UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga
(Final project paper)

Pregled, analiza in implementacija modulov Signal protokola v porazdeljenem okolju

(Overview, analysis and module implementation of the Signal Protocol in a distributed environment)

Ime in priimek: Hristijan Marinkovski
Študijski program: Računalništvo in informatika
Mentor: izr. prof. dr. Jernej Vičič
Somentor: dr. Nastja Cepak

Koper, september 2021

Ključna dokumentacijska informacija

Ime in PRIIMEK: Hristijan Marinkovski

Naslov zaključne naloge: Pregled, analiza in implementacija modulov Signal protokola v porazdeljenem okolju

Kraj: Koper

Leto: 2021

Število listov: 40 Število slik: 11 Število tabel: 1

Število referenc: 10

Mentor: izr. prof. dr. Jernej Vičič

Somentor: dr. Nastja Cepak

Ključne besede: Signal, Peer-to-peer, X3DH, Double Ratchet

Izvleček: Naloga opisuje splošne značilnosti delovanja protokola Signal in prikazuje podrobnosti izvajanja modulov X3DH in Double Ratchet v porazdeljenem sistemu omrežja P2P. V nalogi so najprej analizirani kriptografski primitivi, ki so potrebni za pravilno delovanje protokola Signal in njegovih modulov. V nadaljevanju so predstavljeni štiri najpomembnejši algoritmi, ki sestavljajo protokol. Sledi prikaz integracije algoritmov X3DH in Double Ratchet ter predstavitev pomislekov, ki jih je potrebno upoštevati pri njihovi izvedbi. Analizirane so prednosti, pomanjkljivosti in upoštevanje zmogljivosti modulov protokola Signal v prej omenjenem porazdeljenem sistemu.

Key words documentation

Name and SURNAME: Hristijan Marinkovski

Title of final project paper: Overview, analysis and module implementation of the Signal Protocol in a distributed environment

Place: Koper

Year: 2021

Number of pages: 40 Number of figures: 11 Number of tables: 1

Number of references: 10

Mentor: Assoc. Prof. Jernej Vičič, PhD

Co-Mentor: Nastja Cepak, PhD

Keywords: Signal, Peer-to-peer, X3DH, Double Ratchet

Abstract: This paper concerns itself with the general functioning of the Signal protocol and showcases details regarding an implementation of the X3DH and Double Ratchet modules in a P2P networked, distributed system. Firstly, we analyze the cryptographic primitives which are necessary for the proper functioning of the Signal protocol and its modules. Afterwards, we explain the main four algorithms which construct the protocol itself. Subsequently, we showcase our integration of the X3DH and Double Ratchet algorithm, and considerations which must be taken for their implementation. We analyze the benefits, deficiencies and the performance considerations of Signal protocol modules in the aforementioned distributed system.

Acknowledgements

I would like to thank Professor Jernej Vičič for his mentorship on this thesis and his immensely valuable instructions during my studies. Thanks to his courses, guidances and advices, I gained knowledge in multiple topics related to computer science, from simpler to more advanced.

I thank Doctor Nastja Cepak for her immense guidance and patience during the writing of this thesis. She was the person that properly introduced me to the field of cryptography. The help and instruction she offered during her courses and mentorship was not only of value to the research and writing around the thesis, but also encouraged and inspired me to continue exploring many topics connected to this field.

I thank Assistant Aleksander Tošić for his incredible instructions during my studies, which were particularly interesting and informative, as well as his help related to this thesis.

I would also like to thank my friends from FAMNIT, with which I've shared many enjoyable and memorable moments around the cafes of Koper. They will help me remember my time at FAMNIT during these last few years as a gratifying experience.

A special thanks goes to my parents, for their immense sacrifices, as well as their continual guidance and support during these - for them long - years of my studies abroad.

Contents

List of Tables

List of Figures

List of Abbreviations

DH	Diffie-Hellman
E2EE	End-to-end encryption
ECC	Elliptic curve cryptography
ECDH	Elliptic curve Diffie-Hellman
EdDSA	Edwards-curve digital signature algorithm
KDF	Key derivation function
MITM	Man-in-the-middle
P2P	Peer-to-peer
RSA	Rivest–Shamir–Adleman
VDF	Verifiable delay function
VRF	Verifiable random function

1 Introduction

This paper is concerned with the the end-to-end encryption protocol - Signal and its behaviour in the context of a P2P distributed system.

E2EE is a type of encryption in which only the communicating parties can perform message encryption and decryption on sent messages. Other forms of encryption may rely on third parties performing these operations, hence posing a security risk since the user is forced to rely on a third party and trust them with their original message in plaintext format. This is also an obvious security problem in situations where the aforementioned message may contain sensitive data.

As the number of parties through which the data passes unprotected, as well as the number of parties that have control over encryption keys is minimized to just sender and receiver, E2EE is generally considered a safer alternative to contemporary encryption/decryption methods.

1.1 Symmetric and asymmetric cryptography

There are two main types of cryptographic encryption protocols. E2EE, as well as other cryptographic systems, rely on either one type, or most often on a combination of both.

The first type of encryption is called **symmetric encryption**.

The defining feature of symmetric encryption is the fact that the same key is used for both encryption and decryption during communication between parties. The communicating parties must exchange the key in order for decryption to be possible.

In contrast to symmetric encryption, the second type of encryption, **asymmetric encryption**, makes use of a keypair composed out of a public and a private key. These are two separate, yet mathematically uniquely paired keys.

Encryption is done with a publicly available key of the intended recipient, known to all exchanging and non-exchanging parties. Afterwards, the message is received by the recipient in an encrypted format, and can be decrypted with the recipient's private key.

Since the public counterpart in this keypair is available to everyone, the pair must be generated in such a manner that the private key cannot be deduced from the public

key. To achieve this, asymmetric cryptography uses encryption algorithms to create the public and private keys. These algorithms are based on certain mathematical constructs called trap door functions which cannot be efficiently reversed. The nature of the mathematical problems of reversing these trapdoor functions is such that it is borderline impossible for an outside party to derive a private key based on only its public counterpart. In practice, this means that it would take present supercomputers trillions of years to break the key.

This form of communication offers the primary advantage of communicating parties never having to share a key. Intermediary parties transferring the data have only access to the public keys, which are used for encryption, but never have access to the private keys needed for decryption.

1.2 Cryptographic primitives used in the Signal protocol

Before understanding the full functioning of the Signal protocol, we must firstly analyze the tools required for its functioning.

Two of the most common asymmetric algorithms that are used are RSA and ECC.

RSA is based around the presumed difficulty of factoring large integers. Decryption hence is infeasable on the assumption that the problem of integer factorization lacks an efficient algorithm to solve it.

The second relevant asymmetric algorithm is **ECC**. ECC is based on the presumed difficulty of the discrete logarithm problem. It is more efficient than RSA and it is therefore used by most new, modern systems that do not need to be backward compatible. Signal protocol and the blockchain environment are two such examples.

1.2.1 Elliptic curve cryptography

ECC relies on the algebraic structure of elliptic curves over finite fields. It is assumed that discovering the discrete logarithm of a random elliptic curve element in connection to a publicly known base point is impractical.

An elliptic curve is a set of points which satisfy the equation:

$$y^2 = x^3 + ax + b$$

with the limitation that $4a^3 + 27b^2 \neq 0$. The purpose of this limitation is to avoid singular points.

In order to use elliptic curves in an efficient cryptographic system, we need to find an operation that is easy and efficient to compute (it will be used for encryption), but

computationally impossible to reverse (decryption will be impossible without an extra piece of information - the private key). In the ECC scheme, the operation is scalar multiplication of a point.

Firstly, let us demonstrate how point addition is defined within the ECC system. In order to properly define the addition, it is important to point out two important properties of elliptic curves. Firstly, the curves have horizontal symmetry - that is, if a point (x, y) lies on the curve, then the point $(x, -y)$ lies on the curve as well. The second property is that any line drawn between any two different points on this elliptic curve will intersect the curve in at most one other point.

For demonstrative purposes, let us choose two randomly selected points P and Q on some graphed elliptic curve.

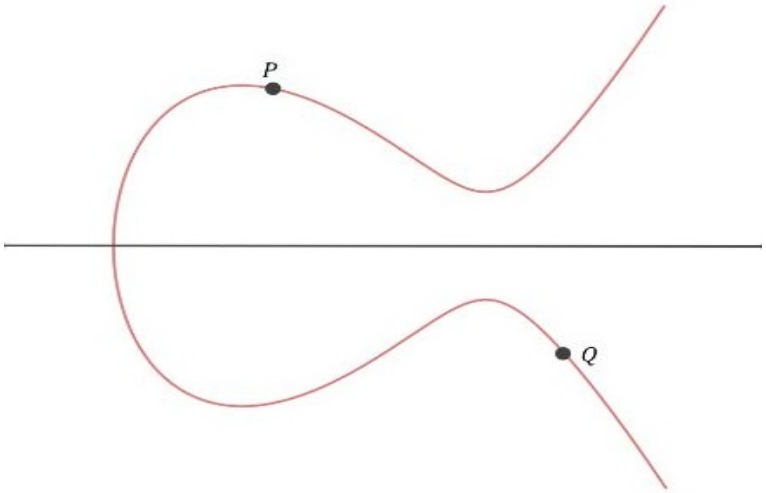

Figure 1: Selected curve with 2 points.

Here we may note the aforementioned property that any line drawn between any two different points on this elliptic curve will intersect the curve in at most one other point.

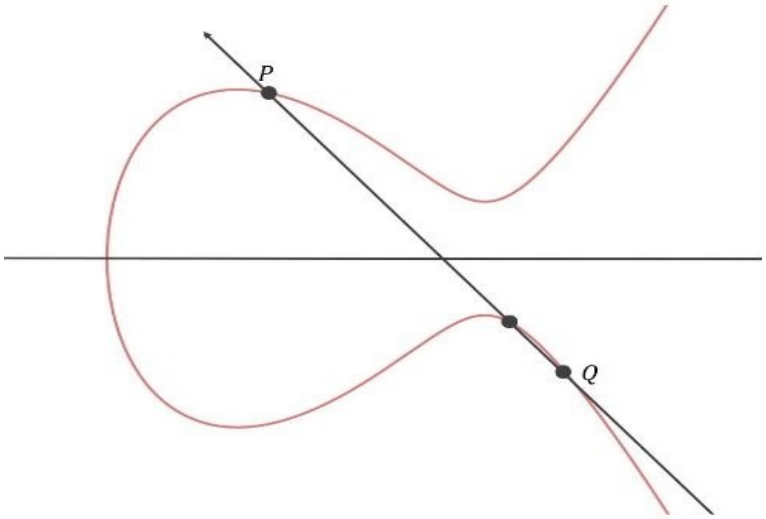

Figure 2: Line between selected points.

We mark the point of intersection and reflect it across the x axis. We denote the reflected point with R and define it to be the addition of points P and Q.

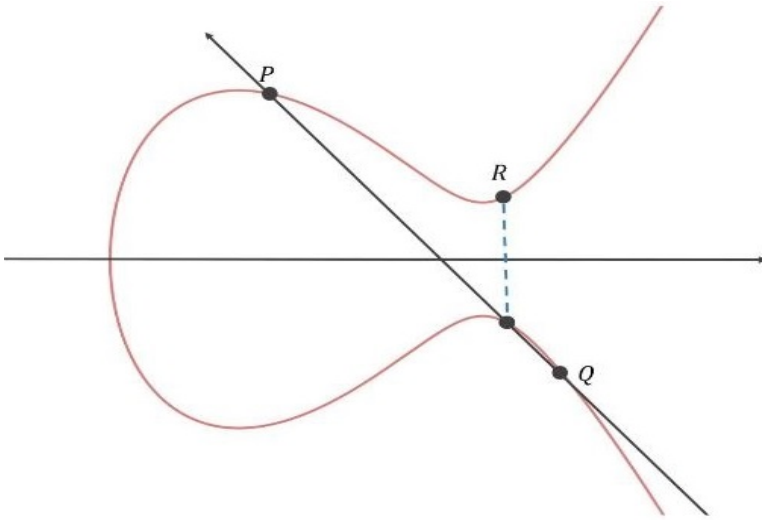

Figure 3: Reflected point on an elliptic curve

In a similar way we can also add a point to itself. Let us find a point $P + P$. In this case we draw a tangent on the curve in point P, see where it intersects the curve, and reflect it. In this way we define multiplication of a point with a scalar with "$*$". In the following example, we find the point $2 * P$.

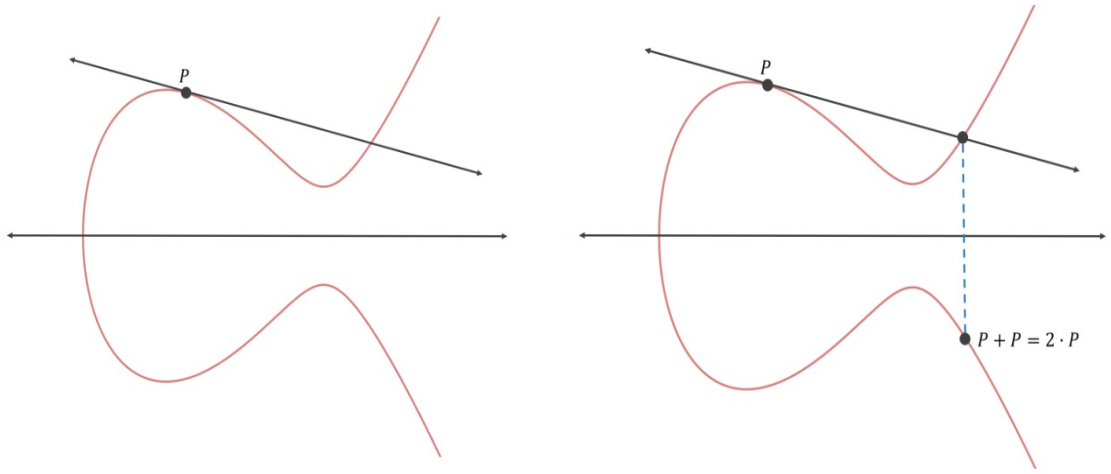

Figure 4: Scalar operation on an elliptic curve.

Note that the operation is repeatable on a single point. Obtaining a point Q with an initial point P and a random integer d is simple, as we can just repeat the operation d times. Reversing this operation, however, would require us to solve the discrete logarithm problem, which is not feasible. That is, given two points P and $Q = d * P$, it is infeasible to find the integer d.

To algebraically demonstrate this addition operation, let us choose a specific curve

$$E : y^2 = x^3 + Ax + B$$

and supposed we want to add the points

$$P_1 = (x_1, y_1),$$
$$P_2 = (x_2, y_2)$$

on this curve. Furthermore, let the line connecting P to Q be

$$L : y = \lambda x + \nu$$

with the slope and y-intercept given by

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, & \text{if } P_1 \neq P_2 \\ \frac{3x^2{}_1 + A}{2y_1}, & \text{if } P_1 = P_2 \end{cases} \text{ and } \nu = y_1 - \lambda x_1.$$

We can find the intersection of E and L by solving

$$(\lambda x + \nu)^2 = x^3 + Ax + B.$$

We know x_1 and x_2 are solutions, and we may find x_3 using the equation

$$\begin{aligned} & x^3 + Ax + B - (\lambda x + \nu)^2 \\ & = (x - x_1)(x - x_2)(x - x_3) \\ & = x^3 - (x_1 + x_2 + x_3)x^2 + (x_1x_2 + x_1x_3 + x_2x_3)x - x_1x_2x_3. \end{aligned}$$

For example, equating the coefficients of x^2 gives us

$$-\lambda^2 = -x_1 - x_2 - x_3$$

and consequently

$$x_3 = \lambda^2 - x_1 - x_2.$$

Computing y_3 using $y_3 = \lambda x_3 + \nu$ allows us to finally arive at

$$\boldsymbol{P_1 + P_2 = (x_3, -y_3).}$$

It must be noted that this example is included purely for demonstrative purposes. A more interested reader can find more detailed information on the mathematics behind ECC in [2].

To conclude, the nature of this operation functions as trap door function for elliptic curves, hence making it cryptographically relevant. The curve, the initial point P, its order and the final point Q may be used as public elements, and the random integer d may be used as the private component. Given that the operation is difficult to reverse, d can safely be used as a private component in our asymmetric cryptography keypair.

1.2.2 Elliptic curve Diffie Helman

The second important tool for the functioning of the Signal protocol is the ECDH algorithm.

ECDH is a key agreement protocol which is used by two communicating parties for the purpose of establishing a shared key. It is a variant of the standard Diffie-Hellman protocol which uses ECC. In this section we will offer a high level description of the algorithm. A more interested reader can find useful details in [2].

As a prerequestite, a curve is decided upon by both communicating parties, and a point G on that same curve. Also, each party must have their own ECC keypair.

Let us suppose Alice and Bob want to establish a safe communication channel. Firstly they generate their respective keypairs. Alice computes

$$Q_a = d_a * G$$

and Bob computes

$$Q_b = d_b * G,$$

where Q_a and Q_b respectively are the public keys, while d_a and d_b are the private keys.

After the keypair has been generated by both parties, the protocol is executed by exchanging the public keys and combining it with their own respective private keys. Alice computes:

$$SharedKeyAlice = d_a * d_b * G$$

and Bob will do the same using his private key:

$$SharedKeyBob = d_b * d_a * G.$$

The operation $*$ is commutative, so Alice and Bob will both arrive at the same key, obtaining a shared secret without directly communicating their private keys. The trap door functioning of ECC is an integral part of this key agreement protocol. If the operation $*$ could be reversed, the party which transmits the public components could derive also the private ones, computing the shared secret between the two parties.

2 Signal protocol

As we previously noted, the Signal protocol is an end-to-end encryption protocol. It was developed by Open Whisper Systems in 2012 and is currently used by Signal, WhatsApp, Facebook Messenger, and other platforms for which it is suitable.

It is primarily constructed by the following algorithms:

- XEdDSA and VXEdDSA, which are used for creating and verifying EdDSA-compatible signatures.
- X3DH, which is used for the secure computation of a shared secret between two parties who wish to communicate.
- Double Ratchet, which is used as the algorithm which determines the mode of sending and receiving messages after the shared secret has been computed.
- Sesame, which is used for the purpose of managing message encryption sessions in an asynchronous and multi-device setting.

In this chapter, we will explore these algorithms in greater detail, although it should be noted that the purpose of this chapter is only to provide a rudimentary understanding of how these algorithms function and what their purpose is within the context of the protocol. Certain information, such as security considerations and implementation details have been omitted and further reference should be obtained from the official Signal documentation which may be referenced in [1]. The following subsections are in great part summarized from the same source.

2.1 XEdDSA and VXEdDSA

A digital signature in asymmetric cryptography is an electronic verification of the sender. The purpose of such signatures is to provide authenticity of the sender to the receiver, and to assure the integrity of the message.

Generally in asymmetric cryptography, establishing a signature is done by encrypting data with the sender's private key. Data is firstly passed through a hash function and padded to avoid issues with data length, and afterwards, the said data is encrypted.

When the receiving party receives the message, they may decrypt it by using the public key of the sender. If the decrypted data is correct, the receiver has received a proof of authenticity by the sender. If the decryption step yields incorrect results however, the receiver will know that the data has either been tampered with or sent by someone other than the expected sender.

EdDSA is a modern and secure digital signature algorithm based on performance-optimized elliptic curves. Details regarding EdDSA are outside the scope of this paper. A more interested reader may find out more about this topic in [3].

The XEdDSA signature scheme, a modified version of the aforementioned EdDSA algorithm, is used by the Signal protocol for the creation and verification of these exact EdDSA-compatible signatures using public and private key formats initially defined for the X25519 and X448 elliptic curve Diffie-Hellman functions.

XEdDSA functions by enabling the use of a single keypair format for both signing and key exchange. In some cases it enables the use of the same, unmodified keypair for both algorithms, which is in general not possible.

XEdDSA is extended by VXEdDSA, which is used to make it a verifiable random function. Successful verification of a VXEdDSA signature returns a VRF output which is unique for the message and public key. Additionally, the output is seemingly random to an outside party which has not seen a VXEdDSA signature for that exact message and key.

2.2 X3DH

The X3DH key agreement protocol is used by Signal for establishing a secret symmetric key between two parties who have mutually authenticated each other using their respective public keys.

Besides having other desirable cryptographic properties, X3DH is also designed to work asynchronously. If two parties want to establish a secret key for future communications, each party may compute the key without intervention from the other directly. This also means that the party which computes said key may start sending messages immediately.

The X3DH protocol involves three parties:

- **Alice**, who wants to send Bob some initial data using encryption, and also establish a shared secret key which may be used for bidirectional communication.
- **Bob**, who wants to allow parties like Alice to establish a shared key with him and send encrypted data. However, Bob might be offline when Alice attempts to do this. To enable this, Bob has a relationship with some server.

- The **server**, which can store messages from Alice to Bob which Bob can later retrieve. The server also lets Bob publish some data which the server will provide to parties like Alice.

The purpose of the protocol is for both sides to generate a 32-byte secret key SK, and in order to achieve this, the following keys are used:

Table 1: Keypair's involved in the X3DH algorithm.

Name	**Definition**
$(IK_A^{-1},\ IK_A)$	Alice's identity keypair
$(EK_A^{-1},\ EK_A)$	Alice's ephemeral keypair
$(IK_B^{-1},\ IK_B)$	Bob's identity keypair
$(SPK_B^{-1},\ SPK_B)$	Bob's signed pre keypair
$(OTK_{B_i}^{-1},\ OTK_{B_i})$	One of Bob's one-time keypairs

The first step of the X3DH protocol consists of publishing a collection of keys to the server by Bob. This collection is set of keys which will be used as ECDH components, and they contain the following keys:

- Bob's public identity key IK_B,
- Bob's public, signed prekey SPK_B,
- Bob's prekey signature $Sig(IK_B, Encode(SPK_B))$,
- A set of Bob's one-time prekeys $(OPK_{B1}, OPK_{B2}, OPK_{B3}, ...)$.

The set of one-time prekeys is to be utilized by other parties which want to attempt to communicate with Bob, so it follows that Bob will eventually have to update it to avoid reusing the same keys.

The signed keypair may be updated at some interval in order to ensure long-term authenticity. Whilst updating the signed key, Bob may keep the private key of the previous signed key in order to handle messages which have been delayed in transit. However, for the security purpose of forward secrecy, this private key should eventually be deleted.

After Bob's collection of keys has been uploaded to the server, another party, such as Alice may attempt to establish communications with Bob. In order to do this, Alice requests Bob's prekey bundle from the server. A prekey bundle in the context of the Signal protocol contains the following keys:

- Bob's public identity key IK_B,
- Bob's public, signed prekey SPK_B,
- Bob's prekey signature $Sig(IK_B, Encode(SPK_B))$,
- (Optionally) One of Bob's one-time prekeys OPK_{Bi}.

One-time keys should always be provided, when available, and should be deleted after being given to Alice or whichever other user. The only situation where a one-time key may not be available is if all have been deleted and Bob has not uploaded any new ones.

Since a signed prekey is available and authentication must be performed, Alice firstly verifies the signature. If verification fails the protocol is of course aborted.

Alice then proceeds to generate her ephemeral key pair, and performs the following computations:

$$\begin{aligned} ECDH1 &= ECDH\,(IK_A^{-1},\, SPK_B),\\ ECDH2 &= ECDH\,(EK_A^{-1},\, IK_B),\\ ECDH3 &= ECDH\,(EK_A^{-1},\, SPK_B),\\ SK &= KDF\,(ECDH1 \,||\, ECDH2 \,||\, ECDH3). \end{aligned}$$

If the obtained pre-key bundle contains a one-time key, the following operation is added:

$$ECDH4 = ECDH(EK_A^{-1}, OPK_B).$$

After these computations are completed, their results are appended and passed through a **KDF** in order to derive the final secret key. A KDF is a cryptographic function which receives a KDF key and some input data, and uses it to derive new output data. The core strength of a KDF is that the output data is virtually indistinguishable from random data.

With this, the secret key is computed as:

$$SK = KDF\,(ECDH1 \,||\, ECDH2 \,||\, ECDH3 || \, ECDH4).$$

This flow of operation may be seen in the figure below:

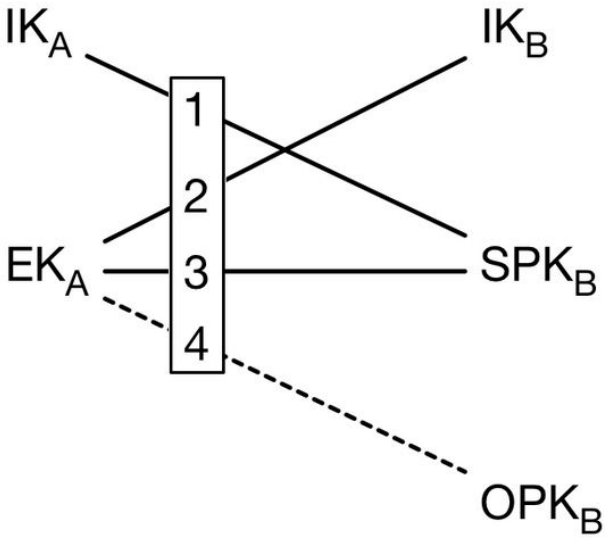

Figure 5: X3DH key computations.

Note that ECDH1 and ECDH2 provide mutual authentication, with the usage of the signed key and the identity keys of both users, while ECDH3 and ECDH4 provide forward secrecy with the usage of a temporary ephemeral key and a one-time key.

After calculating SK, Alice may freely delete private ephemeral key and the ECDH outputs, since they will no longer be necessary. She calculates associative data which contains identity information on both communicating parties:

$$AD = Encode(IKA) \,||\, Encode(IKB).$$

The associative data may also contain other information such as usernames, certificates or similar identifying information.

Alice continues by sending Bob an initial message with a bundle containing the following keys:

- Alice's identity key IK_A,
- Alice's ephemeral key EK_A,
- Identifiers stating which of Bob's data was used in the ECDH calculations,
- Initial, encrypted ciphertext.

The initial cyphertext generally functions as the first message Alice wants to send to Bob. It can also contain data which is used by the post-X3DH Double Ratchet algorithm.

After receiving the key bundle from Alice, Bob may simply continue by computing:

$$ECDH1 = ECDH\,(IK_A,\, SPK_B^{-1}),$$
$$ECDH2 = ECDH\,(EK_A,\, IK_B^{-1}),$$
$$ECDH3 = ECDH\,(EK_A,\, SPK_B^{-1}),$$
$$SK = KDF\,(ECDH1\,||\,ECDH2\,||\,ECDH3).$$

If a one-time key was provided, the following operation is added to the computations:

$$ECDH4 = ECDH(EK_A, OPK_B^{-1}),$$

changing the SK computation to:

$$SK = KDF\,(ECDH1\,||\,ECDH2\,||\,ECDH3||\,ECDH4).$$

Since the private keys act as inverses to their public counterparts, both parties derive the same secret key SK. After Bob derives the shared key SK, he may also delete the ECDH values, and construct the associative data AD using IK_A and IK_B as described previously.

Given that Bob has also received an encrypted message, he may now attempt to decrypt the message, and fail the protocol if said decryption fails. If the decryption succeeds however, he may delete the one-time private key that was used and subsequently use the SK and keys derived from it in the future.

With this, the flow of the X3DH algorithm is finished and a shared secret has been successfully computed.

2.3 Double Ratchet

After the computation of a shared secret key, the Double Ratchet algorithm is used by the communicating parties in order to send and receive messages.

At the core of the Double Ratchet algorithm is the concept of a KDF chain. The defining feature of a KDF chain is the fact that it uses the output keys generated from the function as input keys in the following function run.

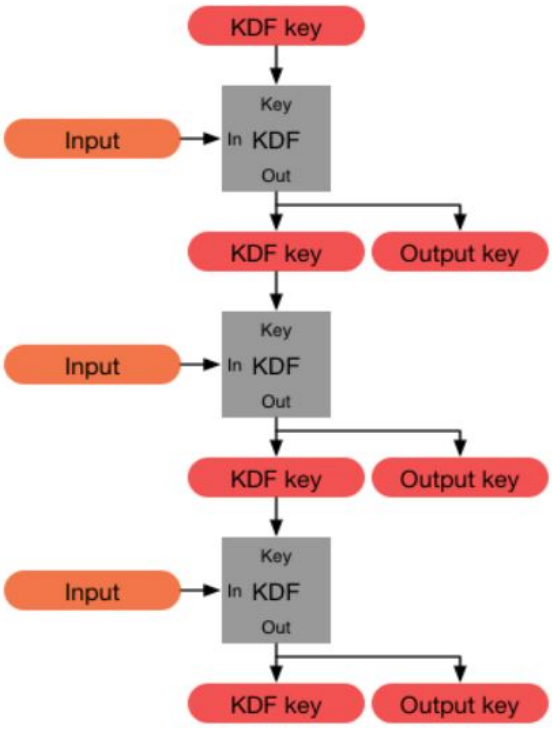

Figure 6: Basic KDF chain construction.

KDF chains are constructed in such a manner that backward secrecy is always assured. With the computation of a new key for each message, and the irreversability of the KDF itself, an attacker has no way of deriving past keys using current output from a KDF chain.

KDF chains also utilize Diffie-Hellman parameters as their inputs. This way, randomized data is mixed into the output of the KDF. Without knowledge of these Diffie-Hellman parameters, an attacker would have no way of computing the subsequent key of a KDF chain, should he obtain the current. Hence, KDF chains also provide forward secrecy. These Diffie-Helman inputs additionaly add a level of entropy during the computation of the KDF outputs.

In a Double Ratchet session between Alice and Bob each party stores a KDF key for three chains:

- A **sending chain**, which is used for encrypting sent messages. Alice's chain is synchronized with Bob's receiving chain if the algorithm is flowing correctly.
- A **receiving chain**, which is used for decrypting received messages. Alice's receiving chain is synchronized with Bob's sending chain if the algorithm is flowing correctly.

- A **root chain**, which is used to update the sending and receiving chains and provide additional security to the algorithm.

The Double Ratchet consists of two separate ratchets, namely the symmetric-key ratchet and the Diffie-Hellman ratchet.

Symmetric-key ratchet

The symmetric-key ratchet serves the purpose of providing backward security to the algorithm.

The symmetric-key ratchet functions by encrypting each sent or received message with a unique message key, which are derived from the aforementioned sending and receiving chains. From now on, we will refer to these keys as **chain keys**.

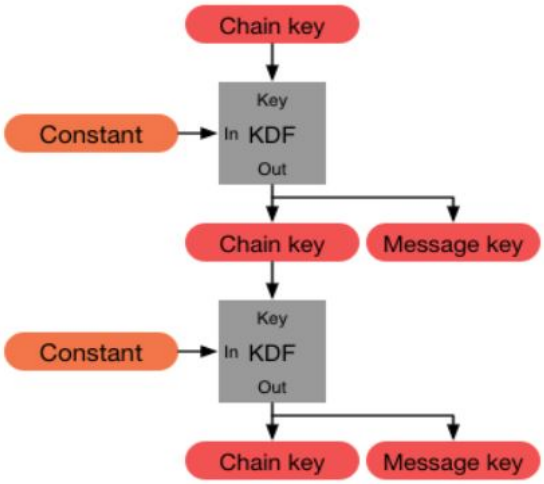

Figure 7: Symmetric key ratchet.

Calculating the next chain key and message key from a given chain key is a single ratchet step in the symmetric-key ratchet. Message keys are not used in the subsequent ratchet step. Since the KDF function, in context of practicality, is not reversible, they may freely be saved to handle out-of-order messages by the recipient.

Note that without the presence of a variable parameter as input, an attacker which obtains the chain keys of a symmetric-key ratchet may tick along with the ratchet and compute all subsequent outputs. Hence, without a further addition of entropy, the symmetric-key ratchet by itself does not provide forward secrecy to its users. This is the primary motivation for the addition of the Diffie-Helman ratchet.

Diffie-Hellman ratchet

As we previously noted, the Diffie-Hellman ratchet serves the purpose of providing forward secrecy to the Double Ratchet algorithm.

The DH ratchet begins with each party generating a DH key pair which becomes their current ratchet key pair. Alice conducts initialization using Bob's ratchet public key. As part of initialization Alice performs a DH calculation between her ratchet private key and Bob's ratchet public key.

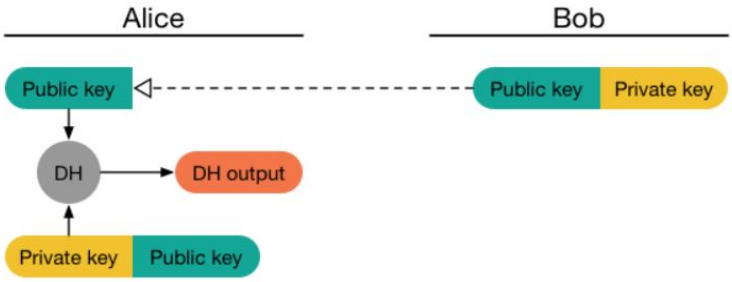

Figure 8: Initialization of the Diffie-Hellman ratchet.

Afterwards, sent messages contain the sender's current ratchet public key in the header, and upon receiving a message, a DH ratchet step is performed which replaces the receiver's current ratchet key pair with a new key pair. The scope of the DH ratchet step in the context of the Diffie-Hellman ratchet contains the following steps:

- Computation of DH output with the receiver's current ratchet key pair and the senders public key.
- Computation of DH output with a newly computed ratchet key pair and the senders public key.

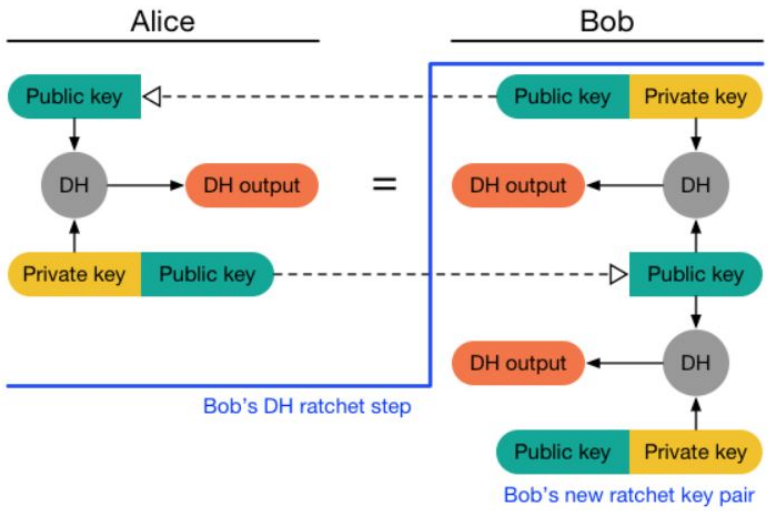

Figure 9: Bob's Diffie-Hellman ratchet step.

Messages sent back by Bob similarly advertise his new public key. Eventually, Alice will receive one of Bob's messages and perform a DH ratchet step, replacing her ratchet key pair and deriving two DH outputs, one that matches Bob's latest and a new one.

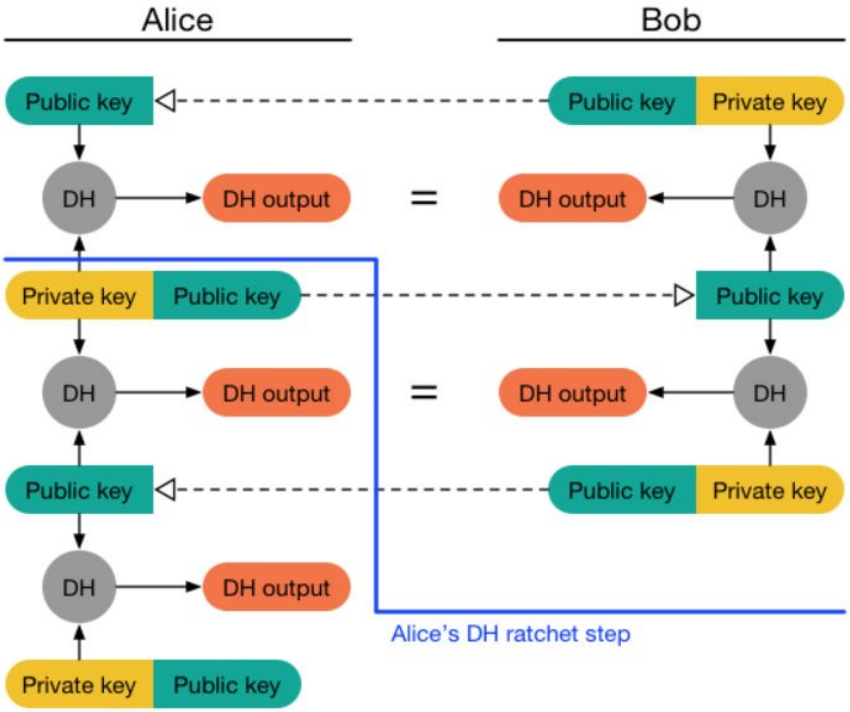

Figure 10: Alice's Diffie-Hellman ratchet step.

This "ping-pong" behaviour between Alice and Bob continues for the rest of their

communication. Note that initialization by Alice using Bob's private key must be performed before the DH ratchet can begin to function. Due to this requirement, Bob must share his DH ratchet public key before the ratcheting may begin.

The DH outputs generated during each DH ratchet step are used to derive new sending and receiving chain keys.

This is done by using the DH outputs as KDF inputs to a root chain, and the KDF outputs from the root chain are used as sending and receiving chain keys.

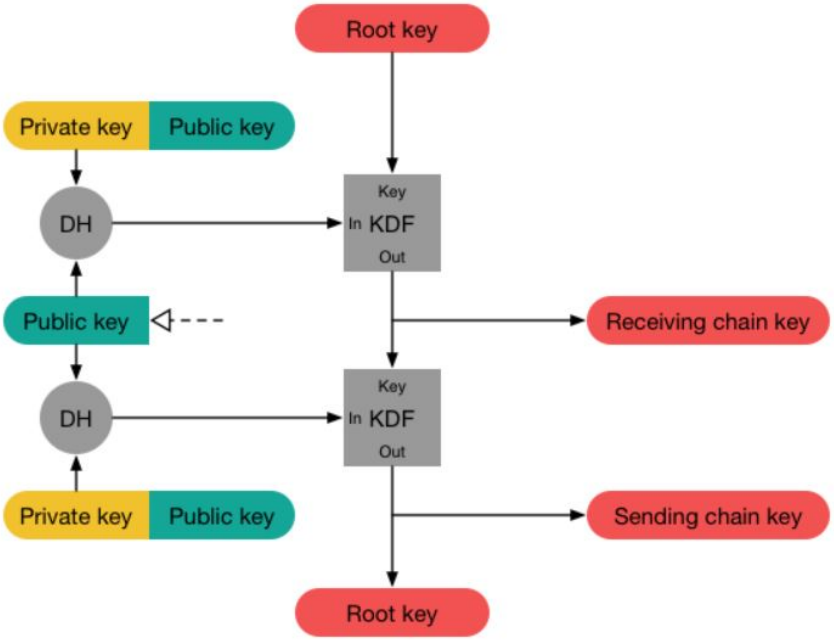

Figure 11: Connection of the Diffie-Helman ratchet outputs to the KDF chains.

Note that an attacker may compromise one of the party's keys, but it will eventually be replaced due to the nature of the algorithm. The additional DH output also provides additional randomness to the keys derived for the sending and receiving chains. Hence, this provides us with forward secrecy.

Combining the symmetric-key ratchet and the DH ratchet constructs the Double Ratchet, consisting of the following functioning:

1. 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.

2. When a new ratchet public key is received, a DH ratchet step is performed prior to the symmetric-key ratchet to replace the chain keys.

2.4 Sesame

With the computation of a shared secret by the X3DH algrotihm and messaging computations of the Double Ratchet algorithm, communications by any two parties have been established. The Signal protocol additionally handles message encryption sessions in an asynchronous and multi-device setting. This is accomplished through the Sesame algorithm.

The purpose of the Sesame algorithm stems for the fact that Alice and Bob might have several devices which they use for communication between them. Hence, the following concerns, quoted from the official Signal documentation, arise:

- Encrypting a message from Alice and Bob might encapsulate creating sessions from Alice's sending device to all of Bob's receiving devices. Alice's sending devices should at least have a copy of the sent message. Otherwise, in the scope of the sending devices that did not receive such a copy, the message was never sent.

- Alice and Bob might add and remove devices, so they will have to add and delete sessions to handle these changes.

- Alice and Bob might simultaneously initiate a new sessions with each other. For the efficacy of the Double Ratchet algorithm, Alice and Bob must send and receive messages using matching sessions. Hence, they must have a way of agreeing which matching sessions to use.

- Alice might chose to erase their private session state, restore it from a backup or similar. In this situation, the sessions which are obtained might not match Bob's sessions.

The Sesame algorithm is constructed with these complications in mind. It managed the creation, deletion and usage of sessions to resolve the aforementioned issues.

The central idea behind the algorithm is for each device to keep a record of an "active" session for each other device it communicates with. When a message is received on an "inactive" session, it becomes the new "active" session.

This is achieved by storing the state for each device. This state is then used for the sending and receiving of messages.

The state in question consists of a set of **user records** for its correspondents, indexed by user identifiers. Each of these user records contains a set of **device records**, indexed by identifiers of the devices in question. These device records in turn may contain an active session and an ordered list of inactive sessions.

Sending copies of the user's outgoing messages to his other devices is handled by containing a user record for its own identifier, without device records included.

Each device stores an identity key pair for authentication. This key pair, as well as its unique identifier, are not replaced within the algorithm flow.

Sesame has different ways of handling how key pairs are tied to users.

One option is to have the key pair be shared by all devices used by a certain user. If this is done, the identity pubic keys for the devices are stored under the set of user records.

The alternative is to have separate key pairs for each device. In this case, it follows that identity keys for the devices are stored under the set of device records.

State modification

In the context of the Sesame algortithm local state, devices have the following operations available to them:

- **Deletion of sessions, user records or device records.**
  The last session of a device record being deleted leads to the device record itself being deleted. If a the last device record of a user record is deleted, then the user record is deleted.

- **Insertion of new sessions into a device record.**
  Inserted sessions instantly become the device record's active session. Inactive sessions are not deleted, but simply stored in decreasing order based on the last time they were active. If there are too many in storage, the older used inactive sessions may be deleted.

- **Activation of an inactive session of a device record.**
  The previously used session is simply ordered below the new active session and marked as inactive.

- **Marking of a record as stale.**
  Stale records are those of users or devices has been deleted. These stale records are kept for a certain time for the purpose of decrypting delayed messages, although senders may delete them instantly.

- **Updating records based on a user identifier, device identifier and a public key.**
  If a relevant user record is not present or has an identity public key that differs from the input key, then an empty user record replaces the existing one, for the user identifier. With per-user identity public keys, the input public key is stored

in the new, empty record. Equivalently, if this happens with device records not existing or storing a differing key, the public key is stored in the empty record. If the user and device identifiers equal the device's, a new device record is not added.

- **Prepare for encryption to a user, device and public key tuple.**
  This is done by deleting relevant records if they are stale, updating the records based on the tuple information, and initialization and creation of a session if no session is currently active.

Sending messages

The Sesame sending process initializes with received plaintext and a set of user identifiers. The recipient set includes the device's own user identifier.

Quoted from the Signal documentation [1], the entire process of sending the data is encapsulated by the following steps, which are executed for each user identifier:

1. If a relevant non-stale UserRecord exists for the recipient UserID, then for each non-stale DeviceRecord in the UserRecord that contains an active session, the sending device encrypts the plaintext using that active session.

2. The recipient UserID is sent to the server, along with the list of encrypted messages and a corresponding list of DeviceIDs indicating the recipient mailbox for each message. These lists will be empty if no relevant active sessions exist.

3. If the recipient UserID is currently in-use and the sender's list of DeviceIDs is current for the recipient UserID, then the server accepts the messages and the messages are sent to the relevant mailboxes. This process then terminates for the recipient UserID, returning to step 1 for the next recipient UserID.

4. Otherwise the server rejects the messages and either informs the sending device if the recipient UserID does not exist; or informs the sending device of the old DeviceIDs and new DeviceIDs needed to make the sending device's records current, and the identity public keys corresponding to any new DeviceIDs.

5. If the server indicates that the recipient UserID does not exist, then the sending device marks the relevant UserRecord (if any) as stale. The sending device then terminates this process for the recipient UserID, returning to step 1 for the next recipient UserID.

6. For each old DeviceID, the sending device marks the relevant DeviceRecord as stale.

7. For each new DeviceID, the sending device preps for encrypting to the tuple (UserID, DeviceID, relevant public key).

8. This process is restarted from step 1 for the current recipient UserID.

For the purpose of state consistency, errors during encryption to a user will lead to the sending device discarding any changes to the relevant user record. Additionally, the continuation of encrypting and sending messages to other users may continue, or the sender may decide to terminate the entire sending process. Excessive re sending may be avoided by simply limiting the number of times the device retries to send a message to the particular user.

Receiving messages

The input to the Sesame receiving process is an encrypted message and the sender's user and device identifiers, which are fetched from the server.

Quoted from the Signal documentation [1], the entire process of receiving and decrypting data is encapsulated by the following steps:

1. If the encrypted message is an initiation message and the recipient device does not have a relevant DeviceRecord containing a session that can decrypt the message, then the relevant public key is extracted from the message header. Afterwards, the device conditionally updates its records based on the (sender's UserID, sender's DeviceID, relevant public key) tuple and lastly creates a new session using the initiating message and inserts the new session into the relevant DeviceRecord.

2. If no session in the relevant DeviceRecord can decrypt the encrypted message, then the encrypted message is discarded, all changes to device state are discarded, and this process terminates.

3. Otherwise, the message is decrypted with the relevant session.

4. If the relevant session is not active it is activated.

Errors occurring in the decryption process leads to the receiving device discarding all state changes (with the goal of state consistency), the message itself and the termination of the decryption process.

3 Implementation

Our scope of implementations around the Signal protocol were the X3DH and Double Ratchet algorithm. Signature generation and verification, and well as implementation for multi-device communications through Sesame, was excluded from the implementation.

3.1 System overview

The system in which we integrated the X3DH and Double Ratchet modules is a distributed blockchain system which was built for the purpose of container orchestrationand autonomous migration. The platform attempts to decentralize the decision making about where containerized applications should migrate in order to optimise resource consumption across the network. If successful, the architecture could replace centralized orchestration tools such as Kubernetes in order to avoid a single point of failure.

One of the key building parts of a blockchain is the consensus mechanism. The implementation uses a vote based mechanism with pseudo-random role selection. In general, for every block, nodes compute a VDF using the previous block hash as input. The VDF produces a verifiable proof, which is used as an entropy pool (shared seed) for pseudo random number generation.

Additionally, VDFs inherently can not be computed in parallel, which prevents any node to peek into the future and learn about the roles of other nodes for the next block.

After computing the proof, nodes randomly self-elect into specific roles for the current block. There are three roles, namely Block Producer, Committee Member, and Validator.

- **Block Producer** proposes a candidate block. The block is prepared, signed and delivered to the committee members. The node then awaits for attestations and in case a majority vote is reached, the block is broadcasted to the network.
- **Committee Members** are used to federate the voting process to provide scalability. It is not feasible to reach consensus in large networks if every node is expected to cast a vote. Instead, committee members are pseudo-randomly

choose to attest (vote) for a block by sending their signature to the block producer. To avoid block congestion, BLS12-381 signature scheme is used, where votes are aggregated into a single signature present in the block header.

- **Validators** are all other nodes participating in the consensus but have not been elected to perform any of the aforementioned roles for this block. Validator nodes are expect to receive a new block. Upon receiving a block, they verify the integrity and signatures of the block, and accept or reject it.

The security of the consensus mechanism depends on the assumption that a malicious actor is not able to influence the role selection, and will have an impractically small probability of having the malicious nodes have the majority vote in the committee. However, an eclipse attack could help the attacker to gain advantage by eclipsing other voting nodes. This attack is generally done by manipulating a node to make connections only to malicious nodes in the overlay network. The malicious nodes may then choose to not propagate specific messages, thereby eclipsing the node.

If committee nodes were to be eclipsed, the attacker would need a smaller number as the votes would not be received by the block producer. In order to protect against this, the entire voting protocol between the block producer, and committee members is encrypted.

3.2 Security considerations

As we previously noted, blocks are accepted or denied to the chain solely on a majority vote by the appointed committee. Hence, it follows that security in regards to how the votes are communicated is important for the system to behave as intended.

There are two main ways in which an attacker may significantly impact which blocks are given or denied attachment to the blockchain:

- **A malicious party may control the majority (or large portion) of the nodes that are in the committee.**
  While a malicious attacker may decide to intentionally accept or reject a block irregardless of verification, in order to significantly impact whether this block is accepted or denied to the blockchain consistently, they must control the majority of nodes in the committee, or at least control an impactful portion of it. However, the fact that committee members are chosen at randomly makes this very difficult for the attacker, as they would have to control a very large portion of the network nodes in order to consistently impact committee votes.

- **A malicious party may impact the votes of other members through a man-in-the-middle attack.**
  A MITM attack is a cyberattack in which the attacker intercepts communications between parties in a system, in order to either passively observe the communication, or manipulate the messages which are received by either party. In the context of our system, a successful attacker could manipulate the votes that are sent by committee members, and therefore have either enhanced, or even full control over the majority vote of a committee.

Given that the first attack is largely impractical for a potential attacker, a MITM attack on committee votes was our primary security concern, and hence, our primary motivation for the implementation of the X3DH and Double Ratchet modules in this system.

3.3 Algorithm integrations

Our primary motivation for choosing the X3DH and Double Ratchet modules specifically, is that they most contributed for security against the aforementioned MITM attack on committee votes, and they were implemented for that purpose alone.

It should be noted, however, that signature generation and verification through XEdDSA and VXEdDSA would still significantly contribute to the overall security of the system, by providing improved node authentication.

The implementations of the X3DH and Double Ratchet algorithms was done under the guidance of the official Signal documentation, which may be referenced in [1].

3.3.1 X3DH integration

While a P2P network is considered safer due to no reliance on server trust, for the integration of the Signal protocol, and more specifically, the X3DH algortihm, it is problematic.

As we previously noted in Chapter 2, the Signal protocol uses the X3DH algorithm for computing a shared secred. The algorithm functions in part by each user publishing prekey bundles to a central server, and obtaining the bundles whenever they wish to establish communication with another user.

Given that a central server is not present in a P2P system, we considered the following alternatives:

- **Simple usage of a single node(a "trusted" node) as a replacement for a central server.**

By allowing a single node to function as a replacement, the server issue is fixed as all data may just be channeled through this trusted node. However, the main drawback of this method was the fact that it would function as a performance bottleneck.

- **Replacement of a central server with a root certification authority.**
  The certification authority may also work to accomplish the function of a central server. The drawback of this method is that it would require the implementation of a public-key infrastructure, significantly increasing the complexity of the solution.

- **Exchange of prekey bundles directly between nodes that wish to communicate.**
  When nodes wish to communicate, they may simply begin the communication by sending their bundles to the recipient, or reqesting a bundle directly from the recipient. The main issue with this method is that we lose node authentication, which would normally be provided by an intermediary such as the server.

Given that node authentication was not of major concern in our system, and implementation complexity is lowest by simply allowing the nodes to directly exchange the prekey bundles, we decided on this method of implementation.

More specifically, joining nodes initiate the X3DH algorithm and begin exchanging key bundles with their acceptors as soon as they are accepted by them into the network. Each node continually holds a record of nodes and the current keys established with them.

3.3.2 Double Ratchet integration

In accordance to the Signal protocol, the shared key computed by X3DH is directly used by Double Ratchet as soon as communication occurs after a the shared key is established. With ratcheting steps occurring, chain keys are continually stored and updated by and for each node in the network.

The Double Ratchet algorithm required no specific adaptations to the system's P2P architecture.

4 Performance considerations

While generally the X3DH and Double Ratchet algorithm are not demanding on performance, it is important to consider how they will impact the system with a larger amount of nodes consistently doing the computations required by the algorithms.

4.1 X3DH performance considerations

With nodes exchanging key bundles whenever a new node joins the network, the complete exchanges and key generation connected to X3DH grows with linear asymptotic complexity. Hence, this algorithm is not an issue in terms of system performance and scalability.

Additionally, the algorithm finishes as soon as the shared key is computed between the nodes, having no impact on further computations in the system, such as committee voting and other activities.

4.2 Double Ratchet performance considerations

In our implementation, Double Ratchet directly impacts the sending and receiving of votes by the committee by encrypting and decrypting the votes, as well as ratcheting the appropriate keys.

With committee votes having to be submitted within as specific time window, lest they be skipped, the additional computations for the encryption, decryptiong and ratcheting may lead to more votes being skipped.

In our current implementation, the time limit was sufficient such that no votes were skipped with the addition of these computations, therefore having no major performance impact.

However, it should be noted that should the time limit be decreased, this algorithm may lead to committee voting skips and therefore, inaccuracies with block approval.

5 Conclusion

The Signal protocol is based on a compilation of cryptographic primitives and algorithms which can greatly contribute to the security of a system.

With their integration, the X3DH and Double Ratchet protocols specifically, can provide forward secrecy, cryptographic deniability, resilience and break-in recovery.

While a proper integration of Signal modules in a P2P architecture can be difficult, its integration can offer security benefits which outweigh potential complications during implementation, depending on the system specifics.

In blockchain systems, which may be reliant on consensus voting for block acceptance, the integration X3DH and Double ratchet modules contributes to the overall security and fair block acceptance by offering a suitable defence against MITM attacks during vote communication.

Furthermore, with proper integration, the linear asymptotic complexity of the X3DH algorithm scales well with increasing number of nodes in the system. Given that X3DH is not performance demanding, the overall performance of the system is not heavily impacted.

While the computations of the Double Ratchet algorithm are not demanding in terms of performance, it may cause issues on vote skipping if consensus votes must be passed within a short time limit. Further testing with increasingly shorter time limits may showcase more accurate results on how impactful this is to the overall accuracy of the consensus voting results.

6 Povezetek naloge v slovenskem jeziku

Naloga opisuje splošne značilnosti delovanja protokola Signal in prikazuje podrobnosti izvajanja modulov X3DH in Double Ratchet v porazdeljenem sistemu omrežja P2P

Prispevek v uvodu predstavi bralcu osnovni opis asimetrične in simetrične kriptografije, saj obe predstavljata osnovo za večino kriptografskih protokolov, vključno s Signalom. Medtem ko simetrična kriptografija uporablja en sam ključ za šifriranje in dešifriranje podatkov, asimetrična kriptografija uporablja par ključev, sestavljenih iz zasebnega in javnega ključa.

Nadaljevanje analizira kriptografske primitive, ki so potrebni za pravilno delovanje protokola Signal in njegovih modulov.

Prvi kriptografski primitiv, ki je predstavlje je "Elliptic curve cryptography"(ECC). Z opredelitvijo posebne operacije seštevanja na eliptičnih krivuljah lahko operacija deluje kot "trap door". Z drugimi besedami, lahko dobimo operacijo, ki je enostavna za izvedbo s posebnimi parametri, vendar jih je težko pridobiti iz rezultata. Te vrste operacij se lahko kasneje uporabijo kot komponente javnega in zasebnega ključa v segmentih asimetrične kriptografije protokola Signal.

Drugi kriptografski primitiv, ki je predstavljen, je "Elliptic-curve Diffie-Hellman"(ECDH). ECDH je različica standardnega Diffie-Hellman protokola za izmenjavo ključa, ki uporablja eliptične krivulje za namen vzpostavitve skupnega ključa med dvema strankama. Je osnova, na kateri temelji modul X3DH protokola Signal.

V nadaljevanju so razloženi štirje najpomembnejši algoritmi, ki sestavljajo sam protokol.

XEdDSA in VXEdDSA omogočata generiranje in preverjanje podpisov, združljivih z EdDSA, s čimer je uporabnikom protokola Signal zagotovljena plast preverjanja pristnosti. XEdDSA deluje tako, da omogoča uporabo enotne oblike para ključev za podpisovanje in izmenjavo ključev. Poleg tega ga razširja VXEdDSA z namenom, da postane preverljiva naključna funkcija.

Nadaljuje opis protokola dogovora o ključu X3DH, ki se uporablja za vzpostavitev skrivnega ključa za strani, ki želijo komunicirati. Algoritem deluje z uporabo več parov ključev in kombinacijo izračunov, ki temeljijo na omenjenem algoritmu ECDH..

Z dodajanjem KDF za dodatno varnost obe strani varno izračunata skupni ključ.

Sledi opis algoritma Double Ratchet, ki za vhod sprejme izhodni ključ X3DH. Ta algoritem služi kot funkcija, ki omogoča in dodatno varuje vsako poznejšo komunikacijo med strankama. To počne z uporabo verige KDF, ki je integrirana v simetrično in Diffie-Hellmanovo zanko(ratchet). Oboje prispeva k višji varnosti in skupaj tvorita dvojno ključavnico.

Zadnji algoritem Signal protokola ki ga obravnavamo, je Sesame, ki se uporablja za obravnavo asinhronih komunikacij in komunikacij z več napravami. Ker imajo lahko strani, ki komunicirajo, več naprav, s katerimi komunicirajo med seboj, algoritem Sesame oblikuje vrsto korakov za pošiljanje in sprejemanje sporočil v nastavitvah za več naprav.

Nadaljuje predstavitev integracije modulov X3DH in Double Ratchet v porazdeljenem sistemu P2P.

Sistem, v katerega integriramo te module, je bil zgrajen za namen uporabe v okolju veriženja blokov.. O vsakem ustvarjenem bloku mora glasovati odbor vozlišč. Bloki, ki dobijo večino glasov, se dodajo verigi blokov. Napad MITM na glasovanje bi lahko napadalcu omogočil nadzor nad velikim delom ali celo večino glasov odbora. Z dodano plastjo zaščite prek X3DH in Double Ratchet se lahko izboljša glasovalna komunikacija. To je bila glavna motivacija za našo integracijo.

Splošni scenarij delovanja protokola Signal zahteva osrednji strežnik. Za izvedbo algoritma X3DH je bilo pomanjkanje osrednjega strežnika zaradi omrežja P2P problematično, saj algoritem zahteva objavo ključnih svežnjev na strežniku za normalno delovanje. Ta problem je bil rešen tako, da je vozliščem omogočen neposreden prenos ključnih svežnjev med seboj. Algoritem Double Ratchet ni zahteval posebnih prilagoditev arhitekture P2P, njegova integracija je bila osredotočena neposredno na glasovanje v odboru.

Nazadnje je obravnavan vpliv predstavljenih algoritmov na zmogljivost sistema.

Integracija X3DH je omogočila linearno asimptotično kompleksnost. Glede na to, da izračuni niso intenzivni, sklepamo, da je razširljivost in zmogljivost sistema ohranjena.

Algoritem Double Ratchet ni računsko drag, vendar z ugotavljamo, da lahko že majhno zmanjše časovnega okvira za glasovanje v odboru povzroči povečanje števila preskočenih glasov. Ta integracija in sum na vpliv na uspešnost predstavljata podlago za nadaljnje preizkušanje povezave med dovoljenim časovnim okvirom za glasovanje v odboru in preskočenimi glasovi.

7 Bibliography

[1] *Signal documentation*, https://signal.org/docs/. (Viewed on: 8/7/2021.) *(Cited on pages 8, 21, 22,and 25.)*

[2] I. Connell, *Elliptic Curve Handbook, McGill University, 1999. (Cited on page 6.)*

[3] N. Mooney, *What's an EdDSA?, https://duo.com/labs/tech-notes/whats-an-eddsa. (Viewed on: 8/7/2021.) (Cited on page 9.)*

[4] Behind the Scenes of SSL Cryptography, https://www.digicert.com/faq/ssl-cryptography.htm. (Viewed on: 8/7/2021.) (Not cited)

[5] N. Sullivan, *A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography, https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/. (Viewed on: 8/7/2021.) (Not cited)*

[6] A Deep Dive on End-to-End Encryption: How Do Public Key Encryption Systems Work?, https://ssd.eff.org/en/module/deep-dive-end-end-encryption-how-do-public-key-encryption-systems-work. (Viewed on: 8/7/2021.) (Not cited)

[7] What are Digital Signatures? - Computerphile, https://www.youtube.com/watch?v=s22eJ1eVLTU. (Viewed on: 8/7/2021.) (Not cited)

[8] How Signal Instant Messaging Protocol Works (& WhatsApp etc) Computerphile, https://www.youtube.com/watch?v=DXv1boalsDI. (Viewed on: 8/7/2021.) (Not cited)

[9] Secret Key Exchange (Diffie-Hellman) - Computerphile, `https://www.youtube.com/watch?v=NmM9HA2MQGI`. *(Viewed on: 8/7/2021.) (Not cited)*

[10] Diffie Hellman -the Mathematics bit- Computerphile, `https://www.youtube.com/watch?v=Yjrfm_oROOw`. *(Viewed on: 8/7/2021.) (Not cited)*