CS4203 Practical 2: The Discussion Forum

Designing and creating a secure discussion platform

190018469

Computer Science
University of St Andrews
November 2023

1 Overview

The specification outlined the creation of a secure discussion system that maximises the confidentiality and privacy of the participants. The final implementation offers a secure platform for users to chat and discuss, with data stored anonymously with zero-knowledge on a database. The report outlines the application's features, giving details on the architecture used. Several key topics were identified, covering areas of data safety, key management and user anonymity. Furthermore, a threat model on these key concerns is presented, showing how the system addresses these and suggesting steps that could be taken in future to strengthen the system further.

Usage Instructions

The server requires the initialisation of a database, which can be created through use of the Server/create_full_db.sql file. The server can then be started using the command java -cp ".:Libraries/*:Shared:Server" Server/SecureChatServer.java <caCrtFile> <crtFile> <keyFile> <dbURL> <dbUser> <dbPassword> <port number [optional]>. After the server has started, the client can be started with the command java -cp ".:Libraries/*:Shared:Client" Client/SecureChatClient.java <host address> <port number> <caCrtFile> <crtFile> <keyFile> <chat key folder>. Sample keys and certificates are provided with the client and server, including a set for a pre-initialised chat with the number 1.

2 Design

2.1 Platform

On the discussion platform, users should be able to create and join groups securely and anonymously. They can connect to the server using their key and a signed certificate. Once connected, the user can input a group ID to connect to a specific group room if they have the group's keys stored locally. Alternatively, they can create their own group by specifying a name and sharing the generated keys with others externally. While users need to identify themselves to the server, messages they send in a group chat are not associated with a specific user.

2.2 Architecture

2.2.1 Data Safety

The use of Java's Secured Socket Layer (SSL) with TLS 1.3 allows for secure communication between the server and client with encryption, authentication and data integrity [1], [2]. When the server is started, it loads the server's private key and certificate into a key store. Furthermore, a trust manager is initialised with another key store holding the certificate authority's certificate. Together, these can be used to initialise an SSL context, from which a server socket can be created. The client can similarly create an SSL context with their own credentials, which can then be used to connect to the server. To allow groups to be created and read over time, the server must store its information and messages in a database. By encrypting all group names and messages on the database, this data can be kept safe at rest. The only information kept unencrypted is each group's ID. However, this is acceptable as knowing the ID does not reveal anything about the contents of the group.

2.2.2 Key Management

To provide secure access to groups, a public key is stored in addition to encrypted data for the group's name and messages. This allows users to log in to groups through challenge-response, where the server verifies that the user holds the public key's corresponding private key. This private key can be distributed by previous members of the group to give access to new members. By using this system, group data can be stored safely with encryption on the server while only allowing the users with the right keys to read it, making the storage zero-knowledge [3]. In challenge response, when a user attempt to connect to a chat, the server provides them with a challenge string. The user's client then encrypts the challenge by using the chat's private key and returns it. If the server can then decrypt the response correctly by using the chat's public key to obtain the original message, the user is allowed in the chat.

The algorithm used for the keypair is RSA. Using the keypair to encrypt messages was not optimal due to overhead and limited size [4]. Instead, a symmetric AES key is used to encrypt and decrypt each message and the chat's name. This key is not stored on the server and should be managed similarly to the RSA private key. Each chat has its own set of three keys, which are generated by the client when creating a chat. After generating the keys, the specified chat name is encrypted and sent alongside the public key to the server. If the server then manages to successfully create the new chat in the database, the chat's ID is returned to the client and the keys stored with the chat ID on the client's disc.

2.2.3 Anonymity

To provide users with secure, anonymous communication, the platform is built on a zero-knowledge architecture and does not use usernames. While chats are stored on the server, they are encrypted with keys held by users of the platform. This ensures that the server cannot read any of the messages sent. Avoiding the use of usernames also aids in keeping the platform anonymous as any single message cannot be associated with a user. While the chat and stored messages are anonymous, to access the server in the first place the user must present credentials as part of TLS 1.3. While this does restrict use of the system to trusted users as long as credentials are not stolen, it does reveal information about the individual user or their group to the server.

3 Threat Model

3.1 Data Safety

3.1.1 Inherent Challenges

Given that the server is intended as a secure chat platform, data stored must be stored securely and kept to only what is required, both during transit and at rest.

3.1.2 Addressing via Implementation

Traffic between the server and client is secured using TLS to create a secure socket layer [2]. To protect data at rest, everything except each chat's ID and public key is encrypted [5]. In order to protect the database, direct access to it is limited to the server and the operations that can be carried out are limited by the application layer of the server. To prevent SQL injections, the application layer exclusively uses prepared and callable statements when making queries.

3.1.3 Remaining Threats

A potential countermeasure to data loss and any resulting service loss could be the use of dead storage backups and replication [6]. This would involve writing regular backups with limited read access and keeping a backup second server in case the first fails. The main server could additionally be kept behind a DMZ to provide a second layer of security, with the second server kept public facing. To tackle DDoS attacks, the server could use rate limitation to ignore excessive traffic.

3.2 Key Management

3.2.1 Inherent Challenges

Another obvious threat in any user platform is the loss of a user's credentials. In this case, a user may lose their login files as well as the private and symmetric keys of chats. This could give an attacker unintended access to the platform or chats they were not invited to. Alternatively, if the files were deleted, the user may simply lose access to their own account and chats.

3.2.2 Addressing via Implementation

While losing one key is not enough to compromise access through normal channels, the symmetric key is arguably the most important key to keep secure as it is the key used to read messages. An alternative design to combat this is to use hybrid encryption [5]. This would involve generating a new symmetric key for each message, which is stored on the server alongside the message and encrypted using the private key. Using this, the client would no longer have to store the symmetric key and the loss of one symmetric key would only endanger one message. However, this simply moves all the pressure onto the private key, presenting the same key management challenge. In turn, the chosen design comes with less overhead and storage requirements with only one locally stored symmetric key per chat. Furthermore, without users being given group management permissions, a compromised set of keys will at most reveal chat information, with no way for the attacker to delete the group itself using just the keys.

3.2.3 Remaining Threats

Under any system, losing credentials provides a serious threat to their platform. As such, putting an emphasis on keeping credentials safe, identifying when they are lost and being able to change their server-side validity can help to tackle this issue. In a user-driven discussion platform, each user is responsible for managing their own credentials, making it impossible to ensure their safekeeping. Care should also be taken regarding the use of certificates. If bad actors' certificate requests are signed or certificate authorities are attacked, attackers could acquire or sign certificates for malicious use [7].

3.3 User Anonymity

3.3.1 Inherent Challenges

In discussion platforms with usernames, it is trivial to associate each message with a specific user. While using a secure system where each discussion chat is only open to chosen users hides what each user says at a surface level, an attacker in a breach or malicious system admin could penetrate this layer and study messages as before. One way to limit the impact of this may be to use an 'active chat', where messages are not stored on the server but only shared with other connected clients when sent, but this presents poor usability for a discussion forum. A less severe alternative might be to purge messages after a certain time.

3.3.2 Addressing via Implementation

Instead, the design chosen does not store usernames at all. This makes it is much harder to associate a given message with a single user. While the user is identified to the server when making a connection to the server, the fact that messages are not associated with a single user means that an attacker would only be able to associate messages sent while they have breached the server and are actively logging its connections. Using zero-knowledge storage as described previously ensures that anonymity is preserved even if data is shared, such as in cases where data must be published for legal reasons [8].

3.3.3 Remaining Threats

An alternative way for attackers to build profiles on users would be to monitor traffic to and from the server. By looking at which traffic is sent where and when, an attacker could potentially identify users by their IP address and which users are part of a given chat [9]. To combat this, users could use, or even be forced to use, a VPN to mask their IP [10] and the server could disguise meaningful traffic by also sending random, encrypted traffic [11].

While anonymity protects the identity of legitimate users, it also protects malicious actors. There is no way for users to know whether someone reading or messaging in a chat is intended to have access to it. This problem relies directly on how the threats regarding data safety and key management are dealt with to reduce the chance of a bad actor getting unintended access to chats. Knowing this, users who are conscious about security are likely to exercises caution in the information they share on the platform and how they share the keys to access a specific chat. Besides accessing chats without permission, attackers may spam chats or send harmful messages, which could decrease the experience of other users [12].

4 Testing

4.1 Functionality Testing

A number of tests to ensure that each feature of the platform works are given in Table 1. These can be followed manually to ensure that each part of the system works. Screenshots of testing can be found in the appendix.

Test Name	Purpose
Connection (Valid)	Ability to successfully connect given the correct credentials.
Connection (Invalid)	Inability to connect with invalid certificates or private keys.
Chat Creation	Ability to create a chat and enter it.
Chatting (New Chat)	Ability to send a message in a newly created chat.
Challenge-Response (Valid)	Ability to pass challenge-response with valid credentials.
Challenge-Response (Invalid)	Inability to pass challenge-response with invalid credentials or chat number.
Chatting (Existing Chat)	Ability to send a message in an existing chat (after challenge-response).
Read Chat History	Ability to read messages sent into a chat before joining it.
Live Chatting	Ability to read messages sent by others while connected to a chat.
Exiting Chat	Ability to exit a chat and return to the chat selection.
Chat Reselection	Ability to choose another chat in case of failed joining or aborted creation.

Table 1: Functionality Testing

4.2 Packet Capture

Wireshark was used to check what an attacker might be able to detect through use of packet capture. Two models of the client and server were tested, one pair insecure and the other secure. It was found that the insecure client's data was always visible over the network and that the use of TLS in the secure pair prevented messages from being read. Furthermore, the secure server did not accept any data from an insecure client and the secure client did not send any data over an insecure connection.

5 Conclusion

The final implementation offers a secure, anonymous chat platform for users with zero-knowledge storage of chats on the server. The design of the platform and how it addresses key security concerns is discussed in the report. Functionality and security is demonstrated in testing and justified, with remaining issues and potential countermeasures outlined.

References

[1] Baeldung, *Introduction to ssl in java*. [Online]. Available: https://www.baeldung.com/java-ssl (visited on 09/11/2023).

[2] C. Cremers, M. Horvat, J. Hoyland, S. Scott and T. van der Merwe, 'A comprehensive symbolic analysis of tls 1.3,' in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1773–1788. DOI: 10.1145/3133956.3134063. [Online]. Available: https://doi.org/10.1145/3133956.3134063.

[3] J. O'Leary, *Technology preview: Signal private group system*. [Online]. Available: https://signal.org/blog/signal-private-group-system/ (visited on 09/11/2023).

[4] A. Al Hasib and A. A. M. M. Haque, 'A comparative study of the performance and security issues of aes and rsa cryptography,' in *2008 third international conference on convergence and hybrid information technology*, IEEE, vol. 2, 2008, pp. 505–510.

[5] L. Yu, Z. Wang and W. Wang, 'The application of hybrid encryption algorithm in software security,' in *2012 Fourth International Conference on Computational Intelligence and Communication Networks*, 2012, pp. 762–765. DOI: 10.1109/CICN.2012.195.

[6] H. Zou and F. Jahanian, 'A real-time primary-backup replication service,' *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 533–548, 1999. DOI: 10.1109/71.774905.

[7] N. Leavitt, 'Internet security under attack: The undermining of digital certificates,' *Computer*, vol. 44, no. 12, pp. 17–20, 2011.

[8] S. Motahari, S. G. Ziavras and Q. Jones, 'Online anonymity protection in computer-mediated communication,' *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 3, pp. 570–580, 2010. DOI: 10.1109/TIFS.2010.2051261.

[9] J. Ren and J. Wu, 'Survey on anonymous communications in computer networks,' *Computer Communications*, vol. 33, no. 4, pp. 420–431, 2010. DOI: https://doi.org/10.1016/j.comcom.2009.11.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366409002989.

[10] A. Popovych, *How to develop an anonymous social media app*. [Online]. Available: https://clockwise.software/blog/how-to-build-anonymous-social-media-app/ (visited on 15/11/2023).

[11] C. Gutu and F. Ding, 'Secretroom: An anonymous chat client,' 2016.

[12] C. Negiar and D. Amin, 'Predicting user quality in anonymous chat networks,'

6 Appendix

6.1 Testing Screenshots

The following screenshots show the results of each test as outlined in the testing section.

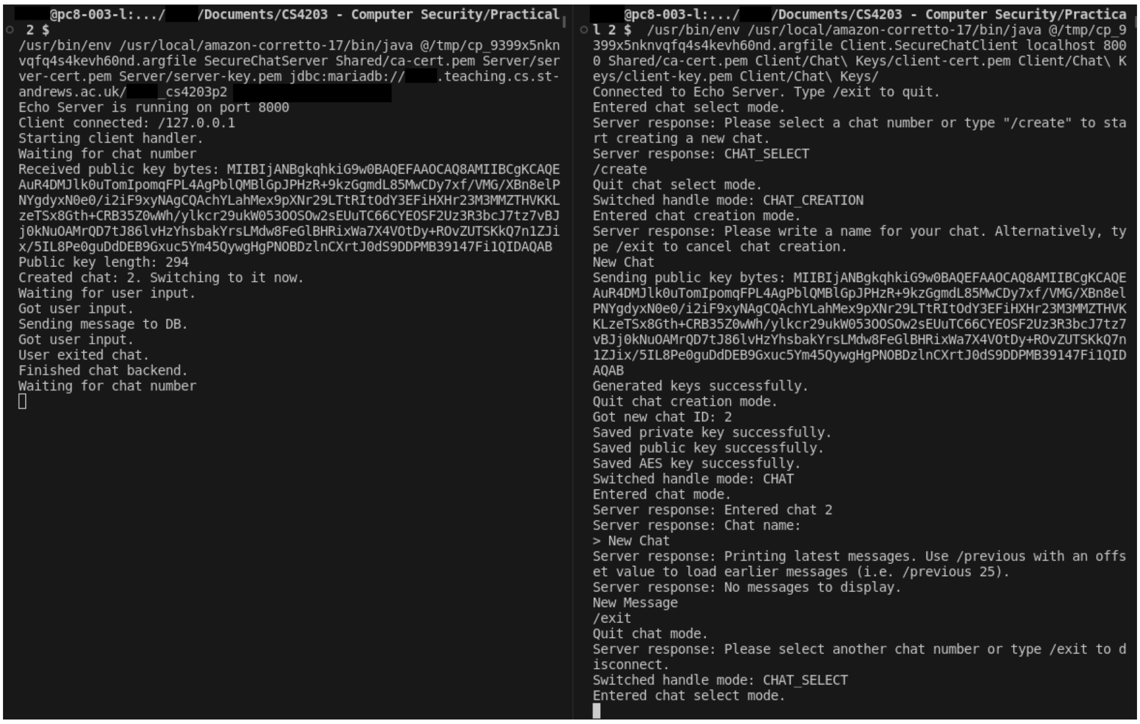


Figure 1: Tests 1, 3, 4 and 10

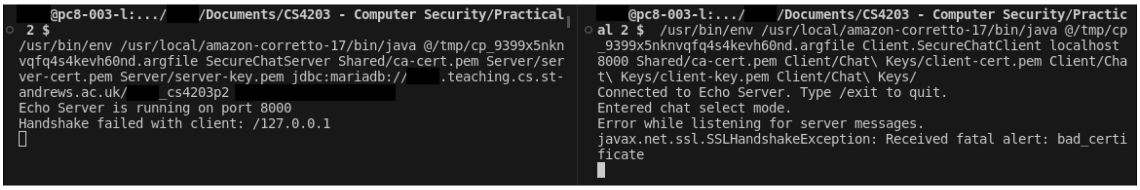
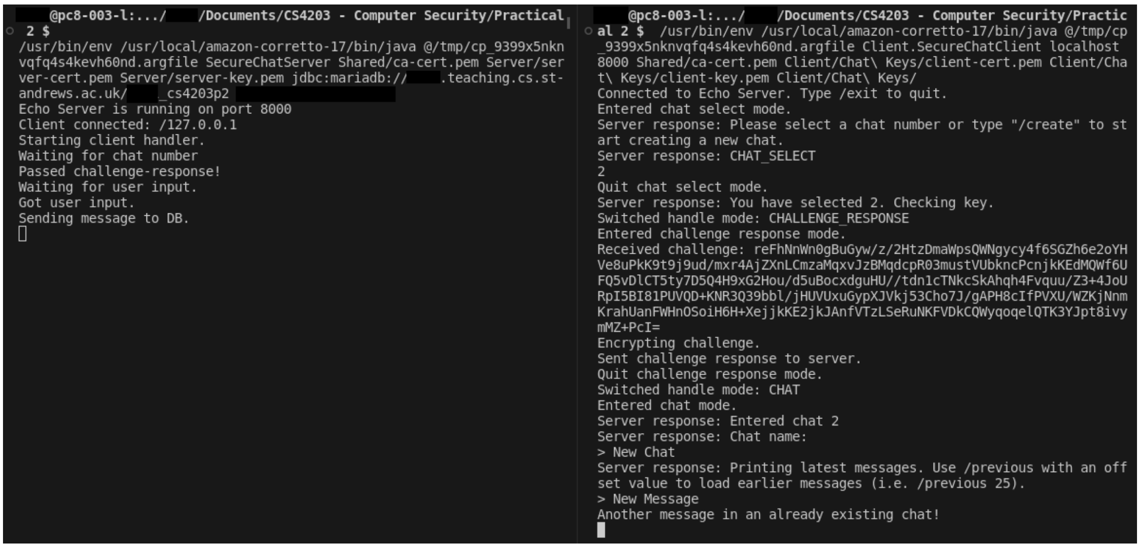


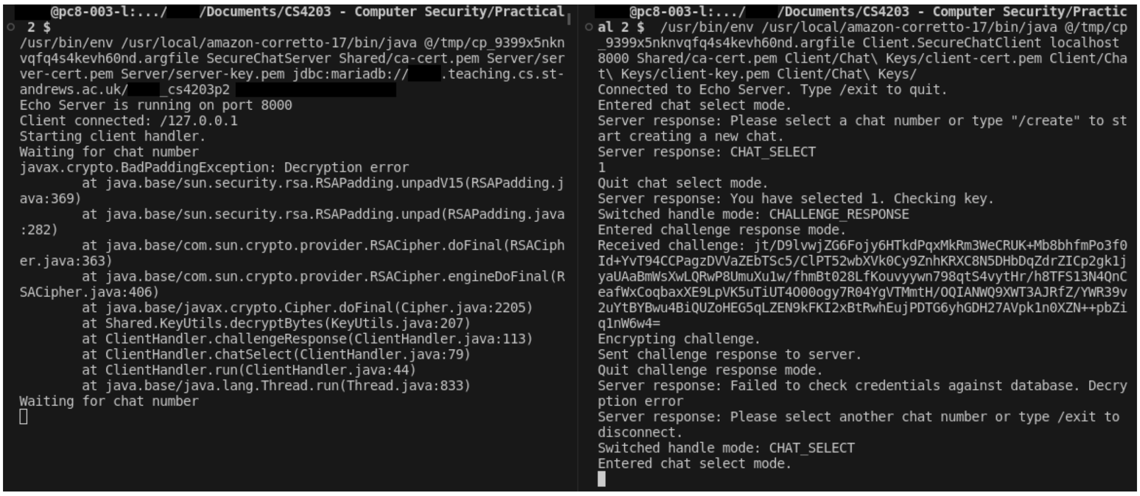
Figure 2: Test 2



```
[redacted]@pc8-003-l:.../[redacted]/Documents/CS4203 - Computer Security/Practical
 2 $
/usr/bin/env /usr/local/amazon-corretto-17/bin/java @/tmp/cp_9399x5nkn
vqfq4s4kevh60nd.argfile SecureChatServer Shared/ca-cert.pem Server/ser
ver-cert.pem Server/server-key.pem jdbc:mariadb://[redacted].teaching.cs.st-
andrews.ac.uk/[redacted]_cs4203p2 [redacted]
Echo Server is running on port 8000
Client connected: /127.0.0.1
Starting client handler.
Waiting for chat number
Passed challenge-response!
Waiting for user input.
Got user input.
Sending message to DB.
```

```
[redacted]@pc8-003-l:.../[redacted]/Documents/CS4203 - Computer Security/Practic
al 2 $  /usr/bin/env /usr/local/amazon-corretto-17/bin/java @/tmp/cp
_9399x5nknvqfq4s4kevh60nd.argfile Client.SecureChatClient localhost
8000 Shared/ca-cert.pem Client/Chat\ Keys/client-cert.pem Client/Cha
t\ Keys/client-key.pem Client/Chat\ Keys/
Connected to Echo Server. Type /exit to quit.
Entered chat select mode.
Server response: Please select a chat number or type "/create" to st
art creating a new chat.
Server response: CHAT_SELECT
2
Quit chat select mode.
Server response: You have selected 2. Checking key.
Switched handle mode: CHALLENGE_RESPONSE
Entered challenge response mode.
Received challenge: reFhNnWn0gBuGyw/z/2HtzDmaWpsQWNgycy4f6SGZh6e2oYH
Ve8uPkK9t9j9ud/mxr4AjZXnLCmzaMqxvJzBMqdcpR03mustVUbkncPcnjkKEdMQWf6U
FQ5vDlCT5ty7D5Q4H9xG2Hou/d5uBocxdguHU//tdn1cTNkcSkAhqh4Fvquu/Z3+4JoU
RpI5BI81PUVQD+KNR3Q39bbl/jHUVUxuGypXJVkj53Cho7J/gAPH8cIfPVXU/WZKjNnm
KrahUanFWHnOSoiH6H+XejjkKE2jkJAnfVTzLSeRuNKFVDkCQWyqoqelQTK3YJpt8ivy
mMZ+PcI=
Encrypting challenge.
Sent challenge response to server.
Quit challenge response mode.
Switched handle mode: CHAT
Entered chat mode.
Server response: Entered chat 2
Server response: Chat name:
> New Chat
Server response: Printing latest messages. Use /previous with an off
set value to load earlier messages (i.e. /previous 25).
> New Message
Another message in an already existing chat!
```

Figure 3: Tests 5, 7 and 8



```
[redacted]@pc8-003-l:.../[redacted]/Documents/CS4203 - Computer Security/Practical
 2 $
/usr/bin/env /usr/local/amazon-corretto-17/bin/java @/tmp/cp_9399x5nkn
vqfq4s4kevh60nd.argfile SecureChatServer Shared/ca-cert.pem Server/ser
ver-cert.pem Server/server-key.pem jdbc:mariadb://[redacted].teaching.cs.st-
andrews.ac.uk/[redacted]_cs4203p2 [redacted]
Echo Server is running on port 8000
Client connected: /127.0.0.1
Starting client handler.
Waiting for chat number
javax.crypto.BadPaddingException: Decryption error
        at java.base/sun.security.rsa.RSAPadding.unpadV15(RSAPadding.j
ava:369)
        at java.base/sun.security.rsa.RSAPadding.unpad(RSAPadding.java
:282)
        at java.base/com.sun.crypto.provider.RSACipher.doFinal(RSACiph
er.java:363)
        at java.base/com.sun.crypto.provider.RSACipher.engineDoFinal(R
SACipher.java:406)
        at java.base/javax.crypto.Cipher.doFinal(Cipher.java:2205)
        at Shared.KeyUtils.decryptBytes(KeyUtils.java:207)
        at ClientHandler.challengeResponse(ClientHandler.java:113)
        at ClientHandler.chatSelect(ClientHandler.java:79)
        at ClientHandler.run(ClientHandler.java:44)
        at java.base/java.lang.Thread.run(Thread.java:833)
Waiting for chat number
```

```
[redacted]@pc8-003-l:.../[redacted]/Documents/CS4203 - Computer Security/Practic
al 2 $  /usr/bin/env /usr/local/amazon-corretto-17/bin/java @/tmp/cp
_9399x5nknvqfq4s4kevh60nd.argfile Client.SecureChatClient localhost
8000 Shared/ca-cert.pem Client/Chat\ Keys/client-cert.pem Client/Cha
t\ Keys/client-key.pem Client/Chat\ Keys/
Connected to Echo Server. Type /exit to quit.
Entered chat select mode.
Server response: Please select a chat number or type "/create" to st
art creating a new chat.
Server response: CHAT_SELECT
1
Quit chat select mode.
Server response: You have selected 1. Checking key.
Switched handle mode: CHALLENGE_RESPONSE
Entered challenge response mode.
Received challenge: jt/D9lvwjZG6Fojy6HTkdPqxMkRm3WeCRUK+Mb8bhfmPo3f0
Id+YvT94CCPagzDVVaZEbTSc5/ClPT52wbXVk0Cy9ZnhKRXC8N5DHbDqZdrZICp2gk1j
yaUAaBmWsXwLQRwP8UmuXu1w/fhmBt028LfKouvyywn798qtS4vytHr/h8TFS13N4QnC
eafWxCoqbaxXE9LpVK5uTiUT4O00ogy7R04YgVTMmtH/OQIANWQ9XWT3AJRfZ/YWR39v
2uYtBYBwu4BiQUZoHEG5qLZEN9kFKI2xBtRwhEujPDTG6yhGDH27AVpk1n0XZN++pbZi
q1nW6w4=
Encrypting challenge.
Sent challenge response to server.
Quit challenge response mode.
Server response: Failed to check credentials against database. Decry
ption error
Server response: Please select another chat number or type /exit to
disconnect.
Switched handle mode: CHAT_SELECT
Entered chat select mode.
```

Figure 4: Test 6

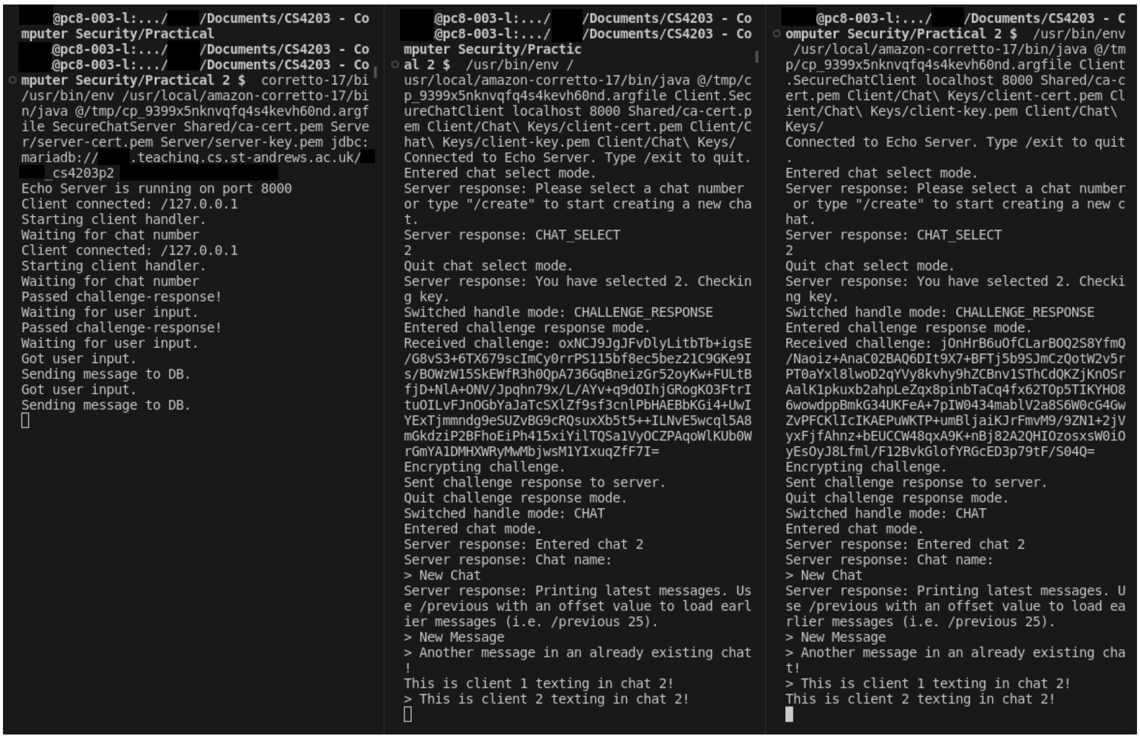


Figure 5: Test 9

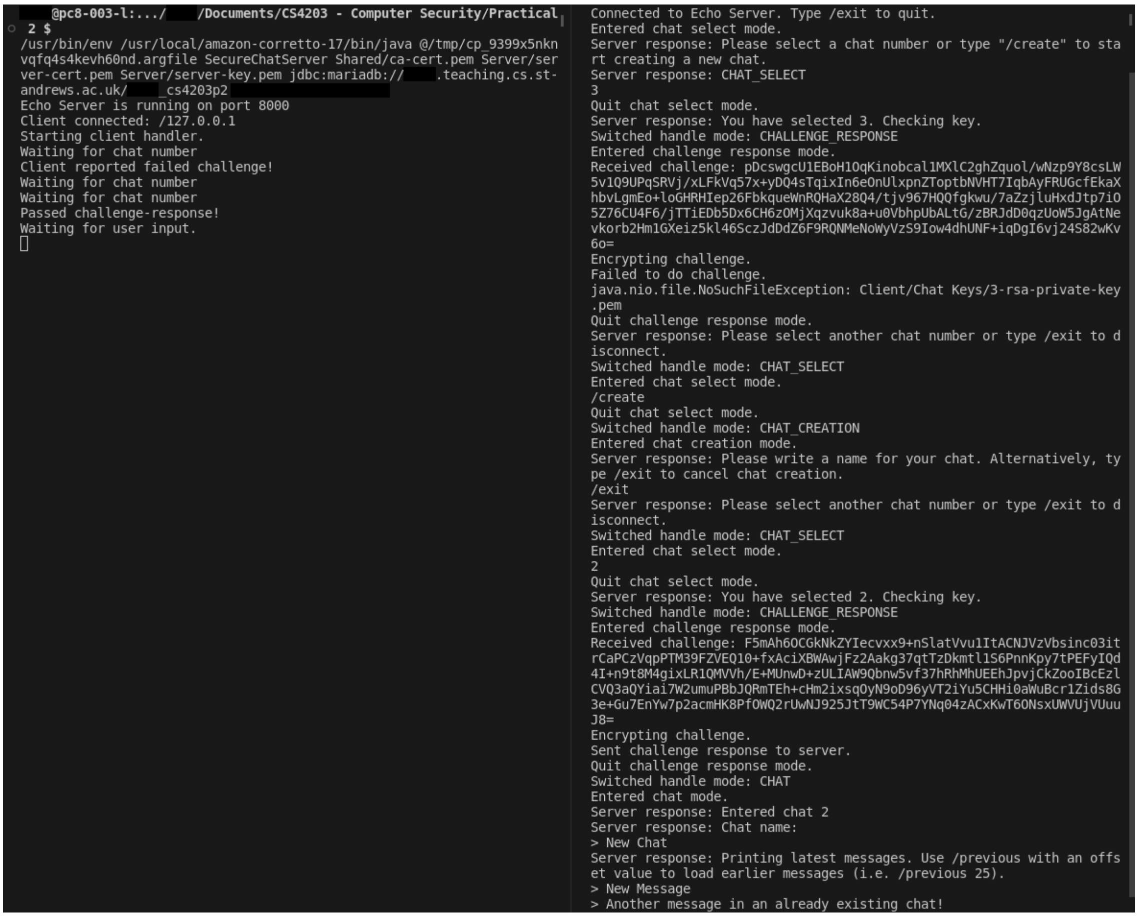


Figure 6: Test 11

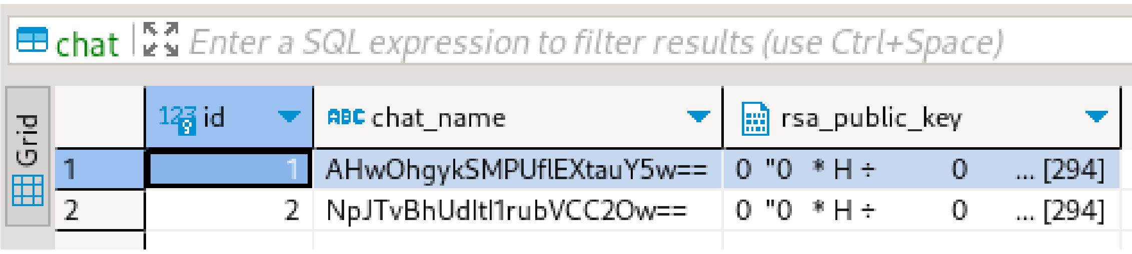
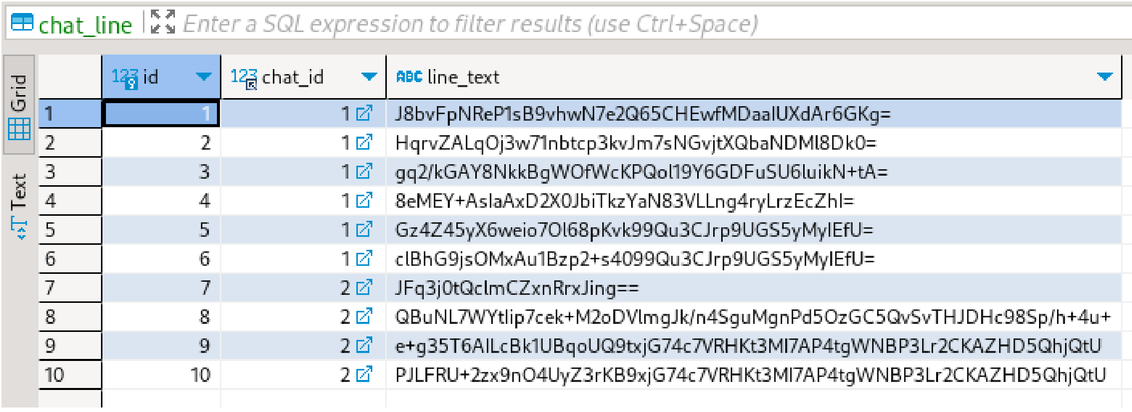


Figure 7: State of the Database Chat Table after Testing



chat_line | Enter a SQL expression to filter results (use Ctrl+Space)

	id	chat_id	line_text
1	1	1	J8bvFpNRePlsB9vhwN7e2Q65CHEwfMDaalUXdAr6GKg=
2	2	1	HqrvZALqOj3w71nbtcp3kvJm7sNGvjtXQbaNDMl8Dk0=
3	3	1	gq2/kGAY8NkkBgWOfWcKPQol19Y6GDFuSU6luikN+tA=
4	4	1	8eMEY+AslaAxD2X0JbiTkzYaN83VLLng4ryLrzEcZhl=
5	5	1	Gz4Z45yX6weio7Ol68pKvk99Qu3CJrp9UGS5yMylEfU=
6	6	1	clBhG9jsOMxAu1Bzp2+s4099Qu3CJrp9UGS5yMylEfU=
7	7	2	JFq3j0tQclmCZxnRrxJing==
8	8	2	QBuNL7WYtlip7cek+M2oDVlmgJk/n4SguMgnPd5OzGC5QvSvTHJDHc98Sp/h+4u+
9	9	2	e+g35T6AlLcBk1UBqoUQ9txjG74c7VRHKt3MI7AP4tgWNBP3Lr2CKAZHD5QhjQtU
10	10	2	PJLFRU+2zx9nO4UyZ3rKB9xjG74c7VRHKt3MI7AP4tgWNBP3Lr2CKAZHD5QhjQtU

Figure 8: State of the Database Chat Line Table after Testing

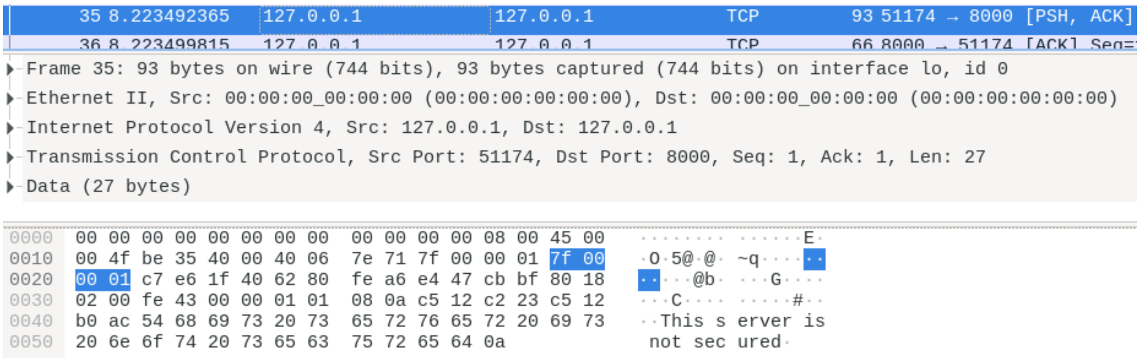


Figure 9: Packet capture reading traffic across an unencrypted connection

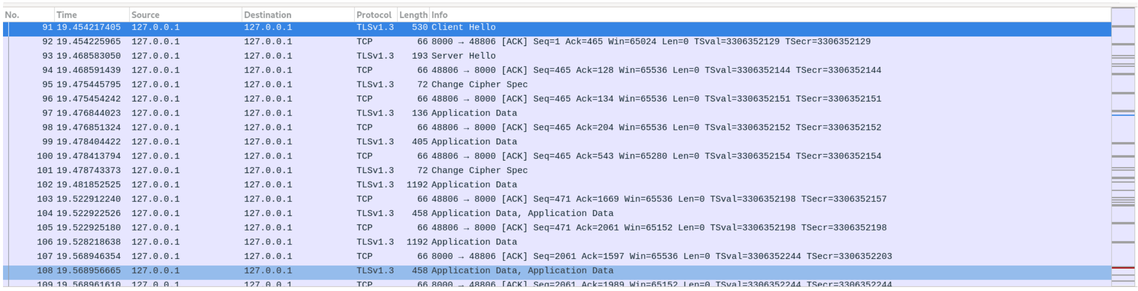


Figure 10: TLS 1.3 being used to secure traffic