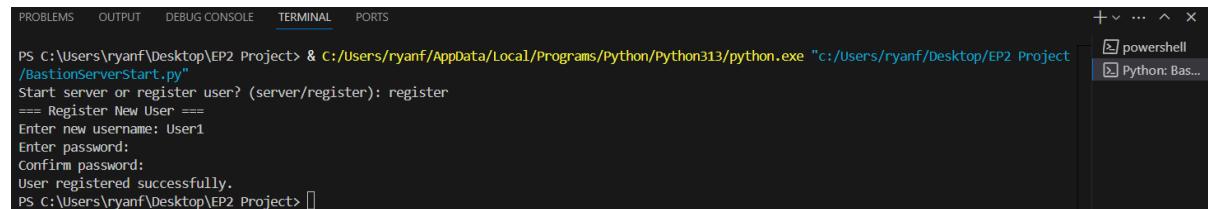
```
ryan@ryan-virtual-machine:~/EP2 Project$ python3 BastionServerFinalVersion.py
[Server] Listening on port 4443...
[Server] Loaded locked users: []
[Server] TLS handshake failed from ('192.168.242.128', 43748): [SSL: TLSV1_ALERT_UNKNOWN_CA] tlsv1 alert unknown ca (_ssl.c:1007)
```

## Bastion Client Wireshark Sniffing (BCWS)

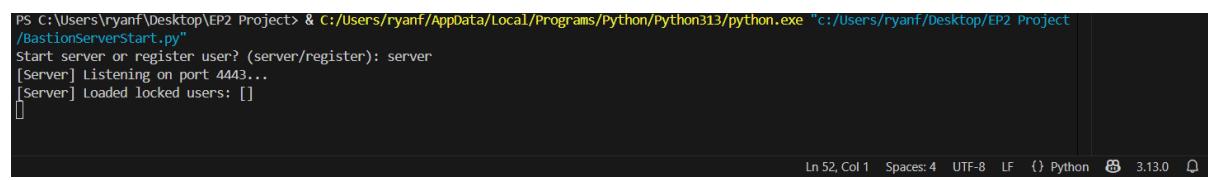


## 8.3 Bastion – Secure Email Fortress

### Bastion – Secure Mail Fortress (BSMF)

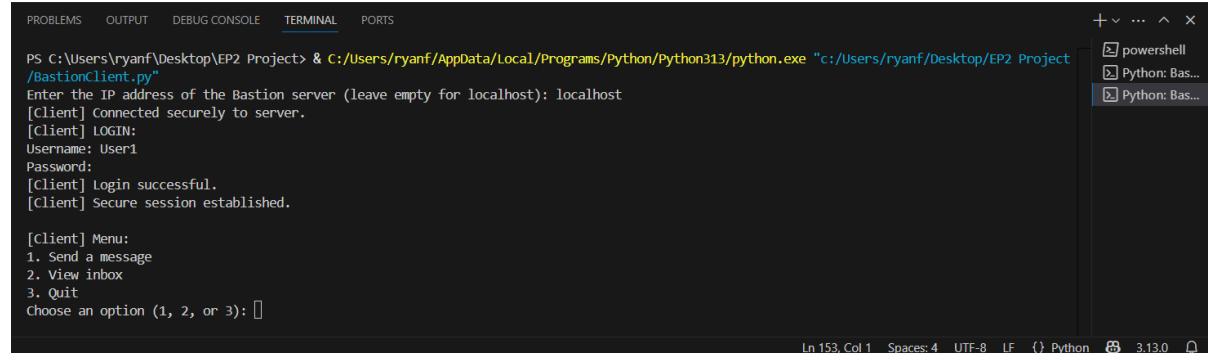


```
PS C:\Users\ryanf\Desktop\EP2 Project> & C:/Users/ryanf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/ryanf/Desktop/EP2 Project/BastionServerStart.py"
Start server or register user? (server/register): register
== Register New User ==
Enter new username: User1
Enter password:
Confirm password:
User registered successfully.
PS C:\Users\ryanf\Desktop\EP2 Project>
```



```
PS C:\Users\ryanf\Desktop\EP2 Project> & C:/Users/ryanf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/ryanf/Desktop/EP2 Project/BastionServerStart.py"
Start server or register user? (server/register): server
[Server] Listening on port 4443...
[Server] Loaded locked users: []

```



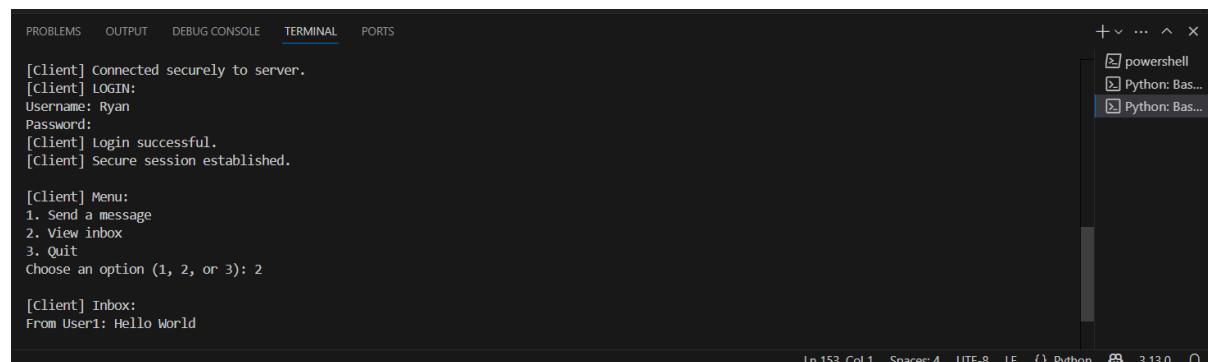
```
PS C:\Users\ryanf\Desktop\EP2 Project> & C:/Users/ryanf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/ryanf/Desktop/EP2 Project/BastionClient.py"
Enter the IP address of the Bastion server (leave empty for localhost): localhost
[Client] Connected securely to server.
[Client] LOGIN:
Username: User1
Password:
[Client] Login successful.
[Client] Secure session established.

[Client] Menu:
1. Send a message
2. View inbox
3. Quit
Choose an option (1, 2, or 3):
```



```
[Client] Secure session established.

[Client] Menu:
1. Send a message
2. View inbox
3. Quit
Choose an option (1, 2, or 3): 1
Enter recipient username: Ryan
Enter your message: Hello World
[Client] Message sent to Ryan
```



```
[client] Connected securely to server.
[Client] LOGIN:
Username: Ryan
Password:
[Client] Login successful.
[Client] Secure session established.

[Client] Menu:
1. Send a message
2. View inbox
3. Quit
Choose an option (1, 2, or 3): 2

[Client] Inbox:
From User1: Hello World
```

