Secure P2P messaging app

Data Privacy and Security

Supervised by:
Prof. Dr. Bernardo Ferreira
Department of Informatics

Faculty of Sciences of the University of Lisbon

Luís Viana
fc62516

Guilherme Santos
fc62533

December 10, 2023

Contents

1 Introduction

In this project, we developed a secure P2P messaging app that allows users to send messages between any two users with end-to-end encryption guarantees. The app will provide standard reliability guarantees, including message delivery and ordering. Additionally, we will implement a client interface for testing and visualization purposes, which will display the list of conversations of a user and the contents of a specified conversation. For the implementation of the messaging desktop app, we used the latest Java version, Java 21, and the software platform JavaFX. We have chosen to implement two extra functionalities: long-term storage of messages and group conversations. For long-term storage of messages, we replicate a file containing a copy of the messages to various cloud providers. This file is encrypted using an Advanced Encryption Standard (AES) key, ensuring the security and integrity of the data. The AES key is divided into shares, each of which is stored with a different cloud provider. The cloud providers used were Dropbox, Google Drive, and Github. Additionally, we streamlined the integration of cloud storage APIs using the build automation tool Maven. This approach guarantees the availability of the messages, as they can be recovered even if the user's device or a single provider is compromised. For the group conversation functionality, we used a ciphertext-policy attribute-based encryption (CP-ABE) scheme. This scheme will provide secure access control, ensuring that only group members can view the messages.

1.1 Assumptions

- The server is secure, giving guarantees of secure storage of the user's IP and associated attributes.
- The server is reliable and trustworthy, ensuring that responses such as usernames with a specific attribute or user IPs can be confidently considered to have accurate and genuine values.
- The server maintains constant availability (a solution with several replicas of the server could be implemented with Docker or cloud services, but since it falls outside the scope of the topics covered in this class, we decided not to implement it).
- The user truststore already has the certificates of other users that he has contact with; it is not possible in this implementation to add a new user to have contact with.
- The server already has the attributes associated with each user (these are statically added at the beginning of the server execution), so it is not possible in these implementations for a user to join a new group chat.

2 Basic Implementation

In the primary phase of our project implementation, we leverage Transport Layer Security version 1.3 (TLSv1.3) to encrypt messages exchanged between peers, utilizing the built-in SSLSocket Java class. In the context of secure communication between peers, each peer possesses a certificate, a keystore, and a truststore. The keystore is a file that stores the pair of keys associated with the peer, while the truststore is a file that stores the certificates of trusted peers. For a peer to establish communication with another, the certificate of the other peer must be present in its truststore. TLS provides confidentiality by encrypting the data transmitted between the client and the server and between users; integrity by using a Message Authentication Code (MAC) for each message transmitted; and authenticity by using certificates to verify the identity of the communicating parties (2, 4).

The project's structure is predicated on the existence of a secure, reliable, and always available server that represents the initial point of contact in the user application. Upon the commencement of its execution, the user shares its IP address and port with the server. The server responds with the secret key and a mapping of groupchat and its corresponding public key. Subsequently, there is an exchange of data between the user and the server, which results in the user knowing all the IP addresses of the users he has contact with, the access policies associated with each group chat, and all of the usernames associated with each group chat. Afterwards, the user attempts to connect with all of its contacts, starting a thread and constantly trying to read any message received if the connection is successful. It also starts a thread that will be continuously waiting for any attempt to connect to the user from other users who started their app, for example.

Upon contact selection, there is the retrieval of the shares from the cloud providers, their combination, the decryption of the file, and the loading of the messages into the interface.

The messages are sent and received through the connections made initially or any new connections made mid-application execution.

Upon the action of closing the application, we save all the messages sent and received during its execution that were stored in a hash map in their corresponding text files.

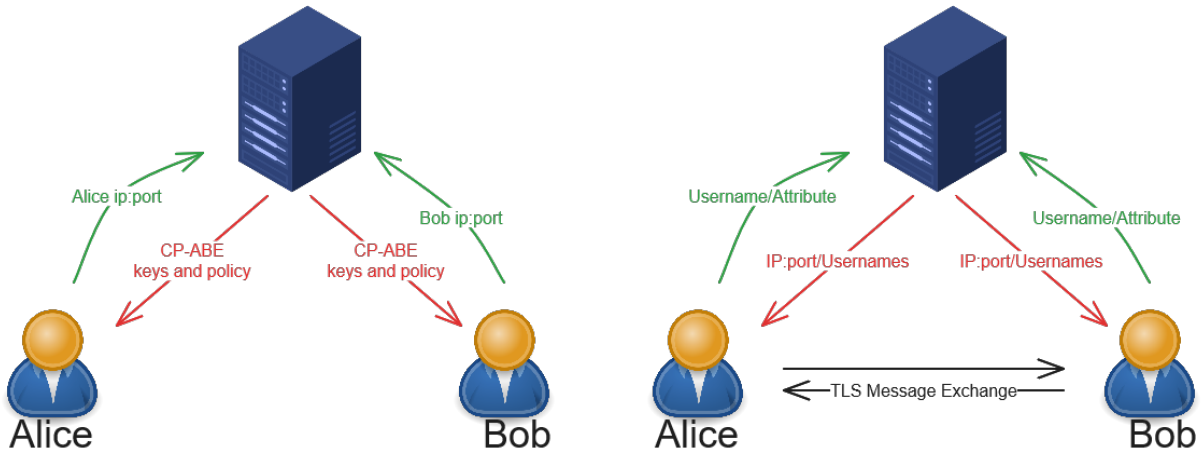

Figure 1: Messages exchanged between peers and servers

3 Long-term Message Storage

In the context of long-term storage, we employ a strategy that involves storing a copy of messages in an encrypted file, protected by an Advanced Encryption Standard (AES) key, whenever the chat application is closed. The encrypted file containing the messages is stored across three different cloud providers: Google Drive, Dropbox, and GitHub. To enhance the security of the AES key, we utilize Shamir's Secret Sharing (SSS) scheme. This scheme divides the AES key into multiple shares, each of which is stored with a different cloud provider. The SSS scheme ensures that even if one or more shares are compromised, the original key cannot be reconstructed without the necessary minimum number of shares. We used a threshold of 2 out of the 3 total parties, guaranteeing security against a minority of malicious parties. This method provides an additional layer of security and ensures the availability of the messages even if one or more of the cloud providers are compromised.

Unfortunately, even though we implemented and tested the use of the three cloud providers, after their integration into the JavaFX project, it became impossible to acquire their share on Github. Nevertheless, we still used all 3 cloud providers to store the encrypted message files, and due to the threshold having value 2, the program execution still proceeded accordingly with only the 2 shares acquired.

This strategy ensures that even if one or more of the cloud providers are compromised, the original AES key cannot be reconstructed without the necessary minimum number of shares. Therefore, the messages remain secure even in the event of a breach by one or more of the cloud providers.

The process of recovering the AES key from the secret shares, which are retrieved from the cloud, and combining them takes approximately 1.7 seconds. This process involves retrieving the secret shares from the cloud and merging them to reconstruct the AES key. Once the AES key is recovered, the process of uploading the files, which include both the secret shares and a copy of the encrypted messages, to the cloud providers takes around 3.4 seconds.

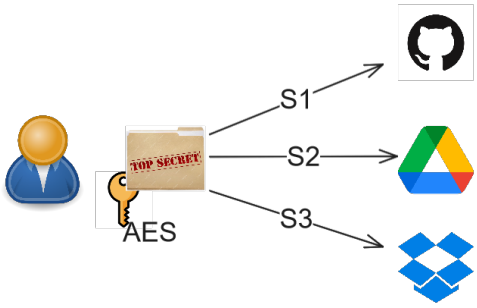

Figure 2: Uploading shares to cloud providers

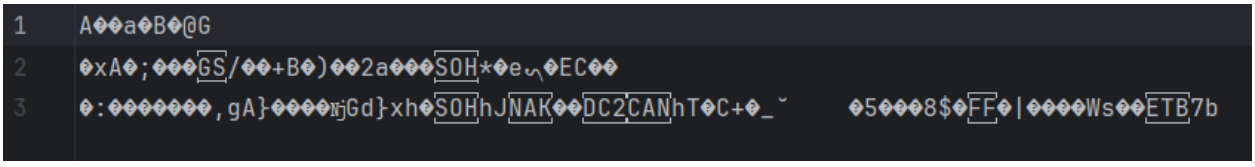

Figure 3: The content's file with the messages encrypted

4 Group Conversations

The functionality we are implementing includes the provision of secure, end-to-end encrypted group conversations among multiple users. In these group conversations, users can only send messages if they maintain an SSL socket connection with every other member of the group. The encryption scheme employed is ciphertext-policy attribute-based encryption (CP-ABE).

An access policy is established for each group chat. This policy, along with its value and the associated public key, are transmitted to each user participating in the group chat. The access policy is constructed in accordance with the tree diagram provided.

This approach ensures that only the intended recipients can view the contents of the messages, thereby providing a high level of confidentiality. The system also provides authenticity, meaning that no unauthorized party can inject a message into the system, and no unintended party can access or understand the communications without being given them by one of the correspondents.

In the context of the application, the left side of the AND operator represents the name of the

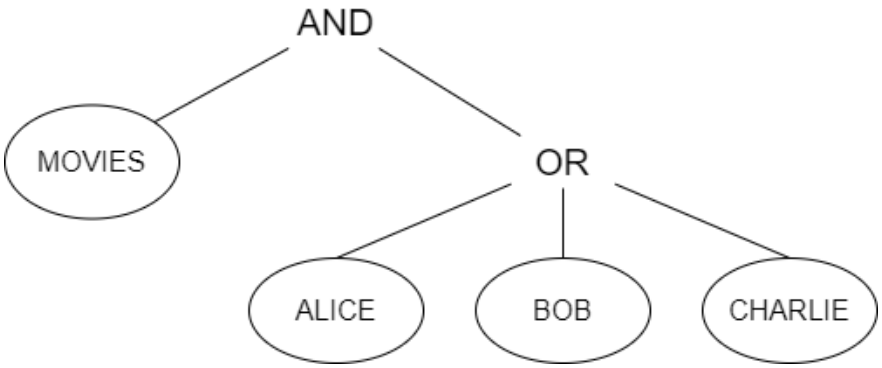

Figure 4: CP-ABE tree-based data-structure

group chat, which is used to describe the topic of interest of the group for practical purposes. This value can be easily modified to a more secure and complex value as needed. The right side of the AND operator corresponds to a disjunction of all the usernames of the elements of the group.

The attributes of each user are his username and all the names of the group chats he's in. These attributes are used for the construction of the secret key, which, after being done by the server, is sent to the corresponding user. It is important to note that each user has one access policy and public key per group chat they're in, but only one secret key.

The secret key is a critical component in this context, as it is used to encrypt and decrypt messages. The use of a secret key ensures that only authorized users can read the messages. This key is securely stored and is only accessible to the users who are part of the group chat.

5 UI Design

The user interface for the application has been developed using JavaFX. The title of the stage is set to reflect the username of the user. The chat list is constructed based on the existing message text files. The text field for message writing is disabled until a chat where all its members are online is selected. Upon selecting a chat, its saved messages stored in the encrypted text file are loaded into the user interface.

For a more detailed understanding of the interface of the desktop application, please refer to the following images:

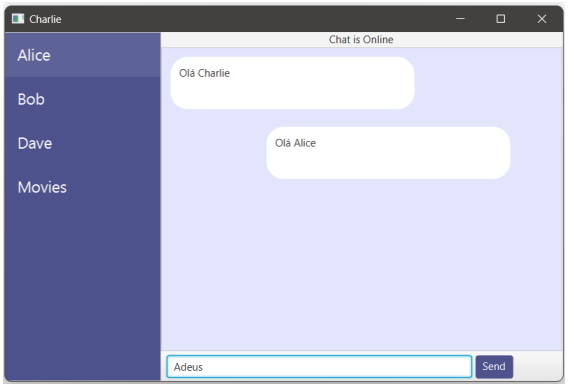

Figure 5: Private chat

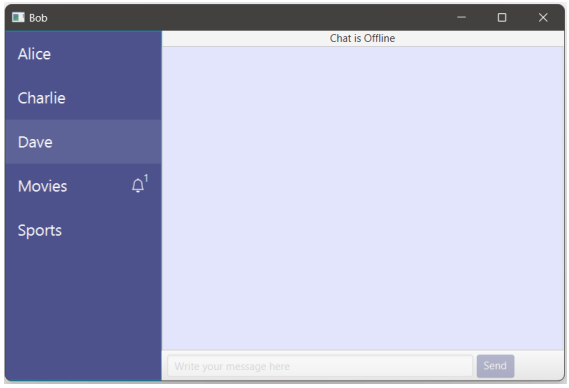

Figure 6: Message notification

These images provide a visual representation of the features described. They illustrate the interface of the application, including the username in the stage title, the chat list, the text field for message writing, and the loading of saved messages from the encrypted text file.