2024 COMP90043 Project

Author
Khai Fung Lee 1242579

Author
Zhennan Zhu 1135920

Author
David Sha 1273615

Abstract

1 Background

This project focuses on implementing and evaluating a robust key exchange protocol inspired by the Extended Triple Diffie-Hellman (X3DH) and Double Ratchet Protocol, which are core elements of modern secure messaging systems such as Signal Protocol. In this report, we will outline our implementation process and evaluate these protocols in great detail, including potential vulnerabilities in the protocol and analysis of its performance overhead in space & time complexity.

1.1 Motivation

Let us preface by explaining how X3DH & Double Ratchet protocol work. We will setup the scenario of Alice wanting to talk to Bob with Signal as an example. The first half of the Signal protocol is the X3DH, which aims to mutually authenticate Alice and Bob and then generate a shared secret key for secure communication.

Initially, Bob sends a few public keys to the server: his identity public key IPK_B (verifies identity of Bob), a signed public key SPK_B (proves that he owns his identity private key and owns his one-time pre-keys), and a list of one-time session pre-keys $OPK_{B1}), OPK_{B2}, ...$ (to be used for each session of conversation with Alice).

Since Alice wants to talk to Bob, the server will send her a pre-key bundle (Bob's identity public key IPK_B, his signed public key SPK_B, and one of his one-time pre-keys OPK_B) to help establish communication with Bob.

Alice then generates her own identity public key IPK_A and ephemeral key EPK_A (a one-time use session key). With this, she will perform 4 Diffie-Hellmans (Figure 1) and append them together with a key derivation function (KDF) to produce the shared secret key that will be used to encrypt & decrypt messages. She will then send a test message encrypted with this secret key and her public keys IPK_A, EPK_A for Bob to recover the master key in the same way. The test message proves whether Bob is able to decrypt the message with the master DH key he created.

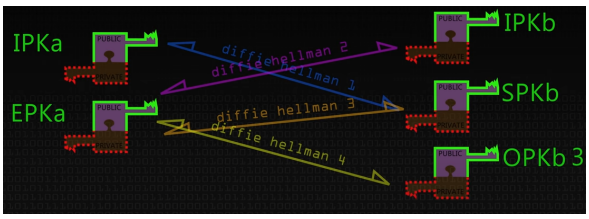

Figure 1: 4 Diffie-Hellmans from Alice & Bob's pre-key bundle

After X3DH is used to create a shared secret key, the Double Ratchet mechanisms are now used to facilitate the exchange of encrypted messages while dynamically updating the root key. The first 'ratchet' is a one-way KDF that Alice and Bob have a sending & receiving ratchet of, used to derive new message keys from the chain key. The root key (obtained from X3DH) is passed through the KDF (with another input) to output a new chain key and new message key. So, for example in Figure 2, if Alice sends a message to Bob, she will use the KDF once to output a new message key MK_{A1} to encrypt her message, which will lead to her sending ratchet ticking over. When Bob receives this message, his receiving ratchet will also tick over, in which he will pass through the same KDF with the same inputs to output the same message key MK_{A1} to decrypt Alice's message.

This process is repeated for every message, where each message leads to Alice/Bob's sending/receiving ratchet ticking over and using the current chain key to output a new chain key and message

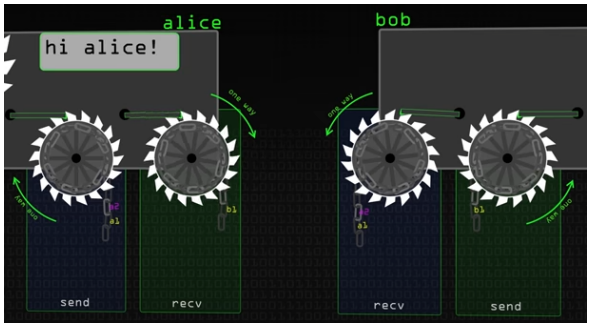

Figure 2: Visual Diagram of Double Ratchet

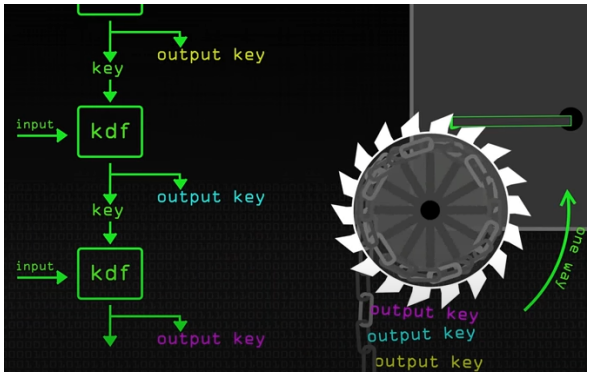

Figure 3: KDF Ratchet Function continuously running

key to encrypt messages [3]. The one-way function ensures that previous messages won't be recovered if the current chain or message key is compromised.

The second ratchet is a Diffie-Hellman ratchet that passes DH parameters as the other input into the KDF to ensure forward secrecy. These DH parameters can be used to 'reset' the first ratchet chain of message keys by creating a new master DH public key and thus update the root key.

In the Evaluation section (Section 3), we will analyze these two processes in more detail and justify the possible reasons why X3DH & Double Ratchet protocol are a robust key exchange protocol.

2 Implementation Process

2.1 Client and Server Architecture

We use a client and server architecture to model the real-world implementation of the Signal Protocol from Signal and other messaging applications alike. More specifically, the client and server architecture is modelled using Python classes (EXAMPLE) whereby we expose send (`Server.get_bundle()`) and receive (`Server.recv()`) endpoints on the server that clients can use to perform its protocol operations. In our current implementation, the server specialises in sending and receiving data related to the X3DH protocol as indicated by the use of a `Server.get_bundle()` function which is part of the second step in the X3DH protocol [].

2.2 X3DH

2.3 Double Ratchet

Based on the Signal documentation, the Double Ratchet protocol is implemented using AES-based encryption and decryption functions. The following are some key functions of the implementation:

- **RatchetInit(self, SK, bob_dh_public_key)**: Initializes the user end.
- **KDF_RK(self, rk, dh_out)**: A key derivation function (KDF) to derive the chain key from the root key. Listing 1
- **KDF_CK(self, ck)**: A KDF to derive the message key from the chain key.

```
def KDF_RK(self, rk, dh_out):
    hkdf = HKDF(
        algorithm=hashes.SHA256(),
        length=64,
        salt=rk,
        info=b'info',
        backend=default_backend())
    derived_key = hkdf.derive(dh_out)
    receive_chain_key = derived_key[:32]
    send_chain_key = derived_key[32:]
    return receive_chain_key,
    send_chain_key
```

Listing 1: Sample Implementation of Key Functions

The following are key parameters chosen for the implementation:

- **Curve: X25519**: An elliptic curve used to securely calculate the private and public keys.
- **Hash: SHA-256**: A cryptographic hash function used to generate a message authentication code (MAC).
- **Associated Data**: Some shared knowledge between users.

Encryption works as follows:

1. Use **KDF_CK** to derive a sending message key and create a header containing the new DH public keys, the message number of the previous chain, and the message number of the current chain.

2. Create a random nonce from the HKDF-derive function using the key value as input, and encrypt the plaintext using the AES encryption algorithm.

3. Generate an authentication tag (auth_tag) as a message authentication code to ensure the integrity of the message.

Decryption follows the same steps, with the additional check of the `auth_tag` to verify message integrity before decryption.

3 Evaluation

3.1 Forward Secrecy and Cryptographic Deniability

X3DH satisfies cryptographic deniability for 3 reasons:

1. Use of ephemeral keys: these are one-time use and won't be used in any following conversations with Bob

2. Diffie-Hellman key generation: no one can prove the key was generated by any one individual

3. Lack of digital signatures: can't prove who sent what

However, cryptographic deniability means no non-repudiation, which results in a lack of accountability when verifying messages, which is a natural flaw of end-to-end encrypted messaging apps. Despite this, Signal can still authenticate Alice & Bob's identity through safety numbers (Figure 4).

Double ratchet satisfies with the second DH ratchet deriving a new root key with new DH parameters every message. This means that attackers can't use old root keys with outdated DH parameters to derive new keys as new root keys are being created by Alice & Bob each message.

3.2 Results Analysis

If a user updates the Diffie-Hellman (DH) public key twice in a single message within the Double Ratchet algorithm, it could result in key synchronization issues between both parties, ultimately leading to communication failure.

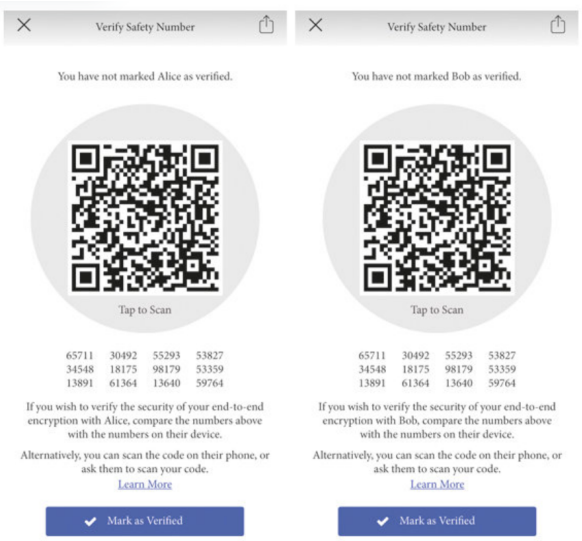

Figure 4: Identity authentication (done outside of the bounds of X3DH protocol)

3.3 Performance Overhead

3.4 Key Management and Lifecycle

Signal Protocol employs elliptic curve for key generation, primarily due to its superior efficiency and security properties, compared to traditional cryptography methods like RSA. Compared to RSA, elliptic curve provides a higher level of security per bit (a 224-bit ECC key is equivalent to a 2048-bit key), thus making key sizes smaller but as secure [1]. This low computational overhead and memory is suitable for Signal as it is primarily used on mobile devices.

3.5 Usability

3.6 Limitations of X3DH & Double Ratchet

Generally, we have found X3DH & Double Ratchet to be one of the most robust key exchange protocol. However, we found that setting up and implementing the protocol from scratch was complex and hard to replicate.
Furthermore, since Signal protocol relies on classical cryptography like ECC & Diffie-Hellman, it could be broken by quantum attacks (). One way to future-proof against these attacks could be to integrate post-quantum algorithms such as Kyber into key generation. Duits (2019) pose various feasible post-quantum cryptography for Signal

protocol.

4 Conclusion

5 Appendix

5.1 Docker

References

[1] Dindayal Mahto and Dilip Kumar Yadav. Rsa and ecc: A comparative analysis. *International journal of applied engineering research*, 12(19):9053–9061, 2017.