Functional Specification

SecureYAC Application

Student Name:	Liucija Paulina Adomaviciute
Student ID:	21790411
Student Name:	Eryk Zygmunt Styczynski
Student ID:	21753851
Date of Submission:	14/11/2024

Table of Contents

1. Introduction

1.1. Overview

The main goal of this application is secure messaging. With Double Ratchet protocol combined with Extended Triple Diffie-Hellman key agreement protocol, this application offers a secure, peer-to-peer messaging platform with a clean and easy-to-use graphical user interface that does not rely on the security of third parties, such as servers or the service provider. The goal is not to obscure the fact that the two users are communicating but to secure the contents of the conversation.

1.2. Glossary

P2P: Peer-to-Peer, a decentralized network allowing two users to communicate directly.

X3DH: Extended Triple Diffie-Hellman, a key agreement protocol used for identity verification and establishing a shared secret.

Double Ratchet Algorithm: A cryptographic protocol used for encrypting messages in a way that ensures forward secrecy.

Tox Protocol: An open-source, serverless P2P protocol known for decentralized communication and incorporating DHT for node discovery.

2. General Description

2.1. Product / System Functions

1. Identity verification using X3DH key exchange protocol
2. Message and file encryption using The Double Ratchet Algorithm
3. File compression using Huffman Coding Algorithm
4. Message and file sending using a custom implementation of the Tox protocol for a P2P network structure, with modifications for custom encryption and key exchange.

2.2. User Characteristics and Objectives

2.2.1. User Characteristics

This application is aimed at security and privacy-conscious users who have at least minimal computer skills. The user's goal is to have secure and private conversations with other users without relying on servers or service providers. The typical customer will use other privacy-enhancing features such as secure browsing and limiting data sharing with third parties. They will carefully select applications they use based on how much data they collect, and share and who has access to that data. The main challenge for the user is to find reliable, trustworthy and secure open-source applications to use for daily activities such as conversations and file sharing. To achieve more privacy and security these users are willing to sacrifice some functionality, speed and convenience.

2.2.2. User Objectives

Authentication – The system should authenticate users based on X3DH key exchange.

File Compression and Encryption – The system should compress and encrypt files before sending them.

File Transfer – The system should be able to send files between two users.

Identity – The system should generate a new identity on user's request.

Key Bundles – The system should allow user to export key bundles that other users can use to initiate messaging with them.

Messaging – The system should send messages between two users after they have authenticated each other.

Peer-to-Peer Connection – The system should allow direct peer-to-peer connection between users.

Security – The system should encrypt all outbound and decrypt all inbound messages.

User Interface – The system's user interface should provide an easy way to generate new identity, export and import key bundles, send and receive messages and files.

2.3. Operational Scenarios

1. Generate identity
2. Initialization between two actors
3. Sending message between two actors
4. Sending a file between two actors

2.4. Constraints

SecureYAC will operate as a fully serverless, P2P application. All communication is strictly peer-to-peer to ensure user privacy and security.

3. Functional Requirements

3.1. Identity Verification Using X3DH Key Agreement Protocol

- **Description**
  X3DH (Extended Triple Diffie-Hellman) is an asymmetric key protocol that establishes a shared secret key between two parties who mutually authenticate each other based on public keys. Due to the server-less nature of the application, instead of the protocol involving three parties, as defined in the whitepaper, it is modified to involve only two – Alice and Bob.

  X3DH has three phases:

  1. Bob gives his "prekey bundle" (identity key, signed prekey, prekey signature, one-time prekey) to Alice.
  2. Alice uses Bob's "prekey bundle" to send an initial message to Bob.
  3. Bob receives and processes Alice's initial message.

Publishing Keys
Identity needs to be generated only once. New signed prekeys and prekey signatures will need to be generated at some interval (once a week or once a month). The new values will replace the previous ones.
Bob creates a set of elliptic curve public keys.
After sharing a new signed prekey, the private key corresponding to the previous signed prekey will be deleted. One-time prekey private keys will be deleted as Bob exports sets containing them.

Sending the Initial Message
To perform an X3DH key agreement with Bob, Alice uses the "prekey bundle" to send Bob the initial message.

Receiving the Initial Message
Upon receiving Alice's initial message, Bob retrieves Alice's identity key and ephemeral key from the message.
If the initial decryption is successful, the protocol is complete and Bob deletes any one-time prekey private key that was used for forward secrecy. Bob then continues to use SK and keys derived from SK for further communication with Alice.

- **Criticality**
  Essential.

- **Technical issues**
  Implementation
  No Java libraries available for the protocol – it will need to be implemented from scratch.

  Key Exchange
  The parties must exchange the initial key bundle through a secure channel.

  Key Reuse
  Post-X3DH protocol must randomize the encryption key before Bob sends encrypted data. Failure to do so may cause key reuse and reduced security.

- **Dependencies with other requirements**
  None.

3.2. Message and File Encryption Using The Double Ratchet Algorithm

- **Description**
  The Double Ratchet algorithm is used by two parties (Alice and Bob) to exchange encrypted messages using a shared secret key. To prevent the decryption of future messages due to the theft of one party's keys, the symmetric-key ratchet is combined with the Diffie-Hellman ratchet, which updates the chain keys based on the Diffie-Hellman output.
  The parties derive new keys with Diffie-Hellman calculation results for every message. Because of this, the later keys cannot be calculated using the earlier ones. If the key is

unknown, the output data is indistinguishable from random. Diffie-Hellman public values are attached to the message.

In Double Ratchet between Alice and Bob, Alice initializes with Bob's public key and the initial root key (shared secret). As part of the initialization, Alice generates a new ratchet key pair using X3DH and feeds the output to the root KDF to calculate a new root key and send a chain key. The old root key may be deleted. When Alice sends her first message, she applies a symmetric-key ratchet to her sending chain key. The output is a new message key.

When Alice receives a message from Bob, it will contain a new ratchet public key. Alice applies a Diffie-Hellman ratchet step to derive new receiving and sending chain keys. Then, a symmetric-key ratchet step is applied to the receiving chain to get the message key for the received message.

- **Criticality**
  Essential.

- **Technical issues**
  No Java libraries available for the algorithm – it will need to be implemented from scratch.

- **Dependencies with other requirements**
  The Double Ratchet algorithm depends on the output of X3DH.

3.3. File Compression Using Huffman Coding Algorithm

- **Description**
  Huffman coding is an algorithm for data compression with the goal of reducing its size without the loss of details (lossless compression). The algorithm uses character frequency analysis to assign each character a code with the most frequent characters being assigned the shortest codes.
  This algorithm can be used to compress various types of files.

- **Criticality**
  High.

- **Technical issues**
  The efficiency on the algorithm depends on having frequent characters in the file.

- **Dependencies with other requirements**
  None

3.4. Peer-to-Peer Connection Using Tox Protocol

- **Description**
  SecureYAC's P2P networking architecture will be based on the Tox protocol, which supports decentralized, server-free communication. Tox's underlying framework uses a Distributed Hash Table (DHT) to help nodes discover each other, creating direct, P2P connections between involved parties. Our application will ideally adopt this protocol, but we will substitute Tox's native encryption and key exchange protocols

with our own custom security protocols (X3DH for identity verification and Double Ratchet for message encryption).

- **Criticality**
  Essential.

- **Technical issues**
  Although DHT provides efficient peer discovery, it may reveal the presence of users on the network. Strategies for DHT usage should be implemented to minimize unnecessary exposure of user availability.

- **Dependencies with other requirements**
  Depends on the completion of X3DH and Double Ratchet protocol implementations for integration into the Tox protocol framework.

4. System Architecture

4.1. Overview of Architectural Components

4.1.1. User Interface Module

The GUI, built with JavaFX, provides users with a simple way to interact with SecureYAC's features. JavaFX is chosen for its high flexibility in creating a responsive interface compatible with the Java-based backend.

4.1.2. Identity and Key Management Module

This module handles identity generation and key bundle management using the X3DH protocol. It includes functions for creating, exporting, and importing key bundles, which other users need to initiate secure messaging. It ensures that cryptographic keys are securely stored and managed according to best practices.

4.1.3. Encryption and Compression Module

This core module handles all encryption and decryption within our application. It contains implementations of the X3DH key agreement protocol for identity verification and the Double Ratchet algorithm for end-to-end encryption of messages and files. It also utilises the Huffman Coding Algorithm for compressing files before sending, optimizing transfer efficiency over P2P connections.

4.1.4. P2P Network Module

This module leverages the Tox protocol for decentralized P2P networking and incorporates a Distributed Hash Table (DHT) for peer discovery. The Tox-based network component allows direct, serverless connections between users. SecureYAC customizes this module by replacing Tox's native encryption with X3DH and Double Ratchet algorithms to meet the security requirements of potential users.

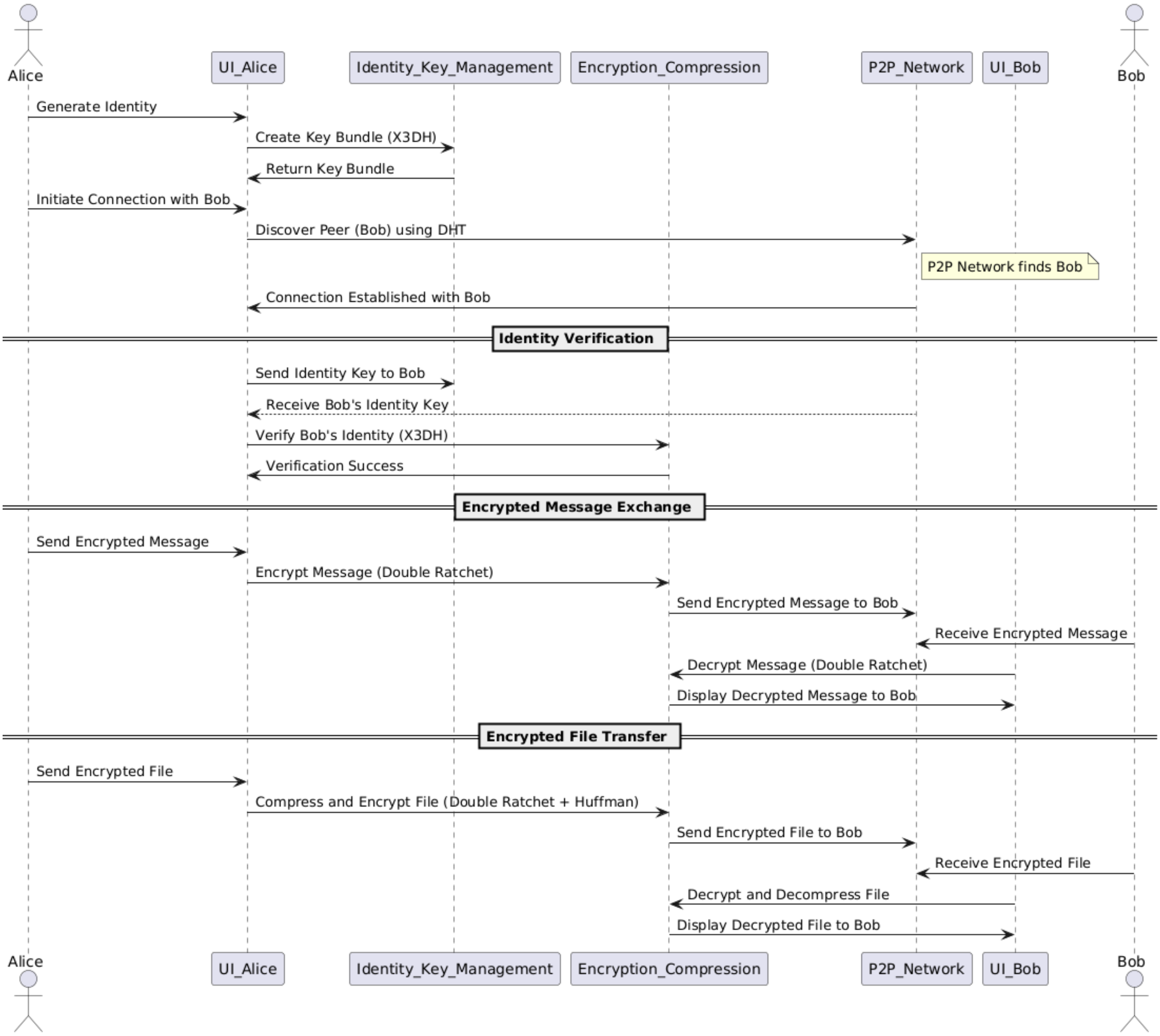
4.1.5. Messaging and File Transfer Module

This module coordinates the sending and receiving of messages and files between users. It works with the P2P Network Module to establish connections, the Encryption and

Compression Module to encrypt and compress content, and the UI to display incoming and outgoing messages/files.

4.1.6. Local Storage Module

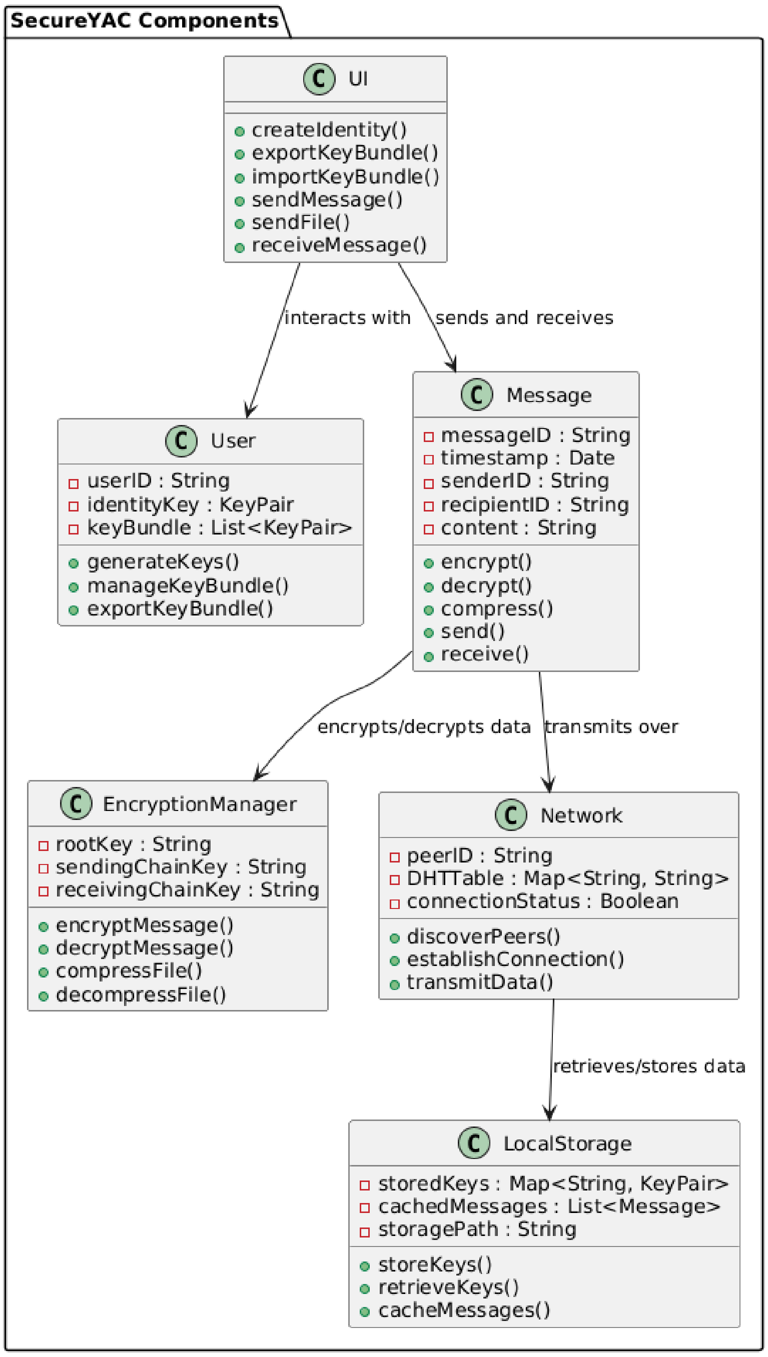
This module is responsible for securely storing local data, such as identity keys, prekey bundles, and cached messages. While the architecture remains serverless for communication, local storage is essential for managing user data and enabling persistent identity across sessions. This module may use encrypted storage mechanisms to enhance data security.


Alice
UI_Alice
Identity_Key_Management
Encryption_Compression
P2P_Network
UI_Bob
Bob
Generate Identity
Create Key Bundle (X3DH)
Return Key Bundle
Initiate Connection with Bob
Discover Peer (Bob) using DHT
P2P Network finds Bob
Connection Established with Bob
Identity Verification
Send Identity Key to Bob
Receive Bob's Identity Key
Verify Bob's Identity (X3DH)
Verification Success
Encrypted Message Exchange
Send Encrypted Message
Encrypt Message (Double Ratchet)
Send Encrypted Message to Bob
Receive Encrypted Message
Decrypt Message (Double Ratchet)
Display Decrypted Message to Bob
Encrypted File Transfer
Send Encrypted File
Compress and Encrypt File (Double Ratchet + Huffman)
Send Encrypted File to Bob
Receive Encrypted File
Decrypt and Decompress File
Display Decrypted File to Bob
Alice
UI_Alice
Identity_Key_Management
Encryption_Compression
P2P_Network
UI_Bob
Bob

Communication Flow Between Architectural Components

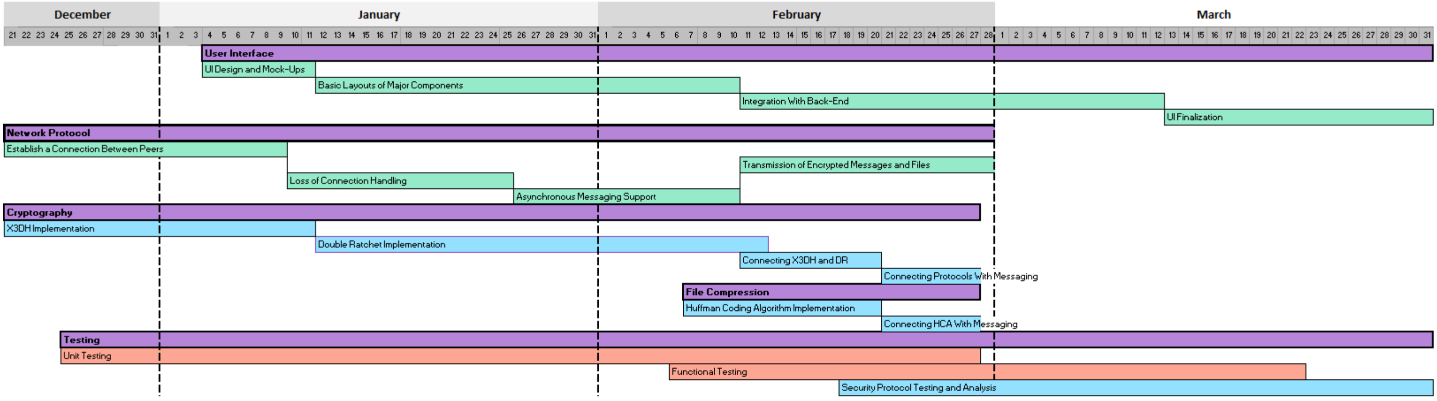
5. High-Level Design

5.1. Object Model



Class Diagram of SecureYAC Components

6. Preliminary Schedule



Gantt Chart for SecureYAC Development

6.1. User Interface

Priority: Medium
Assigned to: Eryk Styczynski
Phases:

6.1.1. UI Design and Mock-Ups
6.1.2. Basic Layouts of Major Components
6.1.3. Integration with Back-End
6.1.4. UI Finalization

6.2. Network Protocol

Priority: High
Assigned to: Eryk Styczynski
Phases:

6.2.1. Establish a Connection Between Peers
6.2.2. Transmission of Encrypted Messages and Files
6.2.3. Asynchronous Messaging Support
6.2.4. Loss of Connection Handling

6.3. Cryptography

Priority: High
Assigned to: Liucija Paulina Adomaviciute
Phases:

6.3.1. X3DH Implementation
6.3.2. Double Ratchet Implementation
6.3.3. Connecting Protocols with Messaging

6.4. File Compression

Priority: Medium
Assigned to: Liucija Paulina Adomaviciute
Phases:

6.4.1. Huffman Coding Algorithm Implementation

6.4.2. Connecting HCA with Messaging

6.5. Testing

Priority: Low-Medium
Assigned to: Eryk Styczynski & Liucija Paulina Adomaviciute
Phases:

6.5.1. Unit Testing
6.5.2. Integration Testing
6.5.3. Security Protocol Testing and Analysis

7. Appendices

7.1. Bibliography

[1] M. Marlinspike, “The X3DH Key Agreement Protocol,” Signal, Nov. 04, 2016. https://signal.org/docs/specifications/x3dh (accessed Nov. 05, 2024).

[2] M. Marlinspike, “The Double Ratchet Algorithm,” Signal, Nov. 20, 2016. https://signal.org/docs/specifications/doubleratchet/ (accessed Nov. 05, 2024).

[3] M. J. Golin and G. Rote, “A Dynamic Programming Algorithm for Constructing Optimal Prefix-Free Codes with Unequal Letter Costs,” IEEE Transactions on Information Theory, vol. 44, no. 5, pp. 1770–1781, Jan. 1998, doi: https://doi.org/10.1109/18.705558.

[4] D. Huffman, “A Method for the Construction of Minimum-Redundancy Codes,” Proceedings of the IRE, vol. 40, no. 9, pp. 1098–1101, Sep. 1952, doi: https://doi.org/10.1109/jrproc.1952.273898..