Functional Specification

PROJECT NAME

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

**Table of Contents**

[1. Introduction 3](#_Toc181728148)

[1.1. Overview 3](#_Toc181728149)

[1.2. Glossary 3](#_Toc181728150)

[2. General Description 3](#_Toc181728151)

[2.1. Product / System Functions 3](#_Toc181728152)

[2.2. User Characteristics and Objectives 3](#_Toc181728153)

[2.2.1. User Characteristics 3](#_Toc181728154)

[2.2.2. User Objectives 3](#_Toc181728155)

[2.3. Operational Scenarios 3](#_Toc181728156)

[2.4. Constraints 4](#_Toc181728157)

[3. Functional Requirements 4](#_Toc181728158)

[3.1. The X3DH Key Agreement Protocol 4](#_Toc181728159)

[3.2. The Double Ratchet Algorithm 4](#_Toc181728160)

[3.3. Placeholder 7](#_Toc181728161)

[4. System Architecture 7](#_Toc181728162)

[5. High-Level Design 7](#_Toc181728163)

[6. Preliminary Schedule 7](#_Toc181728164)

[7. Appendices 8](#_Toc181728165)

[7.1. Sources 8](#_Toc181728166)

# Introduction

## 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 secure, peer-to-peer messaging platform with 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 to not obscure the fact that the communication is happening between two users, but to secure the contents of the conversation.

## Glossary

**Chain keys** – the Key Derivation Function keys for receiving and sending chains.

**Prekeys –** X3DH protocol messages published by one party prior to the other party beginning the protocol run.

# General Description

## Product / System Functions

1. Message encryption and identity verification using X3DH with The Double Ratchet Algorithm

2. File compression using Huffman Coding Algorithm

## User Characteristics and Objectives

### 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 provider. The typical customer will use other privacy-enhancing features such as secure browsing and limiting data sharing with third-parties, and carefully select applications they use based on how much data they collect, 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.

### User Objectives

## Operational Scenarios

1. Log in

2. Identity verification between two actors

3. Sending message between two actors

4. Sending a file between two actors

5. Identity verification between multiple actors

6. Sending message between multiple actors

7. Sending a file between multiple actors

## Constraints

# Functional Requirements

## The X3DH Key Agreement Protocol

* **Description**

**X3DH Parameters**

* **Curve** – X25519.
* **Hash** – SHA-256.
* **Info** – ASCII string identifying the application.

**Cryptographic Notation**

* Concatenation of byte sequences X and Y is **X || Y**.
* **DH(PK1, PK2**) represents a byte sequence which is the shared secret output from an Elliptic Curve Diffie-Hellman function involving the key pairs represented by the public keys PK1 and PK2.
* **Sig(PK, M)** represents a byte sequence that is an XEdDSA signature on the byte sequence M and verifies with public key PK, which was created by signing M with PK’s corresponding private key.
* **KDF(KM)** represents 32 bytes of output from HKDF algorithm with inputs:
  + *HKDF input key material* = F || KM, where KM is an input byte sequence containing secret key material, and F is a byte sequence containing 32 0xFF bytes.
  + *HKDF salt* = a zero-filled byte sequence with length equal to the hash output length.
  + *HKDF info* = ACII string identifying the application.

**Roles**

The modified X3DH protocol involves two parties:

* **Alice** wants to send some initial data using encryption and also establish a shared secret key which may be used for bidirectional communication.
* **Bob** wants to allow parties like Alice to establish a shared key with him and send encrypted data.

**Keys**

X3DH uses the following elliptic curve public keys:

* **IKA** – Alice’s identity key.
* **EKA** – Alice’s ephemeral key.
* **IKB** – Bob’s identity key.
* **SPKB** – Bob’s signed prekey.
* **OPKB** – Bob’s one-time prekey.

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 uses elliptic curve public keys. All public keys have a corresponding private key. Each party has a long-term identity public key, a signed prekey, and a set of one-time prekeys used in a single protocol run. During each protocol run the parties generate new ephemeral keys.

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**

Bob creates a set of elliptic curve public keys containing:

* Bob’s identity key IKB,
* Bob’s signed prekey SPKB,
* Bob’s prekey signature Sig(IKB, Encode(SPKB)),
* Bob’s one-time prekey (OPKB).

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.

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.

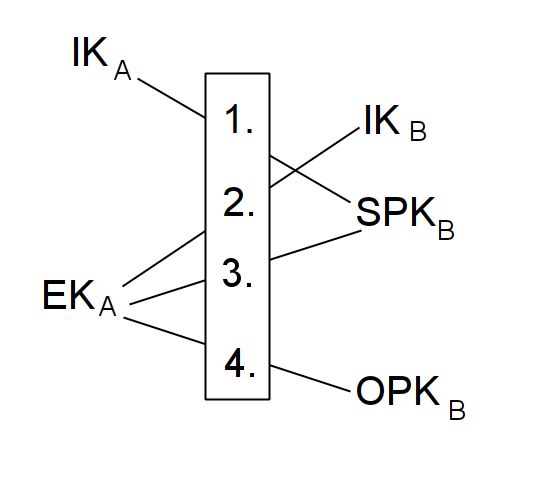
**Sending the Initial Message**

To perform an X3DH key agreement with Bob, Alice uses the “prekey bundle” containing:

* Bob’s identity key IKB,
* Bob’s signed prekey SPKB,
* Bob’s prekey signature Sig(IKB, Encode(SPKB)),
* Bob’s one-time prekey OPKB.

Alice performs the following calculations:

1. DH1 = DH(IKA, SPKB)
2. DH2 = DH(EKA, IKB)
3. DH3 = DH(EKA, SPKB)
4. DH4 = DH(EKA, OPKB)
5. SK = KDF(DH1 || DH2 || DH3 || DH4)



DH1 and DH2 provide mutual authentication, while DH3 and DH4 provide forward secrecy.

After calculating SK, Alice deletes her ephemeral private key and the DH outputs.

Then “associated data” byte sequence AD that contains information for both parties is calculated:

AD = Encode(IKA) || Encode(IKB)

Elice then sends Bob an initial message containing:

* Alice’s identity key IKA,
* Alice’s ephemeral key EKA,
* Identifiers stating which of Bob’s prekeys Alice used,
* An initial ciphertext.

**Receiving the Initial Message**

Upon receiving Alice’s initial message, Bob retrieves Alice’s identity key and ephemeral key from the message. Bob also loads his identity private key and the private key corresponding to the signed prekey and one-time prekey that was used.

Using these keys, Bob repeats the DH and KDF calculations to derive SK and then deleted the DH values.

Bob then constructs AD byte sequence using IKA and IKB. Finally, Bob attempts to decrypt the initial ciphertext using SK and AD. If the decryption fails, Bob aborts the protocol and deletes SK.

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.

**Authentication**

The parties must compare their identity keys IKA and IKB through authenticated channel.

**Protocol Replay and 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.

## 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 theft of one party’s keys and the decryption of future messages, 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.

**Key Derivation Function**

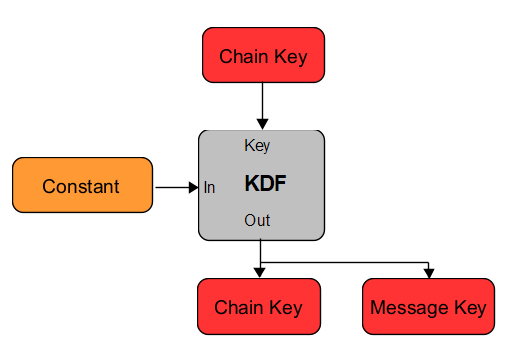
A Key Derivation Function is a cryptographic function that takes a secret and random KDF key and some input data and returns output data. In this application, the secret KDF key is the output of the function defined in the X3DH key agreement.

**Key Derivation Function Chain**

A Key Derivation Function chain is an algorithm where the output from KDF is used as an output key to replace the KDF key, which later may be used with another input.

**Symmetric-Key Ratchet**

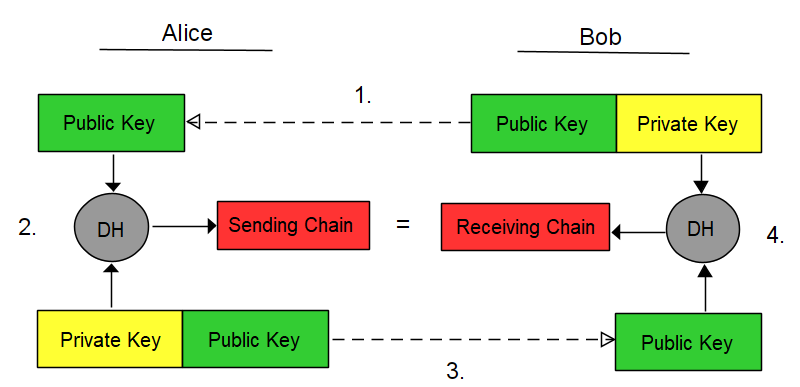
Every message is encrypted with a unique message key. The message keys are output keys from the sending and receiving KDF chains. The sending and receiving chains ensure that each message is encrypted with a unique key that can be deleted after encryption or decryption. Calculating the next chain key and message key from a given chain key is a single ratchet step in the symmetric-key ratchet.



Single step in symmetric-key ratchet

**Diffie-Hellman Ratchet**

As Alice and Bob exchange messages, they exchange new Diffie-Hellman public keys, and the output of the Diffie-Hellman becomes the input for the root chain. The outputs generated during each Diffie-Hellman ratchet step are used to derive new sending and receiving keys.

****

Single Diffie-Hellman ratchet between Alice and Bob

During the Diffie-Hellman Ratchet between Alice and Bob, the ratchet happens in four steps:

1. Alice initializes using Bob’s public key.
2. Alice performs Diffie-Hellman calculations using her private key and Bob’s public key, which results in the sending chain key.
3. Alice sends a message to Bob and advertises her public key to him.
4. Bob performs Diffie-Hellman calculations using his private key and Alice’s public key extracted from the message. The result is a receiving a chain key equivalent to Alice’s sending chain key.

After this, Bob repeats the ratchet using Alice’s advertised public key to calculate a new sending chain key using a new pair of his public and private keys.

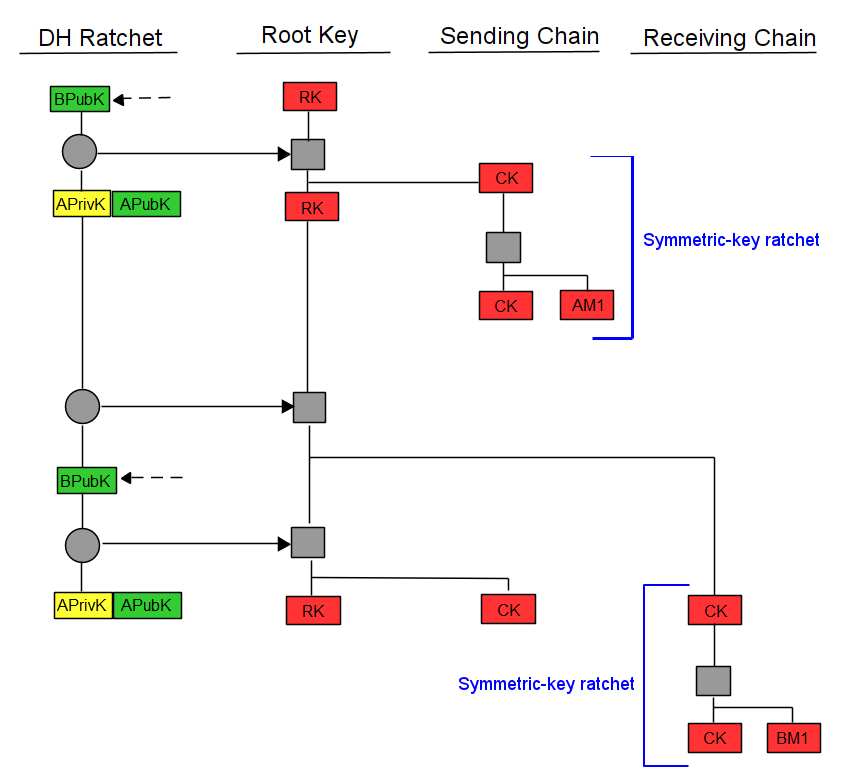
**Double Ratchet**

The Double Ratchet is a combination of symmetric-key ratchet and Diffie-Hellman ratchet.

In a Double Ratchet session between two users, each party stores a Key Derivation Function key for three chains: a root, a sending and a receiving chain. One party’s sending chain matches the other’s receiving chain.

When a message is sent or received, a symmetric-key ratchet step is applied to the sending or receiving chain to derive the message key.

When a new ratchet public key is received, a Diffie-Hellman ratchet is performed before the symmetric-key ratchet to replace the chain keys.



Single Double Ratchet between Alice and Bob

Abbreviations:

* RK – Root Key.
* CK – Chain Key.
* BPubK – Bob’s Public Key.
* APrivK – Alice’s Private Key.
* APubK – Alice’s Public Key.
* BM1 – Bob’s First Message’s Key.
* AM1 – Alice’s First Message’s Key.

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 (RK) and send a chain key (CK). The old RK 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 (AM1).

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 (BM1) 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.

## Placeholder

* **Description**
* **Criticality** – Essential
* **Technical issues**
* **Dependencies with other requirements**
* **Others as appropriate**

# System Architecture

# High-Level Design

# Preliminary Schedule

# Appendices

## Sources

Marlinspike, Moxie. “The X3DH Key Agreement Protocol.” Open Whisper Systems, 4 Nov. 2016, https://signal.org/docs/specifications/x3dh/. Accessed 5 Nov. 2024.

Marlinspike, Moxie. “The Double Ratchet Algorithm.” Open Whisper Systems, 20 Nov. 2016, https://signal.org/docs/specifications/doubleratchet/. Accessed 5 Nov. 2024.