Building Trust with End to End Encryption: An Introduction to the Signal Protocol

Paulo Pacitti

Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Campinas – SP – Brazil

MC889/MO421 — Introduction to Cryptography
Prof. Julio López
First half of 2023

paulo.pacitti@students.ic.unicamp.br, jlopez@ic.unicamp.br

Abstract. *This paper describes an overview on the cryptography used in the Signal Protocol, implemented in messaging apps like Signal and WhatsApp, that provides End to End Encryption (E2EE). This paper will focus on two essential algorithms used in the protocol: the Extended Triple Diffie-Hellman (X3DH) for key agreement between two users, and the Double Ratchet algorithm for key management and derivation. It also discusses possible vulnerabilities of these two algorithms and how the protocol aims to fix those weaknesses. The paper focuses on the conversation between two parties mode, since group messages use other heuristics and algorithms to ensure E2EE. By the end, the paper exposes features of the protocol that will be implemented in the future.*

1. Introduction

The Signal Protocol is a End to End Encryption (E2EE) protocol used in messaging apps like WhatsApp and Signal itself. The protocol is non-federated and the whole encryption algorithm happens on the devices of the parties that are communicating, so the server or man-in-the-middle attacks are unable to decrypt messages. First introduced by Open Whisper Systems in 2013, the open-source protocol is maintained by the Signal Foundation, founded by Brian Acton, co-founder of WhatsApp, and Moxie Marlinspike, one of the original authors of the Signal protocol.

The Signal Protocol consists mainly of two algorithms: the Extended Triple Diffie-Hellman [1] and the Double Ratchet [2]. The first is responsible for initiating a Signal Protocol session by using an extended version of the Elliptic Curve Diffie-Hellman (ECDH) algorithm that works with asynchronous environments and with an extra layer for authentication, and the later is responsible for managing keys between two parties, so one party can encrypt a message and the other can derive a key to decrypt the same one.

2. Extended Triple Diffie-Hellman (X3DH)

The Extend Triple Diffie-Hellman (X3DH) is an algorithm that aims to share a secret between two parties in an insecure channel. It's based on the classic ECDH [3][4] but has extended features such as: works within asynchronous communication, provides forward secrecy and cryptographic deniability, uses one round trip instead of two and also ensures authentication.

2.1. Curve

The X3DH implementation on libsignal is based on the X25519, an ECDH version using the Curve25519 [5]. The protocol can be used with X448 as well, but for the sake of simplicity the paper will only handle the X25519 case.

The Curve25519 elliptic curve used is a Montgomery curve and has the equation $y^2 = x^3 + 486662x^2 + x$ defining a prime field over the prime $p = 2^{255} - 19$. The X25519 uses the compressed elliptic form, using only the x-coordinate of each curve point. Because of that, the generator used is $G = 9$, defining a cyclic subgroup whose order is prime. The cofactor is $l = 8$, so that means that the prime group defined by Curve25519 has $1/8$ of the points of the original elliptic curve. By design, the curve is immune to timing attacks and any 32-byte string is accepted as a valid public key; verification to check if the point belongs to the curve is not necessary.

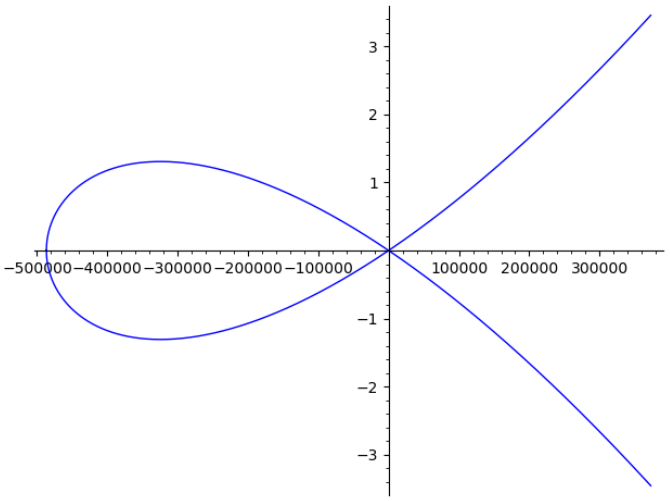

Figure 1. Curve25519 representation.

2.2 Keys

In the X3DH context, each user has a group of X25519 key pairs in which public keys are shared in a server with other users. A long-term identity keypair (IK_{pub}, IK_{priv}) identifies the user, providing authentication when used with a trusted server. In WhatsApps's case, the keypair keypair (IK_{pub}, IK_{priv}) is only replaced when the app is reinstalled, deleted or registered in a new device [6].

A short-term signed prekey keypair (SPK_{pub}, SPK_{priv}), which adds a layer for forward secrecy and ensures uniqueness to the X3DH shared secret output, is also used in the agreement. It's called prekey to make it explicit that these keys are used for the protocol usage only and are shared to the server prior to any new Signal Protocol session. SPK_{pub} is signed, so when Alice gets Bob's $SPK_{pub, Bob}$, Alice can verify if it's actually from Bob by also fetching Bob's $SPK_{pub, Bob}$ signature, signed with Bob's $IK_{priv, Bob}$, $Sig(IK_{pub, Bob}, Encode(SPK_{pub, Bob}))$, and verifying its signature. This

keypair is reset on a periodic timed basis. The encode function compresses the elliptic curve to a byte string so it can be more easily signed.

A short-term one-time prekey keypair (OPK_{pub}, OPK_{priv}), ensures forward secrecy by adding one extra layer to randomness on the X3DH shared secret output, so two X3DH agreements do not produce the same result. Once used, it should be deleted from the server (the public key) and deleted from the owner's device (private key), and replaced with new ones (normally, there's a collection of OPK_{pub} published on the server to ensure many parties can initiate a conversation using these keys). The use of this keypair is recommended in the calculation of the X3DH output, but it's not required.

A ephemeral keypair (EK_{pub}, EK_{priv}) is also used, but is only generated when Alice is initiating a session, by sending a message to Bob, and it is not shared prior to the communication. That way, Alice uses EK_{priv} in the X3DH output calculation and sends EK_{pub} along with the encrypted message to Bob, therefore he can calculate the same X3DH agreement and decrypt the message. The **Table 1** summarizes the cited keypairs used in a X3DH agreement.

Table 1. Variables to be considered on the evaluation of interaction techniques

Symbol	Type	Description
(IK_{pub}, IK_{priv})	long-term	Identity key of a user. Do not expire unless it's compromised.
(SPK_{pub}, SPK_{priv}), $Sig(IK_{pub}, Encode(SPK_{pub}))$	short-term	Signed prekey along with its signature, so it can be later verified its true authenticity.
(OPK_{pub}, OPK_{priv})	short-term	One-time prekey. Deleted from server and after usage.
(EK_{pub}, EK_{priv})	short-term	Ephemeral keys, not shared prior to the initialization message, but generated when a session is initialized.

2.3 The agreement

With these keys, after the end of a successful protocol run, two parties, Alice and Bob, will be able to calculate a shared 32-byte secret key SK. The protocol consists on Alice fetching Bob's keys from the server, which were published prior to Alice initiating the communication, then calculating the secret SK and encrypting her first message to Bob with the same secret. Bob when receiving the message, along with some keys from Alice, will be able to calculate the same secret SK and decrypt the message.

Before an agreement starts, each user has to publish their public keys to the server for authentication and to make sure other users can find each other. Each user has to publish their identity key, signed prekey, prekey signature signed with the identity

key, and a set of one-time prekeys. In the Alice and Bob example, Bob will publish his identity key $IK_{pub,\,Bob}$, his signed prekey $SPK_{pub,\,Bob}$ and its key signature $Sig(IK_{pub,\,Bob}, Encode(SPK_{pub,\,Bob}))$, and, a collection of this one-time prekeys $OPK_{pub,\,Bob,\,1}$, $OPK_{pub,\,Bob,\,2}$, $OPK_{pub,\,Bob,\,3}$... After that, Bob can be found by other users through the server and it's able to receive messages.

Once Bob has published these keys, Alice can initiate a Signal Protocol session by fetching Bob's identity key $IK_{pub,\,Bob}$, signed prekey $SPK_{pub,\,Bob}$, prekey signature $Sig(IK_{pub,\,Bob}, Encode(SPK_{pub,\,Bob}))$, and, if available on the server, a one-time prekey $OPK_{pub,\,Bob,\,1}$. If the $OPK_{pub,\,Bob,\,1}$ is fetched, then it should be deleted by the server right after to provide forward secrecy. Alice then verifies $SPK_{pub,\,Bob}$ signature using $Sig(IK_{pub,\,Bob}, Encode(SPK_{pub,\,Bob}))$. If the signature verification fails, then Alice aborts the session and tries again.

With Bob's keys, Alice can then start the Diffie-Hellman agreement to generate the secret SK to encrypt the first message to send to Bob. For this, Alice generates a ephemeral elliptic curve keypair $(EK_{pub,\,Alice}, EK_{priv,\,Alice})$ and calculates the following elliptic curve multiplications used in ECDH. At last, Alice concatenates each Diffie-Hellman calculation and derives the key using a key derivation function (KDF), resulting in SK:

$$DH_1 = DH(IK_{priv,\,Alice}, SPK_{pub,\,Bob})$$

$$DH_2 = DH(EK_{priv,\,Alice}, IK_{pub,\,Bob})$$

$$DH_3 = DH(EK_{priv,\,Alice}, SPK_{pub,\,Bob})$$

$$SK = KDF(DH_1||DH_2||DH_3)$$

If Alice had been successful on obtaining a one-time prekey from Bob, $OPK_{pub,\,Bob,\,1}$, then it calculates DH_4 as an extra layer for forward secrecy:

$$DH_3 = DH(EK_{priv,\,Alice}, OPK_{pub,\,Bob,\,1})$$

$$SK = KDF(DH_1||DH_2||DH_3||DH_4)$$

It's possible to notice that while DH_1 and DH_2 provide mutual authentication using identity, signed and owned ephemeral keys, DH_3 and DH_4 are responsible for providing forward secrecy since the arguments of both Diffie-Hellman calculations are using short-term use keys. Alice then delete all Diffie-Hellman calculations after finding SK.

Alice will then send an initial message containing her identity key $IK_{pub,\,Alice}$, her public part of her ephemeral key $EK_{pub,\,Alice}$, which one-time prekey $OPK_{pub,\,Bob}$ from Bob she used, and, an initial message with associated data encrypted using the

Encrypt-then-MAC [7] approach, with AES-256-CTR [8] using SK as a key and an authentication tag generated from the ciphertext using HMAC-256 [9][10] using a derivation from SK. The associated data contains the identity of both parties, that means $AD = Encode(IK_{pub,\,Alice})||Encode(IK_{pub,\,Bob})$ but it can also contain additional information, like certificates. After sending the message, Alice deletes the keys used for encryption.

When Bob receives the message, he is able to decrypt Alice's message calculating the following Diffie-Hellman agreements:

$$DH_1 = DH(IK_{pub,\,Alice}, SPK_{priv,\,Bob})$$

$$DH_2 = DH(EK_{pub,\,Alice}, IK_{priv,\,Bob})$$

$$DH_3 = DH(EK_{pub,\,Alice}, SPK_{priv,\,Bob})$$

$$SK = KDF(DH_1||DH_2||DH_3)$$

If Alice had been successful on obtaining a one-time prekey from Bob, $OPK_{pub,\,Bob,\,1}$, Bob will acknowledge which key Alice used when he got the message from her, then it calculates DH_4:

$$DH_3 = DH(EK_{pub,\,Alice}, OPK_{priv,\,Bob,\,1})$$

$$SK = KDF(DH_1||DH_2||DH_3||DH_4)$$

With the secret SK to decrypt Alice's message, Bob needs to also check the associated data AD using his and Alice's identity keys. Bob tries to decrypt the message using both SK and AD, if it fails, Bob aborts the protocol run. After that, Bob deletes all Diffie-Hellman calculations and secrets to provide forward secrecy.

2.4 Cryptanalysis

The X3DH algorithm is a great solution for asynchronous key exchange, but some problems can be identified. Authentication relies on the server trust, but it still can be affected by man-in-the-middle attacks. If authentication is not fully performed, the parties have no cryptographic guarantee as to who they are communicating with. To solve this, many applications use some form of public key fingerprint where they can compare or scan with a QR Code. WhatsApp fix this problem with security codes, by displaying a 60-digit fingerprint F where $F = Encode(IK_{pub,\,Alice}||\,IK_{pub,\,Bob})$ and $IK_{pub,\,Alice}$ and $IK_{pub,\,Bob}$ are respectively Alice's and Bob's device's identity keys. The **Figure 2** displays how this fingerprint is displayed in the WhatsApp app.

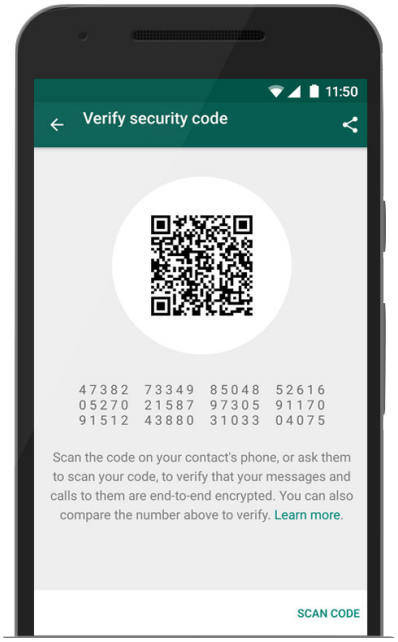

Figure 2. WhatsApp app displaying a security code for a two-party conversation.

The algorithm is also susceptible to protocol replays, allowing key reuse. If Alice's initial message doesn't use a one-time prekey, it may be replayed to Bob and he will accept it. Another consequence of replays is that a successfully replayed initial message would cause Bob to derive the same SK in different protocol runs. To mitigate this, a post-X3DH protocol may wish to quickly negotiate a new encryption key for Alice. The Double Ratchet algorithm aims to fix this.

Besides these issues, the X3DH has the privacy feature of providing cryptographic deniability. That said, unless with access to the end devices, it's not possible to prove that Alice talked with Bob and vice-versa. Bob places his prekey and its signature on a public server that anyone can access. Thus, Bob's keys are not bound to any specific peer, and cannot be used as a proof that Bob communicated with anyone. Bob can deny his involvement in the communication by claiming that his actions were based on Alice's ephemeral key. Alice can deny her involvement based on her ephemeral key because it's deleted by the protocol after the use and it's not signed or identify her in any aspect.

3. The Double Ratchet Algorithm

The Double Ratchet is an algorithm where two parties can exchange encrypted messages based on shared secret keys that update frequently. It's called "The Double Ratchet" because the algorithm is based on a ratchet construct, a one-way function chain where the output is used as the input for the next ratchet step and can't be reversed. For this, Key-Derivation-Functions (KDFs) chains are used. The Double Ratchet, as the name suggests, uses two ratchets that interact with each other: the symmetric-key ratchet and the Diffie-Hellman ratchet.

3.1. Key derivation function chains

Key derivation functions are cryptographic algorithms that takes a secret and some input data, and derive it using a pseudo-random function and outputs a new key. The output is indistinguishable from the provided key and appears totally random. The KDFs are often used to generate new keys or convert keys to new ones of a specific length. Signal Protocol uses the HMAC and the HKDF [11] algorithms as KDFs, and when used with secure hash functions, satisfies the pseudorandom function family definition [12]: (I) Resilience (output looks random to an observer with no knowledge of the internal state); (II) Forward security (Past output of the generator looks random to an observer, even if the observer learns the internal state at a later time); (III) Backward security/Break-in recovery (Future output of the generator looks random, even to an observer with knowledge of the current state). A KDF chain is when the output from a KDF is partly used as an input for another KDF iteration and so on (**Figure 3**).

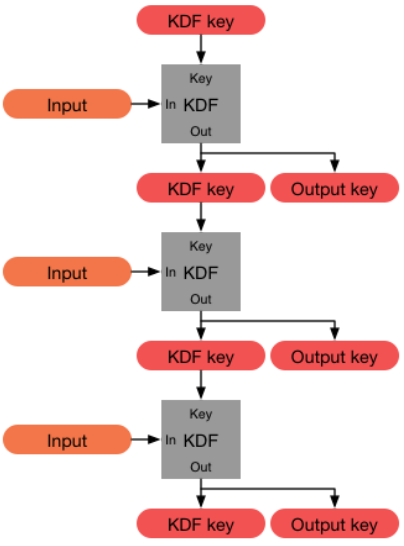

Figure 3. A Key derivation function chain.

In a Double Ratchet session between Alice and Bob, each party stores three chains: a root chain, a sending chain and a receiving chain. Alice's sending chain matches Bob's receiving chain and vice versa.

3.2. Symmetric-key ratchet

Every message sent or received is encrypted with a unique message key. The message keys are output keys from the sending and receiving KDF chains, depending if the party is sending or receiving messages. The KDF inputs used in these two chains are constant, since the break-in recovery property is maintained by the root chain, which will be managed by the Diffie-Hellman ratchet. Message keys aren't used to derive any other keys because the output of any of these two chains are split in two, the chain key and the message key, and only the chain key is used for a new KDF iteration (**Figure 4**), or better said, a symmetric-key ratchet step. Since message keys are not used to derive any other keys, message keys can be stored without affecting the security of the future message keys. This will be useful for handling out-of-order messages.

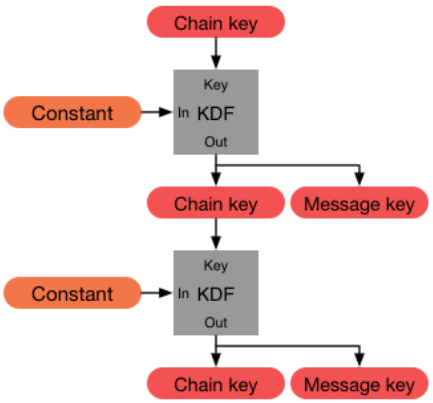

Figure 4. A Key derivation function chain.

For the symmetric-key ratchet chains, in the libsignal, main implementation of the Signal Protocol, the HMAC-SHA-256 and HKDF-SHA-256 [13] algorithms are used as KDFs. Both chain key and message key have the length of 32 bytes and both are derived using HMAC-SHA-256, but the message key is later derived using HKDF-SHA-256 and with a string literal constant as the input key material.

3.3. Diffie-Hellman ratchet

The Diffie-Hellman ratchet aims to solve the problem when an attacker steals one party's sending and receiving chain keys, since the attacker will be able to compute all future message keys and decrypt all future messages using the KDF construct in the sending and receiving chains. The Double Ratchet algorithm uses the Diffie-Helman ratchet to update chain keys in the symmetric-key ratchet based on Diffie-Hellman (DH) outputs. That provides break-in recovery since the sending and receiving chains are often reset, making new messages being encrypted with keys that derive from a new first key chain, making an attacker that managed to get one party's chain key either sending or receiving chains unable to derive future keys and decrypt future messages.

Each party generates a DH ratchet key pair, a Diffie-Hellman public and private key. Every message exchange between Alice and Bob begins with a header containing

the sender's current DH ratchet public key. When this value changes, a DH ratchet step is done to replace the receiver's current DH ratchet key pair with a new key pair. That way, Alice and Bob take turns on replacing DH ratchet key pairs, providing break-in recovery.

Alice can initiate a Double Ratchet session by fetching Bob's DH ratchet public key (**Figure 5**). Alice then calculates the Diffie-Hellman agreement with her private key from her DH ratchet key pair, which outputs a secret.

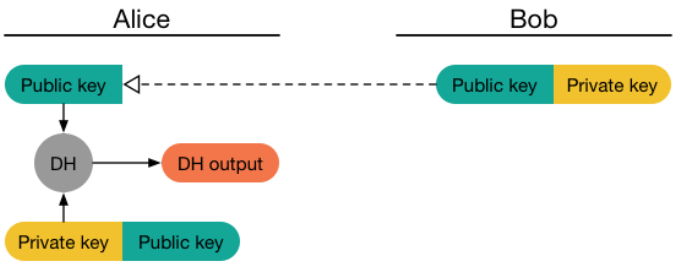

Figure 5. Alice's DH ratchet in a the initialization of a Double Ratchet session

Then Alice sends a message to Bob, with her DH ratchet public key included in the header. Once Bob receives this message, Bob can perform a DH ratchet step calculating the Diffie-Hellman agreement between his DH ratchet private key with Alice's public one, which produces the same secret that Alice calculated in her previous step. Bob then generates a new DH ratchet key pair and does another DH ratchet step with the same public key from Alice, producing a new secret (**Figure 6**).

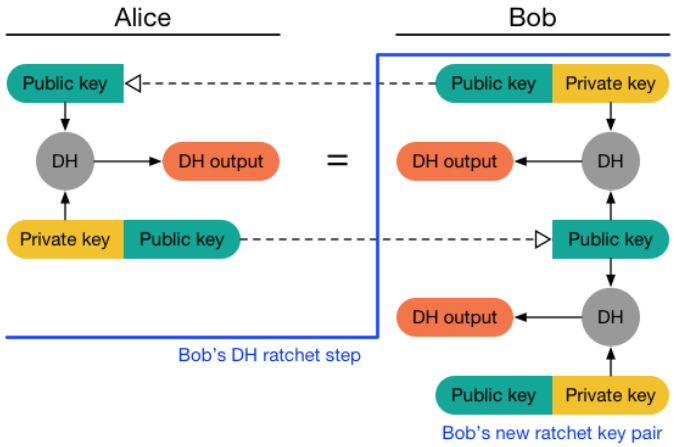

Figure 6. Bob's DH ratchet step when receiving Alice's messages

This new DH output produced by Bob can be reproduced by Alice when she performs a new DH ratchet step. That's how Alice and Bob use the Diffie-Hellman procedure along with the ratchet construct to share secrets that only Alice and Bob can

calculate, given that their keys are not compromised. Even if one key gets compromised, the attacker will be able to have a short-lived secret, because Alice or Bob will do a new DH ratchet step with a new DH public and private key pair, turning the compromised key into an expired one. The shared secret that both Alice and Bob calculated in their DH ratchet will be used by Bob to derive a receiving chain (from the symmetric-key ratchet) that matches Alice's sending chain, which derived from the same secret. Bob then uses his last DH output to generate a new sending chain to send messages to Alice (**Figure 7**).

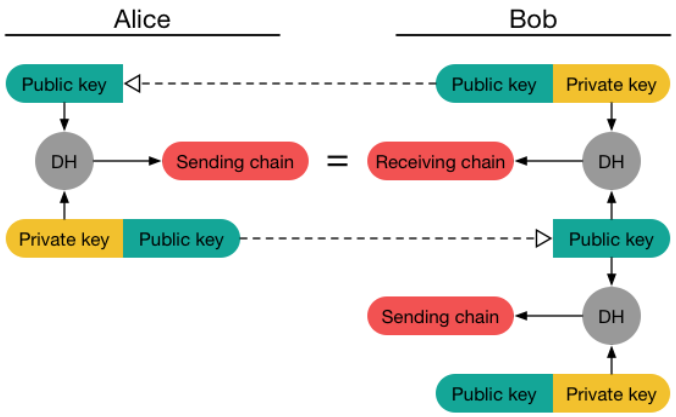

Figure 7. Bob's receiving chain should match Alice's sending chain after their agreement using the DH ratchet. Alice can later derive a new receiving chain that matches Bob's sending chain by performing a DH ratchet step.

In more detail, the DH outputs when the Diffie-Hellman agreements are calculated are not the sending and receiving chain keys itself, but are derived to generate these keys. For this, it uses the root chain described earlier, which uses the previous KDF output generated by the root chain, called root key, along with the DH output as the input key material, producing a new sending or receiving key and a new root key. The KDF construct used in the root chain is the HKDF-SHA-256. Once Bob receives Alice's message with her DH ratchet public key, Bob then calculates the Diffie-Hellman agreement and uses the DH output as input key material in his root chain to derive a new receiving chain key and a new root key. Later, Bob creates a new DH ratchet public and private key pair, calculates a new DH output using Alice's public key and uses as input key material in his root chain to derive a new sending chain key and a new root key (**Figure 8**).

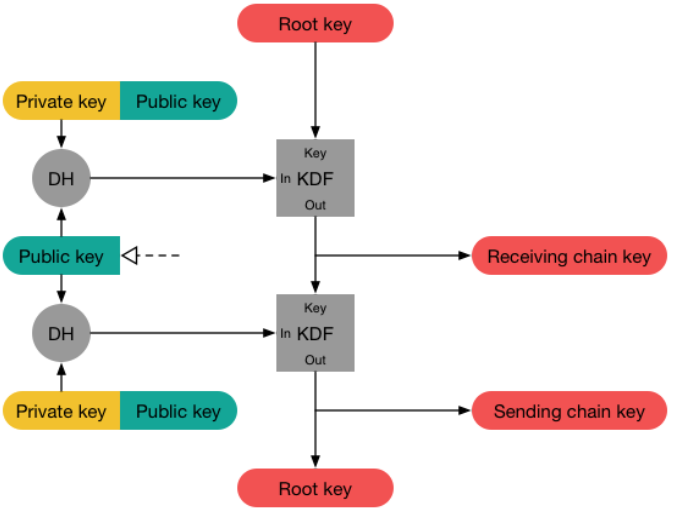

Figure 8. Bob's root chain when deriving new sending and receiving chain keys.

3.4. Double Ratchet

The algorithm uses the "Double Ratchet" as its title because it combines the symmetric-key ratchet with the Diffie-Hellman ratchet. When a message is sent or received, a symmetric-key ratchet step is performed to the corresponding (sending or receiving) chain to derive the message key and a new corresponding chain key to be used in other ratchet steps. When a new ratchet public key is received, a DH ratchet step is executed prior to the symmetric-key ratchet step to replace the chain keys, resetting the sending and receiving chains.

It's possible to understand how the algorithm works with the following example. Alice wants to initiate a conversation with Bob. For that, Alice fetches Bob's DH ratchet public key, and, with her DH ratchet public and private DH ratchet key pair, calculates the Diffie-Hellman agreement and outputs a shared secret. This shared secret key is then used in Alice's root chain to derive a new root key RK and a new chain key CK (**Figure 9**). Since the old root key won't be used anymore, it is safe to delete it to provide forward secrecy.

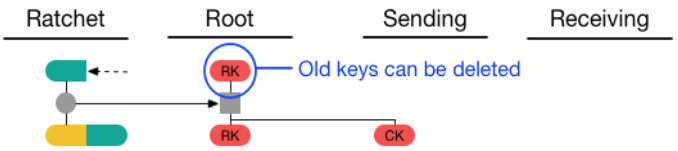

Figure 9. Alice's root, sending and receiving chain when Alice initiates a conversation with Bob. Alice performs one DH ratchet step.

Alice wants then send her message A_1 to Bob. Alice performs a symmetric-key ratchet step in her sending key chain using CK and derives a new chain key and the

message key to be used to encrypt A_1 (**Figure 10**). The old CK can be deleted since there's a new chain key for the sending chain.

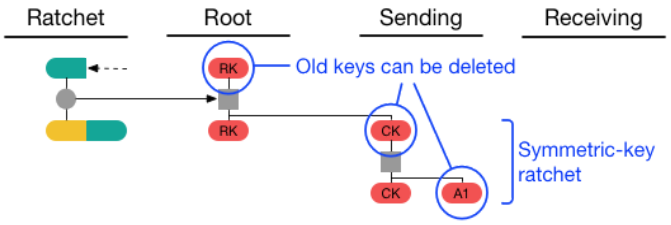

Figure 10. Alice performs one symmetric-key ratchet step in her sending chain to derive a new chain key and a new message key, used to encrypt A_1 .

Alice then receives message B_1 from Bob. That will cause Alice to execute one DH ratchet step with the new Diffie-Hellman public key from Bob, that means deriving a new key in the root chain to form the new receiving chain to obtain the key to decrypt B_1 as well forming a new sending chain to respond to Bob (**Figure 11**).

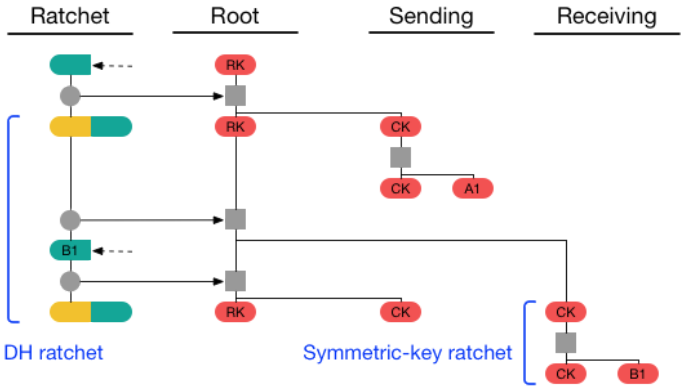

Figure 11. Alice receives B_1 from Bob, resulting in a DH ratchet step which resets both sending and receiving chain resulting from the new key derivation from root chain, caused by two Diffie-Hellman calculations from Bob's public key contained in message B_1 .

Alice then sends a message A_2, receives a message B_2 and sends two more messages A_3 and A_4 (**Figure 12**). Alice will then perform three symmetric-key ratchet steps in her sending chain and one on her receiving chain. Since neither Alice or Bob performed a DH ratchet step, the sending and receiving chains are not reset.

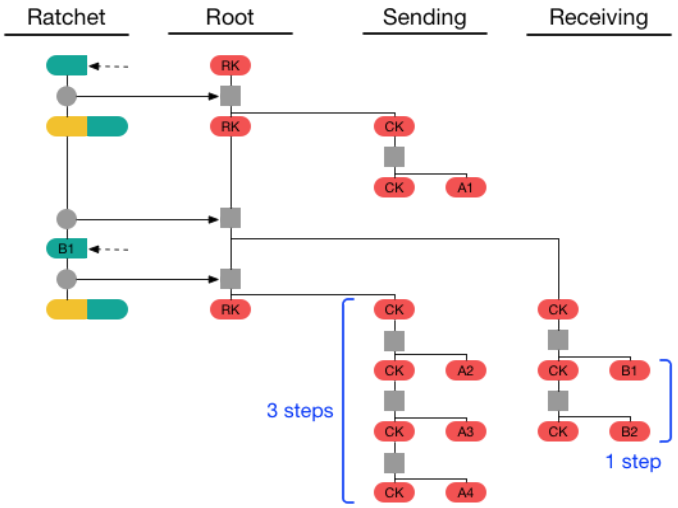

Figure 12. Alice sends A_2 to Bob, receives B_2 from Bob, then responds with A_3 and A_4 to Bob again. The figure shows how many ratchet steps the sending and receiving chains (symmetric-key ratchet steps) were performed.

3.5. Out-of-order messages

The algorithm also ensures that in case messages sent by Bob are not received in order by Alice, messages can still be decrypted without losses on message keys order. For this, each message header contains two parameters: N, the message index in its current chain, and, PN, the length (number of messages) in the previous chains. That way, if a message arrives earlier than the one expected, message keys that will not be used to decrypt the current message can be calculated and stored for future use when these messages are received.

The two parameters N (that can be 0, 1, 2…) and PN (which assume values 1,2,3…) are used to determine which message keys need to be stored with the following: if a DH ratchet step isn't triggered, then the difference between the received N and the length of the current receiving chain is the number of skipped messages in that chain.:

$$\#\{storage_{message\ keys}] = N - \#[chain_{receiving}];$$

Where $\#[storage_{message\ keys}]$ is the length of the collection of the message keys to be stored and $\#[chain_{receiving}]$ is the length of the current receiving chain. If a DH ratchet step is triggered, then the difference between the received PN and the length of the current receiving chain is the number of skipped messages in that chain. The received N is the number of skipped messages in the new receiving chain (i.e. the chain after the DH ratchet step):

$$\#\{storage_{message\ keys}] = PN - \#[chain_{receiving}] + N;$$

Imagine Alice receives B_4 from Bob prior to receive B_3 and B_2. B_4 message's header will have $N = 1$ and $PN = 2$. That means that B_4 is the second message key in the newest chain and the previous chain has two message keys. That said, being B_1 the last message received by Alice, she will ratchet once in her receiving chain to store message key for B_2 and then it will reset her receiving chain since a new one was created by observing the N parameter. By executing a DH ratchet step, the receiving chain is reset and Alice can ratchet that same chain to derive B_3. Lastly, Alice will then be able to derive B_4 (**Figure 13**), and, when the previous messages arrive, the message keys for those messages are already stored.

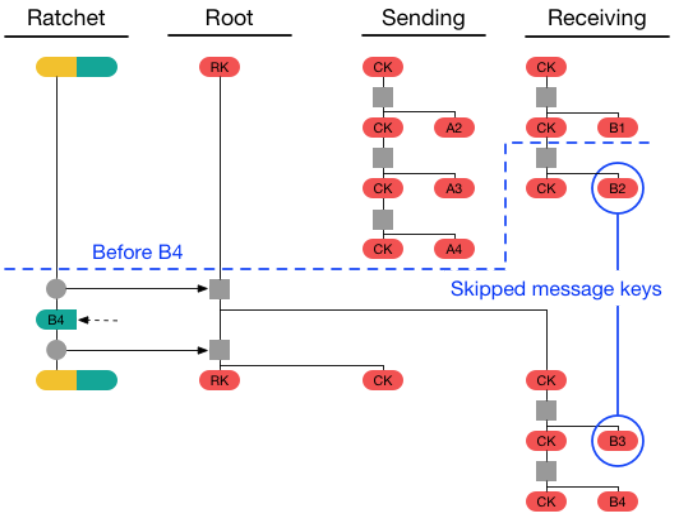

Figure 13. Alice's chains when she needs to handle out-of-order messages.

3.6. Cryptanalysis

One of the key factors of the Double Ratchet algorithm is to provide forward secrecy. That means that in the future, if a key is compromised, then the algorithm prevents the attacker from decrypting past and future messages. This is ensured by the Diffie-Hellman calculations when a DH ratchet step is performed, resetting the sending and receiving chains, not allowing the attacker to derive new keys either decrypt messages in the past, since the KDF chains do not allow the attacker to find out which was the key that derived from it.

One of the failures that the algorithm has is that an attacker could impersonate one of the parties by using the compromised party's identity private key used in the X3DH since it will be able to initiate a Double Ratchet session. Another failure was that an attacker could abuse of the random number generators (RNGs) used to generate Diffie-Hellman key pairs used in the Double Ratchet algorithm, allowing the attacker to know which keypairs could be generated and from that calculate message-keys. Since these two last vulnerabilities are failures where the attacker needs access to one party's

device or even manipulate the messaging app's implementation, the algorithm keeps cryptographic sound status.

4. Future

The Signal Protocol is widely used in many messaging applications to provide end-to-end encryption and has been more adopted over time. In 2017, researchers from the University of Oxford, University of London, and, Canada's McMaster University published an analysis of the protocol, concluding that the protocol was cryptographically sound [14]. To provide security for the near future, the protocol needs to be considered resistant to quantum computing. Since elliptic curves based cryptography is vulnerable to this type of computation, the protocol the way it is can be compromised.

To prevent this, Signal Foundation and the main implementation of the protocol libsignal, is already adding post-quantum cryptography. In version v0.27.0 of libsignal [15], the implementation finished adding post-quantum algorithms replacements such as CRYSTAL-Kyber [16], the NIST standard candidate and a in-development post-quantum algorithm called PQXDH (Post-Quantum Extended Diffie-Hellman), which uses CRYSTAL-Kyber.

References

[1] Perrin, T. and Marlinspike, M. (2016) "The X3DH Key Agreement Protocol", Revision 1, https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf

[2] Perrin, T. and Marlinspike, M. (2016) "The Double Ratchet Algorithm", Revision 1, https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf

[3] Koblitz N. (1987), "Elliptic curve cryptosystems", Math. Comp. 48 203–209. https://doi.org/10.1090/S0025-5718-1987-0866109-5

[4] Miller, V.S. (1986). "Use of Elliptic Curves in Cryptography". In: Williams, H.C. (eds) Advances in Cryptology — CRYPTO ’85 Proceedings. CRYPTO 1985. Lecture Notes in Computer Science, vol 218. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-39799-X_31

[5] Bernstein, D. J. (2006). "Curve25519: New Diffie-Hellman Speed Records". In M. Yung, Y. Dodis, A. Kiayias, & T. Malkin (Eds.), Public Key Cryptography - PKC 2006 (pp. 207–228). Springer Berlin Heidelberg, https://cr.yp.to/ecdh/curve25519-20060209.pdf

[6] WhatsApp LLC (2023). WhatsApp Encryption Overview, Technical white paper, https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf

[7] International Organization for Standardization (2009) "ISO/IEC 19772:2009. Information technology -- security techniques -- authenticated encryption". https://www.iso.org/standard/46345.html

[8] NIST (2001) Special Publication 800-38A, Recommendation for Block Cipher Modes of Operation, Methods and Techniques, https://doi.org/10.6028/NIST.SP.800-38A

[9] Bellare, M., Canetti, R., & Krawczyk, H. (1996). Keying Hash Functions for Message Authentication. Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, 1–15.

[10] Frankel, S., & Kelly, S. G. (2007). Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec. RFC Editor. https://doi.org/10.17487/RFC4868

[11] Krawczyk, H. (2010). "Cryptographic Extraction and Key Derivation: The HKDF Scheme", https://eprint.iacr.org/2010/264

[12] Barak, B. and Halevi, S. (2005). "A model and architecture for pseudo-random generation with applications to /dev/random", https://eprint.iacr.org/2005/029

[13] Hansen, T., & 3rd, D. E. E. (2011). US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC Editor. https://doi.org/10.17487/RFC6234

[14] Cohn-Gordon K., Cremers C., Dowling B., Garratt L. and Stebila D., "A Formal Security Analysis of the Signal Messaging Protocol," *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, Paris, France, 2017, pp. 451-466, doi: 10.1109/EuroSP.2017.27

[15] libsignal (2023), "Release v0.27.0 - signalapp/libsignal", https://github.com/signalapp/libsignal/releases/tag/v0.27.0

[16] Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., & Stehlé, D. (2017). CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. Cryptology ePrint Archive, Paper 2017/634. https://doi.org/10.1109/EuroSP.2018.00032

Figures

[1] WikiMedia Commons (2022), "Curve25519 geplottet über den rationalen Zahlen mit Sage", https://commons.wikimedia.org/wiki/File:Curve25519.png

[2] FA News Desk. (2021). "WhatsApp adds Quick Replies shortcut, to add end-to-end encryption indicators". Retrieved June 14, 2023 from https://www.fonearena.com/blog/353904/whatsapp-quick-replies-shortcut-end-to-end-encryption-indicators.html

[3-13] Signal (2016). "The Double Ratchet Algorithm", https://signal.org/docs/specifications/doubleratchet