A Comparison of Secure Messaging Protocols and Implementations

Aulon Mujaj

Thesis submitted for the degree of
Master in Informatics: Programming and Network
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2017

A Comparison of Secure Messaging Protocols and Implementations

Aulon Mujaj

Abstract

In recent years, it has come to attention that governments have been doing mass surveillance of personal communications without the consent of the citizens. As a consequence of these revelations, developers have begun releasing new protocols for end-to-end encrypted conversations and then commonly used chat applications have been updated with implementations of these protocols. New applications have also been developed to support these types of protocols with security in mind from the beginning. These usually contain existing and audited algorithms to ensure the encryption between participants is up to its standards.

This thesis investigates protocols for end-to-end encrypted instant messaging, focusing on the existing implementations of one of the recent and popular such protocols, called Signal. The first protocol studied is the Off-the-Record (OTR) protocol, since it was the first such protocol introduced ten years ago, and which most recent protocols are based on, or take inspiration from. Then a large part of the thesis carefully goes through the inner workings of the Signal protocol, which itself is based on OTR.

The documentations of three secure messaging protocols is studied to find what types of security and privacy properties they provide. The study of the protocol properties is also based on recent academic articles. The conclusions are summarized and explained with the purpose to be used in the rest of the thesis.

A second major part of the document is devoted to analyzing the most used secure messaging applications. A series of experiments is then conducted on these implementations to find out which types of security and usability properties each application provides. Six applications are tested. A major concern is about what kind of information the application gives to the users when cryptographic keys change during conversations, as well as how users can verify the identities of each other.

The results of the experiment show that the apps have variations of usability and security properties regarding the user's account. The apps give different amounts of information to the user about potential security attacks. While some gave enough for the user to know when cryptographic keys change, others do not provide any information.

The thesis also gives proposals for improving each application wrt. security, privacy, and usability. Hopefully, the users find the information in this research useful in choosing a particular application, and positively, encourages other researchers to look more carefully into usability and security challenges of secure messaging applications.

Acknowledgements

This thesis concludes my Master's Degree in Informatics: Programming and Networks with the Department of Informatics at the University of Oslo. It took two years of work to write this thesis, and I would like to acknowledge the people who have helped me, both directly and indirectly.

First of all, I would like to thank my supervisor, Christian Johansen, for his great help and advice when I needed it the most. Your guidance truly helped me complete my thesis, by always staying positive, pushing me to write when my motivation was not great, and giving me constructive feedback throughout this thesis.

The backing of both family and friends has given me the motivation to finish and never give up when writing. Most of all my mother and father who came to Norway in 1992, by leaving their families in Kosovo, to give my sister and brother a better life away from the war and terror. To be born in Norway and raised in two cultures has been hard, but the two of you have always encouraged and given me advice on how to live with them both in unison. I thank you for all the hard decisions you had to overcome for us; it truly means everything. My big brother, Ermal, thank you for all the support throughout my years at the University, for reading through this thesis and giving me valuable advice, and for inspiring me to become both a Computer Science student and a person who values security highly. Lastly, my big sister, Elira, thank you for the endless support you have given me, for making me forget about the thesis at times when I needed it.

My friends and fellow students, thank you for the last five incredible years at the University of Oslo. Thank you for all the memorable and fun years; especially all the happenings in our room Assembler and the semester abroad in San Jose, USA.

I would also like to thank my friends outside of school who have been patient with me while I have been writing this thesis.

I hope you as a reader enjoy this thesis and it offers you some insight into the field of secure mobile conversations.

Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, the trend to use mobile applications for communication has grown and become the standard method of communicating. New messaging applications started to emerge and try to replace traditional SMS, but building them with security and privacy in mind was not important for the developers. The popular messaging tools used in recent years do not support end-to-end encryption, only standard client to server encryption, which gives the service providers access to more private information than necessary.

When Edward Snowden published the secret papers about NSA, people finally understood that mass surveillance was an issue, and secure mobile messengers became more critical and popular. Instant Messaging clients who did not provide asynchronous communication became a problem because of the rise of smartphones and applications that were not always online. Secure messaging protocols such as Off-the-Record did not support asynchronous messaging, which gave other researchers and developers the motivation to develop new protocols for asynchronous communication in mind. After the Snowden revelations, new secure messaging applications started to emerge such as the Signal application with their protocol, Signal. After a while, the new protocol became quite popular among developers and researchers because of their secure messaging capabilities, and then it started to get implemented in other applications that only supported client-to-server encryption.

End-users have started to use secure mobile messaging applications, but it has become crucial to give the users a good user experience with easy to understand usability properties. While the user experience is essential for applications to have, which attracts new users to them, the usability aspects have been the vital part of the applications, which might have overshadowed the security aspects. If the developers do not take the security seriously, the end-users are the ones at risk because the conversations may get attacked by adversaries or impersonators could get access to the conversations.

New secure mobile applications are easy to use by end-users, where they only need a phone number to create an account, while the applications generate and exchange the cryptographic keys in the background, without the user having to do any interaction. The users would never know about the different cryptographic keys in use, and this is useful for the layman because they get seamlessly end-to-end encryption without having to understand the background on how the keys work.

1.1 Background and Motivation

Throughout the last couple of years, the need for secure messaging protocols has become apparent. People are more prone to understand the security implications that mass surveillance is not something which should be taken lightly. Edward Snowden has sparked a debate throughout the world that our individual privacy is not private anymore because of the mass surveillance multiple countries have been doing for decades[1].

No need to look further than the first quarter of 2017, when WikiLeaks[2] leaked documents from the U.S. Central Intelligence Agency (CIA). The leak, codenamed "Vault 7" by WikiLeaks is the largest ever publication of confidential documents on the agency [15]. The documents that leaked has information on how to get access to mobile phones or personal computers, without the users knowledge, and how the CIA did their fair share of mass surveillance.

There was a rise in startups that offered encryption and user privacy options after the Snowden scandal. Multiple companies started implementing secure messaging protocols and applications to counter the mass surveillance and offer an end-to-end encrypted messaging system which does not leak any information about the user's message content. There was a need for these types of applications and protocols, but the problem with new and bleeding-edge applications is the adoption of it. After a while, companies such as Google, Facebook, and Open Whisper Systems united to implement protocols into already wide adopted applications such as WhatsApp, which has over one billion monthly active users [49].

There are a lot of new secure messaging applications in 2017 that offer end-to-end encrypted message conversations over mobile phones and computers, but they often sacrifice usability aspects for security, which normally is a good thing, but it should be possible to have the best of both worlds. Applications should give the users enough information for them to know when or if a conversation is not secure anymore and the options to secure it once again.

Another point to be made is to show how the secure messaging protocols have evolved since the last time researchers went through the protocols which implement end-to-end message encryption. Unger et al [53] did a thorough analysis of the various secure protocols two years ago, but since then the protocols have evolved and implemented new security properties, and new protocols have surfaced.

[1]`https://en.wikipedia.org/wiki/List_of_government_mass_surveillance_projects`

[2]`https://wikileaks.org/`

1.2 Problem Statement

The area of end-to-end encryption in secure messaging protocols and implementations has become rather broad recently with various existing protocols. It is difficult for a user to find digestible and easy to understand information sources, and even less when it comes to comparative integrated studies. Therefore, a first research question is:

- Provide a detailed, yet comprehensible and comparative study of relevant approaches to end-to-end encrypted messaging technologies?

This study should include both old protocols, but more importantly the new ones which are also implemented in existing applications.

A comprehensive analysis of the security and privacy properties provided by secure messaging protocols is not easy and there are very few such studies (which we build upon). End-to-end encrypted messaging should be both usable for the user to adopt them, but also have usually very strong property requirements. These two, usability and security, are usually conflicting, and a good balance is difficult to find. This leads us to the second research question:

- What are the security and privacy properties provided by current end-to-end messaging protocols and applications, and to what extent these achieve these properties?

The Signal application (and protocol) is one of the most used end-to-end messaging technology currently available for smart phones and desktop PCs. This is why maybe focusing the above evaluations on Signal would already be useful. Moreover, the Signal protocol is employing state of the art technology in encryption and key establishment.

- What are the security mechanisms behind the Signal protocol?

1.3 Scope and Limitations

All the tests done with the mobile applications are limited to only the usability aspect of the application. The tests do not look at how the application is implemented from the bottom up, or at the source code. The test only checks what the user is shown during the test scenarios from the registration phase, key changes within a conversation and the usability of the verification process of users.

The use-case tests are performed the same on each of the different applications and are done by myself and not a group of participants in a case study because of limitations of finding enough participants. The test phones used were Android phones with the original operating system installed, and are not rooted to get access to information a normal user would not have the opportunity to get.

1.4 Research Methodology

In 2002 Crnkovic [14] studied *The Scientific Method in Computer Science* and identified three different scientific methods that could be applied to Computer Science research papers:

- **Theoretical Method:** This method adheres to the traditions of logic and mathematics. It follows the very classical methodology of building theories as logical systems with stringent definitions of objects and operations for deriving or proving theorems [14]. In computer science, for example, this can be adopted when working on the design and algorithms analysis to find new solutions or solutions to performance issues.

- **Experimental Method:** This method is used when doing experiments to extract results from real-world implementations, such as to test theories or do explorations to obtain new knowledge [20]. Experimental approaches are also useful in methods which take the human factor into consideration when gaining results from the experiments.

- **Simulation Method:** This method is about investigations beyond current experimental capabilities, such as study phenomena that cannot be replicated in laboratories [14].

In this thesis, the experimental method is the chosen method for this research, where experiments are conducted on real-world implementations of secure messaging application. The discussion of the different security properties is done in Chapter 2, to gain enough knowledge about security properties and then go into detail about the requirements for the Signal protocol in Chapter 3. Chapter 5 is about the design of our testing methodology and experiment, and step by step on what each of the application do for each test scenario that is laid out. The evaluation of the experiment is done in chapter 6 and 7, where the result of each experiment is analyzed and then discussed different properties the applications achieve and what should be done to obtain the missing properties.

1.5 Related Work

There has not been much research done on the usability and security area for secure messaging applications. In the last couple of years, it has become more and more important for researchers to look at the usability and not purely on the technical issues surrounding secure messengers. The first research paper that looked at the usability issues for end-to-end encrypted messengers is the paper by Schroder et al. [51] where they did a comprehensive user study on the usability of Signal's security features and came with recommendations on how to fix the issues it had with users failing to detect and deter man-in-the- middle attacks. This thesis does not use participants, but it does look at the same types of potential attack spots as Schroder et al., but with additional applications.

Unger et al. [53] did a more comprehensive work on secure messaging by looking at the security properties around trust establishment, conversation security, and transport privacy. Their survey shows that protocols that specialize in encryption do not manage to provide every important security and privacy property. In this thesis, the properties are based on the same properties that Unger et al. goes through, but only on the conversation security for three specific end-to-end encrypted messaging protocols.

1.6 Main Contribution

In section 1.1 about *Background and Motivation* two main motivations of writing this thesis was given. One is to update the overview on conversation security from Unger et al. [53] and to look at how secure messaging protocols have evolved since the release of that research paper. The other was to look at how secure messaging applications implement security and usability properties and to see if the applications have managed to keep the best of both worlds.

- The first research question is answered throughout Chapter 2 and 3 where we give users a thorough explanation of how the Off-the-Record and the Signal Protocol work.
- The thesis has compiled an overview of security and privacy properties for some secure messaging protocols such as OTR, Signal, and Matrix. The information is presented in a digestible way, such that future researchers can get a basic information source of the different protocols on how they work and why they provide some of the properties.
- Secure messaging applications have been tested in various test scenarios to see how they handle changes to cryptographic keys and how participants verify each other. Recommendations for improvement of the applications has also been given in the end.

1.7 Outline

The rest of the thesis is organized as follows:

Chapter 2 details the technical background of the thesis. There will be information about basic security principles and concepts and an overview of Off-the-Record Protocol.

Chapter 3 explains how the Signal Protocol is made up of, by both a Double Ratchet algorithm and a 3-Diffie-Hellman Key Agreement Protocol.

Chapter 4 present a systematization of knowledge about three secure end-to-end encrypted messaging protocols, with a discussion of the information retrieved in the end. *The second research question is answered.*

Chapter 5 presents the research about six mobile phone applications which support a few of the protocols shown in chapter 4, and some others which are not open source applications.

Chapter 6 presents the results from each test scenario of the various test cases conducted from the research in chapter 5.

Chapter 7 presents the discussion of the results presented in the previous chapter. *The third research question is answered.*

Finally, Chapter 8 presents the conclusions that can be drawn from the different results in the earlier chapters. In the end it will give a summary of this thesis and list our thoughts for future work.

The research questions described earlier are answered by the following chapters. *Question 1* is answered through the study in Chapter 2 and 4; *Question 2* is answered through the study in Chapter 4 and 5; whereas Chapter 6 and 7 present the results from the tests that were done and the proposed recommendations; *Question 3* is answered mainly through the study in Chapter 3, but also all other chapters detail other tangent aspects of Signal.

1.8 Summary

This chapter first gave a brief introduction to this master thesis, discussing the recent years of mass surveillance by the governments and what has been the consequences of this in the area of secure messaging.

Following this was a section regarding the background and motivation of the thesis. The section explains more about the surveillance by our government, and the revelations of both Edward Snowden scandals and the recent leaks by Wikileaks about Vault 7.

WThe problem statement of the thesis was given, explaining that this thesis could act as a compendium, by providing an overview of all information about different secure messaging protocols, and at the same time as an information source to see how the different applications handle changes of cryptographic keys and other usability aspects.

The scope and limitation explained that the applications are not tested on how they are implemented from the bottom up, only on the usability properties they provide, and that the tests are performed only by me and not by a group of participants.

The next chapter goes into the technical background of this these, explaining the different security principles and the Off-the-Record Protocol.

Chapter 2

Technical Background

2.1 Encryption

Encryption can be intuitively understood as a method for converting *plain text* (or other form of information) into *encrypted text* that is difficult (impossible) to understand by anyone except authorized parties (i.e., parties holding a decryption key). The most important application of encryption is to protect the information stored on computers or transmitted on the Internet. Encrypted messages do not prevent interception, but it does deny the content that is inside the encrypted message, such that the interceptor has no way of knowing what the decrypted message contains. The only way an interceptor (which we usually call an *eavesdropper*) can read the encrypted message is by having the key to decrypt the message, if not the eavesdropper would see only useless (unintelligible) gibberish.

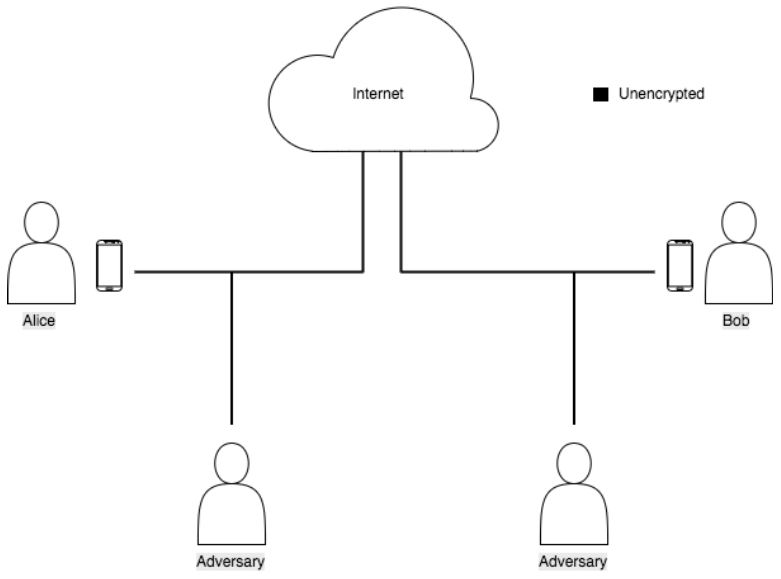


Figure 2.1: Unencrypted messages between Alice and Bob [19]

This thesis is about end-to-end encryption of messages between Alice and Bob, but to fully understand how this can and cannot protect them, it is important to explain how an unencrypted message system works.

Alice and Bob want to discuss between each other and do not care what type of messaging application they use or if it is secure. Messages passed between Alice and Bob without encryption, move from their devices to their local network and then across some amount of internet devices until it reaches the received local network and then to the appropriate device. Since they send unencrypted messages, eavesdroppers can read the message at any point of the transit [19].

The next step from unencrypted messages is to encrypt them from Alice to the server, decrypt them there, and then encrypt from the server to Bob. This communication encryption is Transport Layer Security (TLS)[12] since it is designed to secure communications between a client and server.

Messages sent to the server are decrypted by the server, which means that it can openly read, store or edit the message before encrypting them again and sending it to the other end. The problem with this is the possibility of the server being attacked by an adversary, even though the clients trust the server to handle the packages. At the same time, the servers may be contacted by law enforcement to give them the information sent by clients to the server, and there is nothing the clients can do if the law enforcement has legal jurisdiction [19].

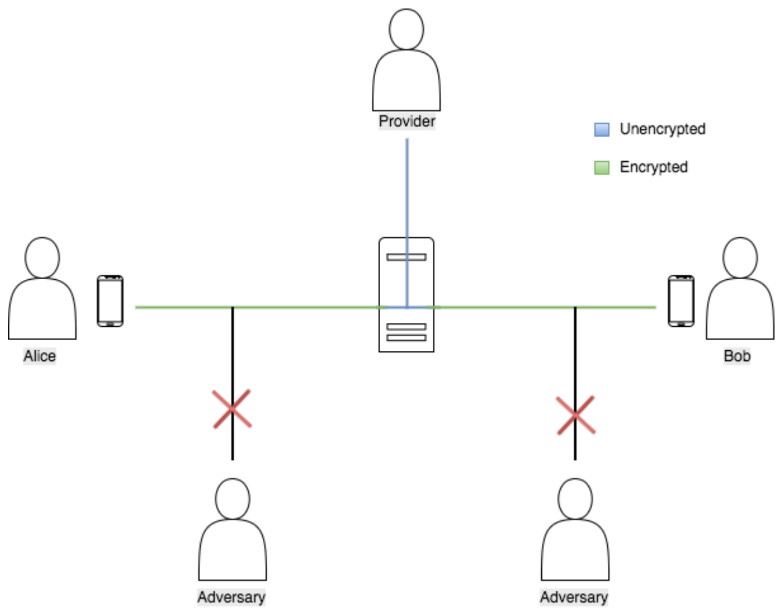


Figure 2.2: TLS Encryption between Alice and Bob [19]

The last change to the encryption is the end-to-end encryption implementation. If implemented, then the endpoints on each of the client's applications do the encryption while the servers only transmit the messages without knowing the content. Figure 2.3 shows how the end-to-end encryption works by encrypting Alice's and Bob's application endpoints and then sending it through the servers, where neither, the attackers between client and server, nor the server can get access to the messages.

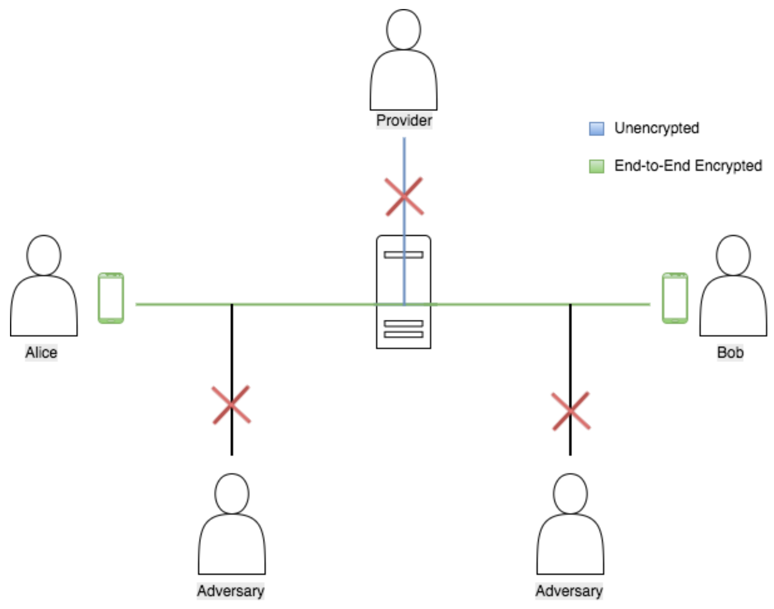


Figure 2.3: End-to-End encryption between Alice and Bob [19]

2.2 Security Principles, Usability and Adoption

This sections talks about the different security principles that a secure messaging system should have for it to be called secure. These have a fundamental part throughout the thesis on how to compare the specifications of different protocols that advertise secure messaging applications. There will be talk about the most important principles such as authentication or forward secrecy, but also smaller ones that are equally important to have for secure messaging such as out-of-order resilient and asynchronicity. Group chats are important for the end-user, this means that the different protocols should also have additional features to support this which is discussed more at the end of this section.. The security and privacy features that are explained and based on, if nothing else is specified, are from the research paper by Nik Unger et al [53].

2.2.1 Authentication

Authentication is about identifying individuals within a conversation, to confirm that the person is who they say they are. If one user sends a message to another user, and the message is modified en route to the receiver, then the receiver is assured of detecting the modification. Message authentication is also called *data-origin authentication*. Message authentication protects the integrity of a message, to ensure that each message which is received is in the same condition that it was sent out, with no modification done to the message [6].

Message authentication codes (MAC)[1] provides assurance about the source and integrity of a message. A message authentication code is computed by using the message and a shared secret between two parties. Which means that to authenticate a message, the receiver has to share the secret key used to compute the message authentication code with the sender [25]. An adversary cannot validate the message because only the sender and receiver have the shared secret, and if the adversary changes the message, then the computed MAC key changes as well.

2.2.2 Perfect Forward Secrecy

The idea of forward secrecy is that when a long-term key is compromised, sessions keys that were previously established using that long-term key should not be compromised. Typical example of protocols which provide forward secrecy is key agreement protocols where the long-term key is only used to authenticate the exchange. Key transport protocols in which the long-term key is used to encrypt the session key cannot provide forward secrecy [9].

Definition 2.1. A key establishment protocol provides forward secrecy if compromise of the long-term keys of a set of principals does not compromise the session keys established in previous protocol runs involving those principals.

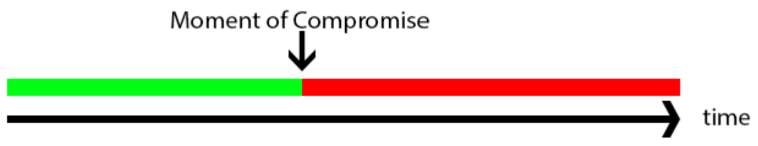


Figure 2.4: Forward Secrecy

2.2.3 Future Secrecy

Backward secrecy or Future secrecy, as it has been coined by the Open Whisper Systems [34], is when a protocol can guarantee that compromise on long-term keys does not allow subsequent ciphertexts to be decrypted by passive adversaries [53]. A protocol does also support future secrecy when it can provide the ”self-healing” aspect of the Diffie-Hellman ratchet, which will be described in section 2.4, because if any ephemeral key is compromised or found to be weak at any time, the ratchet will heal itself and compute new ephemeral keys for the rest of the messages sent during the conversation [34].

[1] `https://en.wikipedia.org/wiki/Message_authentication_code`

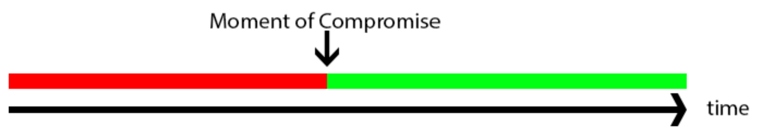


Figure 2.5: Backward Secrecy

2.2.4 Deniability

Deniability is a property common to new secure messaging protocols, where it is not possible for others to confirm that the data was sent by one particular person. If Bob receives a message from Alice, he can be sure it was Alice that sent it, but cannot prove to anyone else that it was Alice who wrote it. Secure messaging protocols which offer deniability can assure the user that anyone can forge messages after a conversation to make them look like they came from them, but if it is during a conversation, the participants are assured that the words they see are authentic and are not modified by anyone else other than themselves [22].

2.2.5 Synchronicity

There are two types of communication, synchronous and asynchronous, and a chat protocol can be one of those two. Synchronous protocols have a requirement that all participants must be online for them to receive or send messages. If a chat protocol is asynchronous, it means that the participants do not need to be online to receive messages, such as SMS text messaging or emails, since there is a third party in play to save the information until the recipient gets online again.

When it comes to today's chat protocols, it is advised against using a synchronous protocol. The reason is that there are social and technical constraints, such as device battery, limited reception or other social happenings which mean people will have problems always being online to receive messages. That is why the majority of Instant Messaging (IM) solutions provide asynchronous environment by having a third party server which stores the messages until the other participant gets online to receive it. The new secure messaging protocols should by design be asynchronous platform for communication for it to become popular amongst the end-users.

2.2.6 Confidentiality

Confidentiality ensures that the necessary level of secrecy is enforced at each junction of data processing and prevents unauthorized disclosure [26]. This is usually achieved by encrypting the data from the sender to the receiver, and only those with the correct decryption key can read what the encrypted data contains. In cryptographic protocols confidentiality is essential to ensure that keys and other data are available only as intended [9].

Attackers have the opportunity to ruin confidentiality mechanism by stealing password files, breaking encryption schemes, or by social engineering. Users, on the other hand, can intentionally or accidentally disclose sensitive information by not encrypting it before sending it to another person, or by falling prey to a social engineering attack [26].

Confidentiality can be provided by encrypting data as it is stored and transmitted, enforcing strict access control and data classification, and by training personnel on the proper data protection procedures [26].

2.2.7 Integrity

Integrity is having the assurance that anyone throughout the transmission does not modify the messages and its content. Any honest party should not under any circumstances accept a message that is modified. Hardware, software, and communication mechanisms must work in concert to maintain and process data correctly and to move data to intended destinations without unexpected alteration. The systems and network should be protected from outside interference and contamination [26].

Environments that enforce and provide this attribute of security ensure that attackers, or mistakes by users, do not compromise the integrity of systems or data [26]. This can be achieved through the use of hash functions[2] in combination with encryption, or by use of a message authentication code (MAC) to create a separate check field. Data integrity is a form of integrity that is essential for most cryptographic protocols to protect elements such as identity field or nonces[9].

2.2.8 Other Security Properties

These security properties are smaller and more concise than the other properties, but are still as important to implement into the end-to-end secure messaging protocols.

- **Participant Consistency:** At any point when a message is accepted by an honest party, all honest parties are guaranteed to have the same view of the participant list.
- **Destination Validation:** When a message is accepted by an honest party, they can verify that they were included in the set of intended recipients for the message.
- **Anonymity Preserving:** Any anonymity features provided by the underlying transport privacy architecture are not undermined (e.g., if the transport privacy system provides anonymity, the conversation security level does not deanonymize users by linking key identifies).

[2]`https://en.wikipedia.org/wiki/Hash_function`

- **Speaker Consistency:** All participants agree on the sequence of messages sent by each participant. A protocol might perform consistency checks on blocks of messages during the protocol, or after every message is sent.
- **Causality Preserving:** Implementations can avoid displaying a message before messages that causally precede it.
- **Global Transcript:** All participants see all messages in the same order. When this security feature is assured, then it implies both speaker consistency and causality preserving are assured.
- **Deniability** can be divided into three different parts:
  - **Message Unlinkability:** If a judge is convinced that a participant authored one message in the conversation, it does not provide evidence that they authored other messages.
  - **Message Repudiation:** Given a conversation transcript and all cryptographic keys, there is no evidence that a given message was authored by any particular user. We assume the accuser has access to the session keys,but not the participants long-term secret keys.
  - **Participation Repudiation:** Given a conversation transcript and all cryptographic key material for all but one accused (honest) participant, there is no evidence that the honest participant was in a conversation with any of the other participants.

2.2.9 Group Chat

In recent years it has become easier to have a group conversation among friends and colleagues with the use of Facebook Messenger[3], Slack[4] or other popular messaging applications out there[5]. The problem with these messaging applications is that they are not end-to-end encrypted, but, i.e., Open Whisper Systems have worked hard to make this easier for everyone by introducing fast and reliable end-to-end encrypted group chats. Therefore, it is important that the new protocols and applications have additional security properties and features to support end-to-end encryption.

- **Computational Equality:** Do the participants share an equal computational load when talking to each other.
- **Trust Equality:** There is not a single participant who has more trust or responsibility, within the group, than any other.
- **Subgroup messaging:** Participants can send messages to only a subgroup of others without generating a new conversation.

[3]`https://www.messenger.com`
[4]`https://slack.com`
[5]`https://www.engadget.com/2016/09/30/12-most-used-messaging-apps/`

- **Contractible membership:** The group do not need to restart the security protocol when a member leaves the conversation.
- **Expandable membership:** There is no need to restart the security protocol when adding a new member after the group has been generated.

It is important for the protocol to have the ability of changing cryptographic keys when a new user joins the secure group conversation, since then the new users to not have the ability to decrypt previously sent messages. New cryptographic keys should also be exchanged when a user leaves the conversation. The changing of keys is not that big of a problem, it is simply restarting the protocol, but this is often computationally expensive. Protocols which offer contractible and expandable membership achieve these features without restarting the protocol.

2.2.10 Usability and Adoption

Other usability aspects should be taken into account when looking at usability and adoption, but these are discussed in Chapter 5, where different UX decisions the developers have done are taken into consideration. This section is about other usability factors, such as resilience, multi-device support and if the protocol needs additional services.

- **Out-of-order resilience:** If a message is delayed in transit, but eventually arrives, its contents are accessible upon arrival.
- **Dropped Message Resilient:** Messages can be decrypted without receipt of all previous messages. This is desirable for asynchronous and unreliable network services.
- **Asynchronous:** Messages can be sent securely to devices which are not connected to the Internet at the time of sending.
- **Multi-Device Support:** A user can connect to the conversation from multiple devices at the same time, and have the same view of the conversation as the others.
- **No Additional Service:** The protocol does not require any infrastructure other than the protocol participants. Specifically, the protocol must not require additional servers for relaying messages or storing any kind of key material.

2.3 Basic Concepts

2.3.1 Diffie-Hellman Key Exchange

The *Diffie-Hellman key exchange* (DHKE), proposed by Whitfield Diffie and Martin Hellman in 1976 [13], was the first asymmetric scheme published in the open literature. It provides a practical solution to the key distribution problem, i.e., it enables two parties to derive a common secret key by communication over an insecure channel [45]. The elegant and simple construction has been the basis for a vast range of protocols [9], such as Secure Shell (SSH)[6], Transport Layer Security (TLS)[7], and Internet Protocol Security (IPSec)[8]. This section will have a simple explanation of Diffie-Hellman and a common vulnerability.

Diffie-Hellman consist of two protocols, one that is used to set-up the public parameters that Alice and Bob are going to use, and the primary protocol which performs the key exchange. The set-up protocol consist of the following steps [45]:

Diffie–Hellman Set-up

1. Choose a large prime p.
2. Choose an integer $\alpha \in \{2, 3, \dots, p-2\}$.
3. Publish p and α.

Figure 2.6: Diffie-Hellman Set-up [45]

Those two values that are devised from the set-up protocol are called *domain parameters*. If Alice and Bob both know the public parameters p and alpha, they can generate a shared secret key k with the following key-exchange protocol.

Diffie–Hellman Key Exchange

Alice		**Bob**
choose $a = k_{pr,A} \in \{2, \dots, p-2\}$		choose $b = k_{pr,B} \in \{2, \dots, p-2\}$
compute $A = k_{pub,A} \equiv \alpha^a \bmod p$		compute $B = k_{pub,B} \equiv \alpha^b \bmod p$
	$\xrightarrow{k_{pub,A}=A}$	
	$\xleftarrow{k_{pub,B}=B}$	
$k_{AB} = k_{pub,B}^{k_{pr,A}} \equiv B^a \bmod p$		$k_{AB} = k_{pub,A}^{k_{pr,B}} \equiv A^b \bmod p$

Figure 2.7: Diffie-Hellman Key Exchange [45]

On the other hand, the basic Diffie-Hellman Key protocol does not offer any authentication of the messages sent. The problem with no authentication is that anyone can exploit it by a *man-in-the-middle* attack. An attacker can insert itself between Alice and Bob's communication channel, and impersonate both Alice and Bob to the other party. When this happens, Alice will think she is doing a DHKE with Bob, and Bob thinks he

[6]https://en.wikipedia.org/wiki/Secure_Shell
[7]https://en.wikipedia.org/wiki/Transport_Layer_Security
[8]https://en.wikipedia.org/wiki/IPsec

is doing a DHKE with Alice. After the DHKE is done, they will get a shared secret with the attacker instead of each other. The figure below shows how the attacker C can attack the DHKE between A and B:

Attack on basic Diffie-Hellman

$$\begin{array}{lclcl}
\textbf{A} & & \textbf{C} & & \textbf{B} \\
r_A \in_R \mathbb{Z}_q & & & & \\
t_A = g^{r_A} & \xrightarrow{\quad t_A \quad} & r_C \in_R \mathbb{Z}_q & & \\
 & & t_C = g^{r_C} & \xrightarrow{\quad t_C \quad} & r_B \in_R \mathbb{Z}_q \\
 & \xleftarrow{\quad t_C \quad} & Z_{AC} = t_A^{r_C} & \xleftarrow{\quad t_B \quad} & t_B = g^{r_B} \\
Z_{AB} = t_C^{r_A} & & Z_{CB} = t_B^{r_C} & & Z_{AB} = t_C^{r_B}
\end{array}$$

Figure 2.8: Attack on basic Diffie-Hellman Key-Exchange [9]

2.3.2 Key Derivation Function

Key Derivation Function(KDF) is part of Key Establishment, which deals with establishing a shared secret between two or more parties. Key Establishment can be divided into two different methods, *key transport* and *key agreement* methods [45]. This section will only be about the key agreement method since the Signal Protocol uses it as part of its algorithm.

In most security systems it is desirable to use cryptographic keys which are only valid for a limited time. These keys are usually called *session keys* or *ephemeral keys*. The reason for limiting the period in which a cryptographic key is used is because it has an advantage, such as it does less damage if the key is exposed [45]. Another reason is that an attacker has less ciphertext to work with under each generated key.

It is important to update the key frequently, such that attackers can't decipher the ciphertext. This can be done by using an already established shared secret key to *derive* fresh session keys, by using a key derivation function. It works by using a non-secret parameter r to process together with the join secret k_{AB} between two users, Alice and Bob.

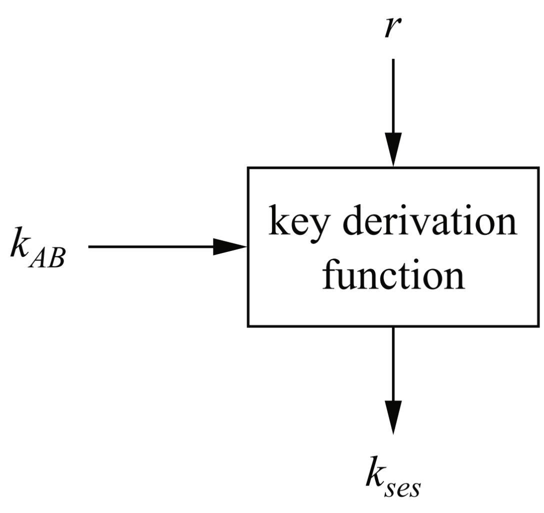


Figure 2.9: Principle of key derivation [45]

The key derivation function has a characteristic that it should be a one-way function. When the function has a one-way property, it will prevent an attacker from deducing k_{AB} should any of the session keys become compromised, which would allow the attacker to compute all future keys [45].

2.4 Off-the-Record Protocol

Before discussing the Off-The-Record protocol (OTR), is is important to understand how the general usage of Off-The-Record works. Take a scenario where Alice and Bob are alone in a room. Nobody can hear what they are saying to each other unless someone records them. No one knows what they talk about, unless Alice and Bob tell them, and no one can prove that what they said is true, not even themselves. This type of scenario and conversations are called "Off-The- Record." The good thing about an Off-the-Record conversation (in reality) is the legal support behind it since it is illegal to record conversations without participants knowing. It also applies to conversations over the phone, since by law, it is illegal to tap phone lines. There was nothing for communications over the web that could work until cypherpunks[9] released a new Off-the-Record protocol.

There are a couple of things such protocols need for it to work as an Off-the- Record protocol. Perfect forward secrecy is one of them, and repudiation is the second since without those there would be no possibility to deny what the participant has said in a past conversation. The OTR protocol addresses these issues.

Off-the-Record was designed to provide features for an underlying instant messaging protocol [23]. Instant messaging are a series of messages

[9]https://otr.cypherpunks.ca/

exchanged between two parties but could not exactly alternate, which is that a party could send multiple messages before he or she receives a reply.

OTR was developed in such a way that it uses the instant messaging protocol as a transport layer, this means that each binary message of the OTR protocol is converted to characters which the instant messaging protocol can transport over the web. The receiver then takes that message and converts it back to the original format. Each OTR message is marked with a unique tag so that it is easy to recognize which version of the OTR the sender uses.

The next steps go through the different steps that Off-the-Record protocol does and how they work. It first starts with the Off-the-Record Authenticated Key Exchange, then goes over to how the message transmission works. Step three is about the re-keying that OTR often does during the conversation between parties and in the end, the last step is the publishing of MAC-keys. After the four necessary steps, an important step is shown that ensures no passive or active attacker, such as Man-in-the-Middle, is possible during a conversation. When combining all these steps, we get a Diffie-Hellman ratchet which generates new cryptographic keys for each new message.

2.4.1 Step 1: Authenticated Key Exchange

Upon starting a conversation, both Alice and Bob need to know if they both want to have a conversation with OTR. This is done by one of them, which means that Alice informs Bob that she is willing to use the OTR protocol to speak with him. She sends Bob a Query Message requesting Bob to start an OTR conversation with her. She can also in the same message state which version of OTR she wants to use. In the end, it is up to Bob if he wants to speak with OTR or not; he is going to be the one starting the Authenticated Key Exchange if he agrees.

According to the site of Off-the-Record, cypherpunks, it does use a variation of Diffie-Hellman Key Exchanged called SIGMA. All exponentiations are done module a particular 1536-bit prime, and g is a generator of that group[23]. Alice and Bob do also have each of their long-term authentication public keys, which are pub_A and pub_B, respectively. The point here is to do an unauthenticated Diffie-Hellman key exchange to set up an encrypted channel, and inside that channel do mutual authentication.

Diffie-Hellman is used to find a way of generating shared secret between two parties, in a way that is not possible for others to compute the same shared secret. The key does not get shared during the exchange, but the two parties create the key together, which is an important distinction.

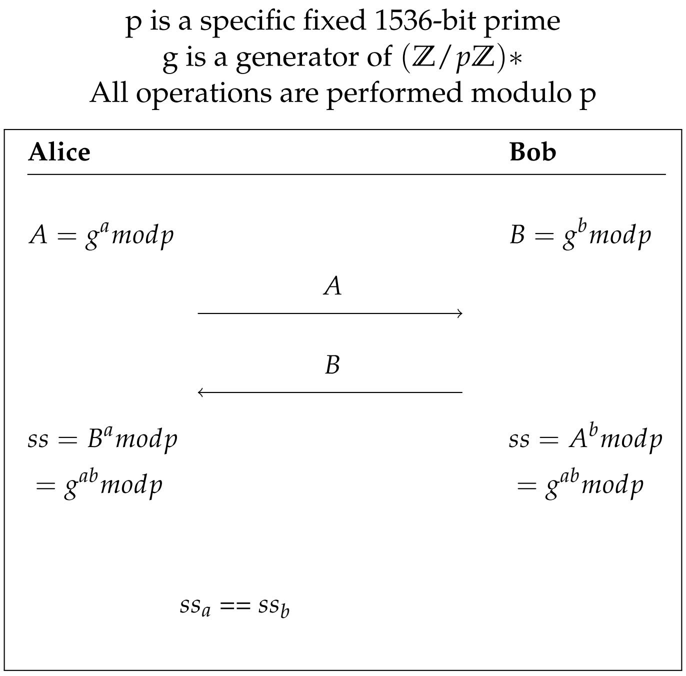


Figure 2.10: Plain Diffie-Hellman Key Exchange

Since both Alice and Bob have their 1536-bit prime number and a generator of that group, they can start doing they Diffie-Hellman Key Exchange. They agree on these number publicly, so that they are the only ones who know them, but they do not need to be protected. Alice then picks a random number only she knows, computes $g^a mod p$ and sends the number to Bob. Bob on the other side does the same, picks a random number, computes $g^b mod p$ and sends it to Alice. When both Alice and Bob have each others computed numbers, they calculate the common secret. Alice calculates $g^{ab} mod p$ and Bob calculates $g^{ba} mod p$, and if the number is the same on both sides, they know that by using this number, their messages are secure.

The plain Diffie-Hellman key exchange is vulnerable to active attacks, such as man-in-the-middle. If Eve is in the middle of the key exchange, she can take the message from Alice and send her key instead to Bob. Bob would not know the difference; he is thinking that he is exchanging with Alice, but is talking with Eve. The established key is equal for both sessions, but while Alice associates it with Bob, he, on the other hand, associates it with Eve [11]. This authentication failure would happen if OTR did only use plain Diffie-Hellman, which they did in the beginning, but have now implemented a signature-based authenticated DH exchange, named SIGMA, which solves this weakness [32].

The SIGMA acronym is short for "SIGn-and-MAc," because SIGMA decouples the authentication of the DH exponentials from the binding of key and identities. The former authentication task is performed using digital signatures while the latter is done by computing a MAC function keyed via g^{ab} and applied to the sender's identity [32].

SIGMA has a few different forms, from a basic form where it does not provide identity protection, to a four message variant known as SIGMA-R which OTR uses since it provides both defenses to the responder's identity against active attack and to the initiator's passive attacks. The basic form is first shown to give a better understanding of the SIGMA protocol

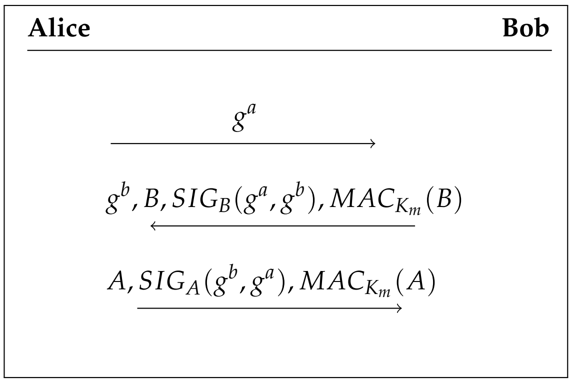

Alice
Bob
g^a
$g^b, B, SIG_B(g^a, g^b), MAC_{K_m}(B)$
$A, SIG_A(g^b, g^a), MAC_{K_m}(A)$

Figure 2.11: Basic Form of SIGMA [32]

The output of the protocol is a session key K_s derived from the Diffie-Hellman value g^{ab}, while the K_s is used as a MAC key in the protocol derived from the same Diffie-Hellman value [32]. It is important that those keys are computationally independent (information from one of the keys can not be learned form the other key, or vice versa).

There are two fundamental elements in the logic of the protocol. The first one is that the Diffie-Hellman exponential chosen by two parties is protected from modification by the attacker via the signature that the party applies to its exponential. The second one is the MACing of the sender's identity under a key derived from the Diffie-Hellman key. The second one is the functionality to bind the session key to the identity of each of the protocol participants in such a way that it ensures the "consistency" of Key-Exchange Protocols [32].

The other form of SIGMA, shown in figure 2.12, which OTR uses, is the four message variant called SIGMA-R without encryption:

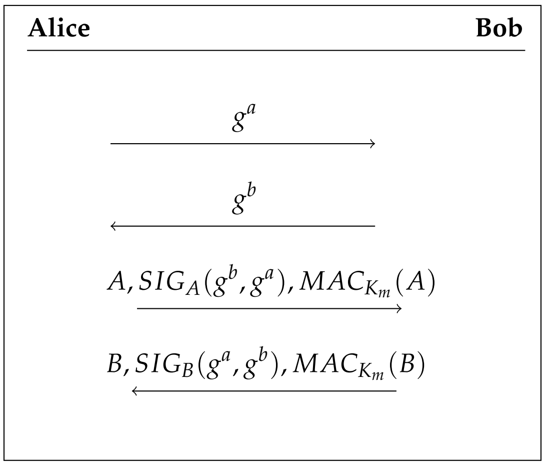

Alice
Bob
g^a
g^b
$A, SIG_A(g^b, g^a), MAC_{K_m}(A)$
$B, SIG_B(g^a, g^b), MAC_{K_m}(B)$

Figure 2.12: Four Message Variant: SIGMA-R [32]

The difference between SIGMA-R and the basic form of SIGMA, is that Bob actually delays sending his identity and authentication information for the fourth message, after it has verified Alice's identity and authentication from the third message.

There is a problem with this four-way message version of SIGMA that it is vulnerable to reflection attacks. An attacker can simply replay the message back to Alice, as long as Alice is willing to accept a key exchange with itself. Alice's defense against this is to ensure that the messages have a sense of direction, which means to add different tags to the MAC for each party, or using different MAC keys for each direction [32]. By implementing these measures, SIGMA-R can ensure that the attacks are correctly prevented from happening. When the exchange between Alice and Bob with the SIGMA protocol runs successfully, then they know each other's Diffie-Hellman public keys, and share a secret value.

On the first step of the Off-the-Record protocol, Alice and Bob pick a random x and y, respectively. Alice sends g^x to Bob, and then Bob sends to Alice g^y. Bob then waits until Alice has sent him the signature step from SIGMA-R to ensure there are no reflection attacks against the two parties. When Alice sends to Bob her $SIG_A(g^y, g^x)$ and $MAC_{Km}(A)$, then Bob sends the same but his MACs his key instead. In the end, they both compute the shared secret $SS = g^{xy}$.

2.4.2 Step 2: Message Transmission

Message transmission is where the encryption and authentication of messages happen before sending them over the web. The Off-the-Record Protocol uses AES [42] as its encryption protocol, using it in counter mode [18]. The message is first encrypted using the AES in counter mode; then the resulting ciphertext is authenticated using a keyed-hash message authentication code (HMAC) [5].

The reason the Off-the-Record team chose AES in counter mode was the desire of having malleable encryption scheme because that increases deniability. Malleability is only a property of some cryptographic algorithms. The basic of it is that an algorithm is malleable if it is possible to transform a ciphertext into another ciphertext which then decrypts to a related plaintext [11], which gives us increased deniability. This means that a valid ciphertext can not be connected with either Alice or Bob since anyone can create a ciphertext that can be decrypted correctly and then compute a valid MAC from the ciphertext, because old MAC keys are published to the web (more about this in step 4).

The Advanced Encryption Standard (AES) [42] is a block cipher which can also be used as a stream cipher, which the counter mode does. There are multiple modes of operation with AES, which encrypts and sends block sizes of data, but all the modes have one goal; Encrypt data and provide confidentiality for messages sent from Alice to Bob [46].

The counter mode is a confidentiality mode that features the application of the forward cipher to a set of input blocks, called counters, to produce a sequence of output blocks that are XORed (Exclusive-ORed)[10] with the plaintext to produce the ciphertext, and vice versa [18]. Each block in the sequence of ciphers, need to be different from every other block. This is not restricted to a single message, but across to all of the messages that are encrypted under the given key [18].

To produce the ciphertext, during the encryption, the cipher function is invoked on each counter block, as explained before, then the resulting output is XORed with the corresponding plaintext block to get the ciphertext. The same thing is done in the decryption but XORed with the ciphertext to get the plaintext. What is important to understand, is that this can be done in parallel on both sides, which means that the ciphertext block can be independently decrypted and does not have to wait for the ciphertext block which is before that one [18].

[10]`https://en.wikipedia.org/wiki/Exclusive_or`

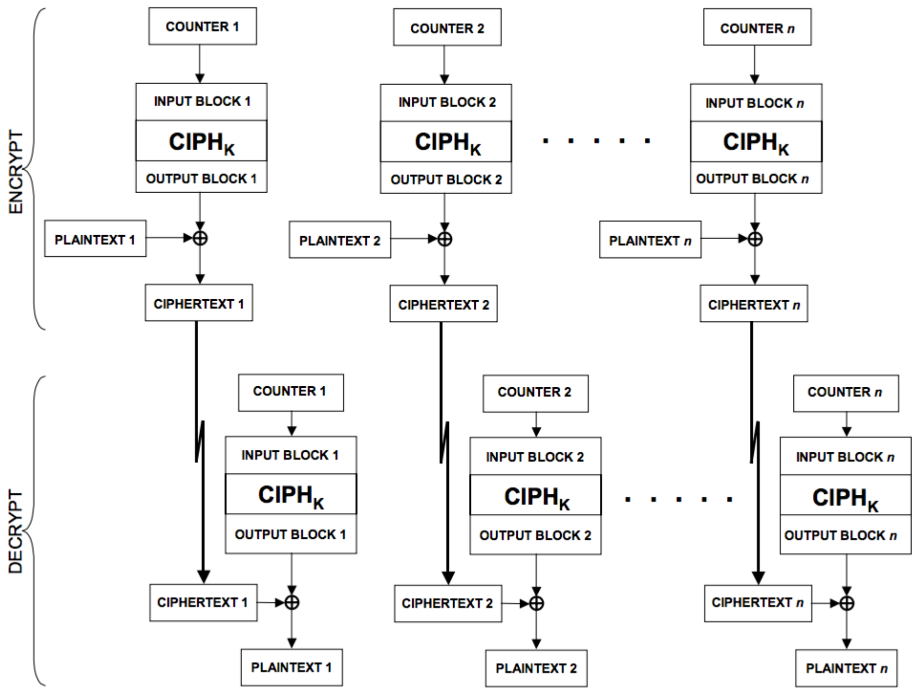


Figure 2.13: AES in CTR Mode [18]

During message transmission Bob cannot prove to Charlie that the messages he gets are coming from Alice, even though he is assured of this after the fact: AES counter mode allows messages to be altered once the MAC key is published. Entire new messages, or full transcripts, can, in fact, be forged.

When Alice wants to send a message to Bob, she needs to compute an *Encryption Key* and a *MAC Key*. The encryption key is used to encrypt the message while the MAC key is used to ensure the authenticity of the message. This method is called encrypt-then-mac approach, where the encryption key is a hash of the shared secret, $EK = Hash(SS)$, and then the encryption key is hashed another time to compute a MAC key $(MK = Hash(EK))$.

After the encryption and MAC key are computed, Alice encrypts first the message, $Enc_{EK}(M)$ and then MACs the encrypted message, $MAC(Enc_{EK}(M), MK)$. The Enc is the symmetric encryption shown earlier in this step, AES in counter mode. When Bob receives the message from Alice, he first needs to compute his own EK and MK from the same SS that both Alice and Bob share. After the computation is done, he verifies the MAC using his MK and then decrypts the message with his EK.

2.4.3 Step 3: Re-key

Step three is about re-keying as often as possible. Alice and Bob need to pick a new x′, y′, and then do a new Diffie-Hellman exchange. The way Off-the-Record does it today, is to change the keys every time the conversation changes directions, this makes the duration of vulnerability to attack as short as possible.

Once the new key is established it will be used to encrypt and authenticate new messages, while the previous ones are erased [11]. After Alice and Bob got themselves the new x′, y′, ss′ and EK′, they erase the old ss, x, y, and EK, such that no one can forge or decrypt the messages that have been sent. The reason to securely erase this information is to get perfect forward secrecy in our instant message conversations. The MK key does not get erased, and the reason for this is discussed in step four.

2.4.4 Step 4: Publish MK

The next step of OTR is to publish the MAC key. Alice and Bob do not need to forget since they both know that they have moved over to MK′, hence if one of them gets a message with the old MK, they will know that the message has been forged. The old MAC keys are added to the next message that Alice or Bob sends to each other, in plaintext, since they do not care if they are readable

The reason for publishing it is because by doing this they will let other people to forge transcripts of conversations between Alice and Bob. This is useful since it provides extra deniability to both of the parties. The reason for this is if someone goes to Alice and says they have a transcript of a conversation between her and Bob saying something incriminating, Alice can turn around the conversation and show the other person a transcript of them doing something incriminating. None of them have significant proof that the transcripts where real or not, since Alice and Bob have booth published their MK.

One could say that Alice's deniability relies on Bob's deleting the MK, but this is not the case since Alice' secrecy relies on Bob deleting the key information, but Alice's deniability relies only on Alice publishing her MK's after she is done using them. This way no one can confirm that she is the one behind the transcripts.

2.4.5 Socialist Millionaire Protocol

The problem with secure instant messaging is that there is no way to tell if there has been a Man-In-The-Middle attack. The Diffie-Hellman Key Exchange does not have any protection (authentication) if the exchange has been changed in between by Oscar, that is why OTR implements the Socialist Millionaire Protocol (SMP).

Look at it like two millionaires want to exchange information to see whether they happen to be equally rich, without revealing anything about the fortunes themselves [1].

This is what Socialist Millionaire Protocol does between Alice and Bob, but they don't want to check if they are equally rich, but they wish to know whether $x = y$, where Alice and Bob have x and y respectively. Socialist Millionaires Protocol allows Alice and Bob to compare $x == y$ without revealing any other information between them [23]. With our messaging protocol, OTR, the secret information between them are the long-term authentication public keys, and also information entered by themselves.

p is a specific fixed 1536-bit prime
g is a generator of $(\mathbb{Z}/p\mathbb{Z})*$
All operations are performed modulo p

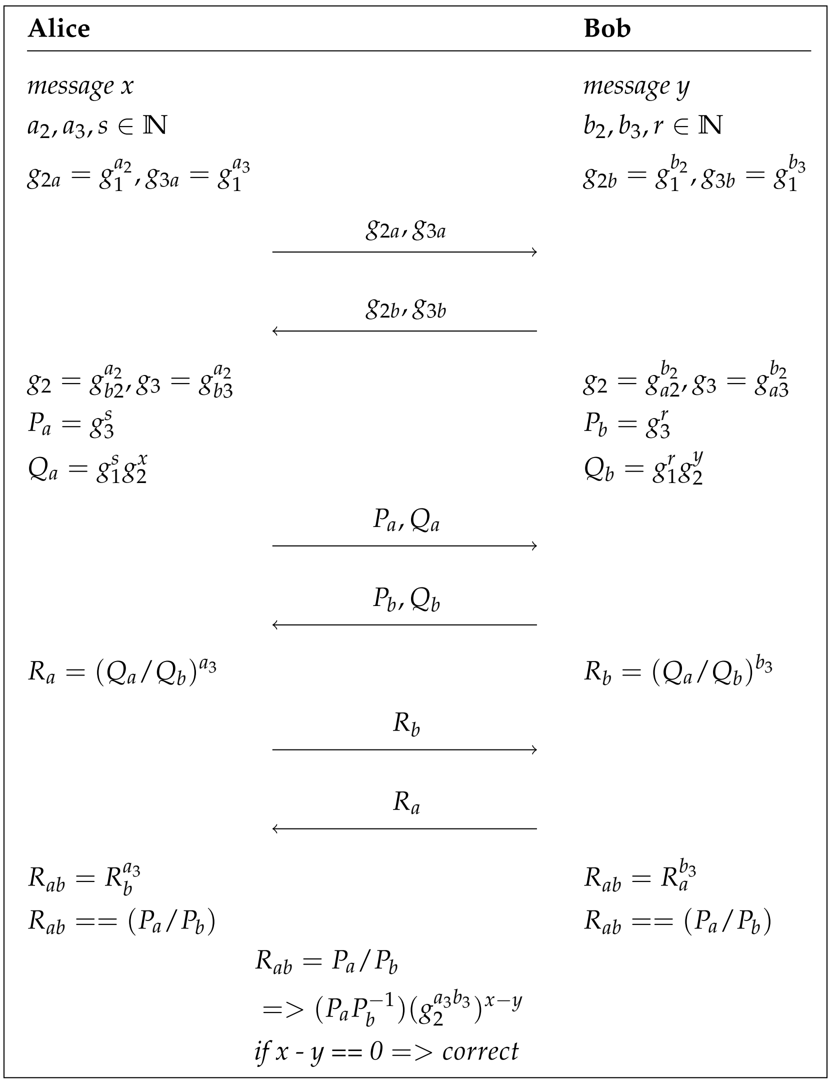


Figure 2.14: Socialist Millionaire Protocol (SMP)

Let us go through the figure 2.14 step by step. Both Alice and Bob pick random exponents a_2, a_3, s and b_2, b_3, r respectively.Alice sends both g_{2a} and g_{3a} to Bob, and he sends g_{2b} and g_{3b} to Alice. They then calculate their g_2

and g_3 on each side by using what they got back from the other side.

The next step is to compute both P and Q on both sides and send them to each other. When computing the Q on both sides, they use their shared secret which they want to check and see that they are the same shared secret. Alice uses message x, and Bob uses message y in the computation.

When they get each others Q and P, they hold off on using the P in the computations until later, but use Q to compute R respectively. After they are done computing R on their side, they send it to the other party.

The last computation before checking if they have the same shared secret is to compute $R_a b$. In the end both need check whether $R_a b == (P_a/P_b)$. If everything is done correctly, then $R_a b$ should hold the value of (P_a/P_b) times $(g_2^{a_3 b_3})^{(x-y)}$. This means that the test at the end of the protocol will only succeed if $x == y$ or $x - y$ is 0. Since $g_2^{a_3 b_3}$ is a random number not know to any party if x is not equal to y, no other information is revealed, and the conversation is terminated.

This is the Socialist Millionaire Protocol; it solves the problem of checking if both parties are equally rich, using a shared secret only Alice and Bob know of, while Oscar does not know this secret. They use the SMP algorithm to verify that the other party knows the same shared secret, and the job of SMP is to confirm this without revealing anything other. In the end, if the secrets are not equal, the parties only get the information that they are not equal. By this usage, SMP can assure the two parties are indeed exchanging messages with the right person.

2.5 Summary

This chapter first presented different security principles which are important for secure messaging protocols to provide if they want to have a cryptographic secure protocol. The first section went through what encryption is, from unencrypted messaging between two parties to end-to-end encrypted messaging.

It then explained other security principles which are as important as the encryption, such as deniability, authentication, and confidentiality between parties.

The last section of this chapter explained how the first end-to-end encrypted instant messaging protocol, Off-the-Record, works. It went through different steps it takes to authenticate two parties and how they encrypt and sign each message before sending them over the Web. At the end of the section, an explanation on why each party re-keys the cryptographic keys for each message was given, and what the Off-the-Record protocol has implemented to defend against "man-in-the-middle" attacks

The next chapter is about the next state of the end-to-end encryption, the Signal Protocol, which is based on the Off-the-Record protocol. It explains how it is set up and how the protocol works.

Chapter 3

Signal Protocol

This chapter describes a new end-to-end encryption protocol called Signal which developers have started implementing in their messaging applications, with the goal of having a better adoption than the Off-the-Record protocol. The OTR protocol was one of the first protocols for instant messaging that implemented end-to-end encryption. OTR had an original feature, refresh the keys often: for each round trip of messages, the users established a new Diffie-Hellman shared secret. When a user always gets a new shared secret for each round trip, it will be impossible for others to decrypt previous messages. This became known as ratcheting [10].

The Signal Protocol is designed by Moxie Marlinspike and Trevor Perrin from Open Whisper Systems[1]. Open Whisper Systems wanted to develop a new end-to-end encryption standard which works in both synchronous and asynchronous messaging environments [34]. The goals of Signal include end-to-end encryption and advanced security properties such as forward secrecy and future secrecy [10]. Initially, Signal was divided into two different application, TextSecure[2] and RedPhone[3]. The former was about SMS and instant messaging, while the latter used as an encrypted VoIP[4] application. TextSecure was based on the Off-the-Record Protocol by taking the Ratchet from OTR and implemented a Double Ratchet, combining OTR's asymmetric ratchet with a symmetric ratchet [10], and naming it *Axolotl Ratchet*. Signal went later on combining TextSecure and RedPhone to form the new Signal application together with the protocol having the same name

In recent years, the Signal Protocol has been adopted by numerous companies, such as WhatsApp[5] by Facebook, the Messenger[6] also by Facebook, and Google's new messaging app, Allo[7].

The next sections provide more technical information about different methods that Signal uses to ensure end-to-end encryption. Section 3.1

[1]`https://whispersystems.org/`

[2]`https://whispersystems.org/blog/the-new-textsecure/`

[3]`https://whispersystems.org/blog/low-latency-switching/`

[4]`https://www.voip-info.org/wiki/view/What+is+VOIP`

[5]`https://whatsapp.com`

[6]`https://messenger.com`

[7]`https://allo.google.com`

presents the Double Ratchet Algorithm [38], giving an overview of how it works. Then section 3.2 presents Signal's way of implementing the Diffie-Hellman key agreement, called *Extended Triple Diffie-Hellman*[8].

The pictures shown during this chapter are taken from the official documentation about the Double Ratchet Algorithm and the X3DH Key Agreement protocol, and permission has been given by Moxie Marlinspike through an e-mail discussion.

3.1 The Double Ratchet Algorithm

This section presents more thoroughly the algorithm Double Ratchet. The Signal Protocol uses it to exchange encrypted messages based on a shared secret key, that two parties use. Typically the parties use some key agreement protocol to agree on the shared secret key [38]. The Signal Protocol uses the X3DH Key Agreement [39] Protocol which we describe later in section 3.2

The Double Ratchet Algorithm has several steps. The first step is the Key Derivation Function Chains (KDF Chains) [38]. The next two steps are about the two different ratchet algorithms, the first being symmetric-key ratchet and the second is the Diffie-Hellman ratchet. These three together form the Double Ratchet algorithm.

3.1.1 KDF Chain

KDF Chain is a core concept in the Double Ratchet algorithm [38]. We talked about the basics of a Key Derivation Function in section 2.3.2, it takes a secret, random KDF key and some input data and then returns one key used as a new KDF key for the next chain as well as an output key for messages. The output data is always indistinguishable from random provided the key is not known. As we mentioned in section 2.3.2, the parameter r (here: KDF key) does not have to be a secret or random number, because the KDF output provides a secure cryptographic hash of the input data and keys either way.

[8]`https://whispersystems.org/docs/specifications/x3dh/`

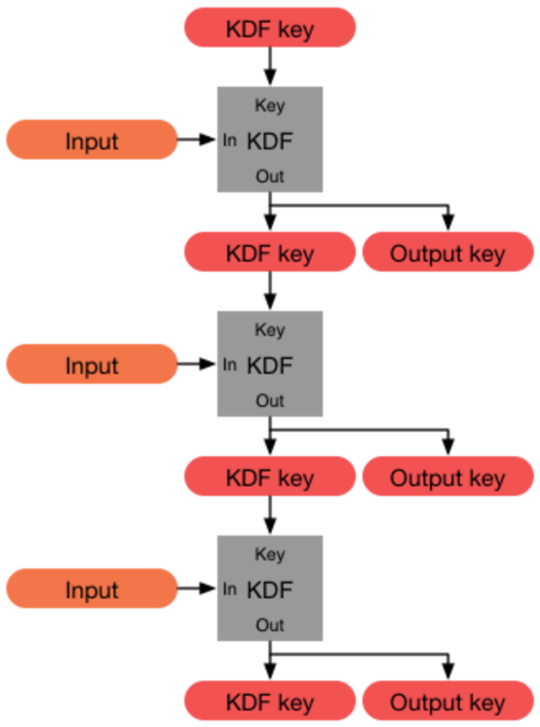


Figure 3.1: KDF chain processing three inputs and producing three output keys [38]

The KDF Chain has the following important properties [38]:

- **Resilience:** The output keys appear random to an adversary without knowledge of the KDF keys, even if the adversary has control of the KDF inputs
- **Forward security:** Output keys from the past appear random to an adversary who learns the KDF key at some point in time
- **Break-in recovery:** Future output keys appear random to an adversary who learns the KDF key at some point in time, provided the future inputs have added sufficient entropy.

The job of the KDF chain throughout the Double Ratchet session is to store each parties KDF key for three chains: a *root chain*, a *sending chain*, and a *receiving chain* (Alice's sending chain matches Bob's receiving chain, and vice versa, which we show more in section 3.1.3, where KDF is integrated with the Diffie-Hellman).

The KDF chains are both part of the Diffie-Hellman ratchet step and the Symmetric-key ratchet step. While the parties exchange messages, they need to exchange new Diffie-Hellman public keys. The secrets from the DH output become the inputs to the root chain for the KDF chain, and then the output keys from the root chain become new KDF keys for the sending and receiving chains. This is the Diffie-Hellman ratchet [38].

The outputs from the Diffie-Hellman ratchet, the sending and receiving chains advances for each sending and receiving message, by using the chains as inputs in the KDF and the output keys are then used to encrypt and decrypt messages. This is called the symmetric-key ratchet [38].

3.1.2 Symmetric-Key Ratchet

The symmetric-key ratchet uses KDF chains for its sending and receiving chains. The output keys are unique message keys which are used to either encrypt or decrypt messages. The KDF keys for the symmetric-key ratchet chains will be called chain keys throughout the rest of the Double Ratchet description [38].

The KDF chains used for the sending and receiving chains are constant; they do not need to be random or a secret because the chain key is derived from the Diffie-Hellman KDF chain which is cryptographically secure. The documentation explains the constant can be a single byte 0x01 for the message key and then a single byte 0x02 for the chain key [38]. The sending and receiving chains ensure that each message is encrypted or decrypted with a unique key and can be deleted after use. There is only a single symmetric-key ratchet step to calculate a new message and chain key from an already given chain key. The diagram below shows two steps in this process. The first KDF gets a chain key from the Diffie-Hellman KDF chain and outputs a new chain key for the next KDF and a message key to encrypt or decrypt a message.

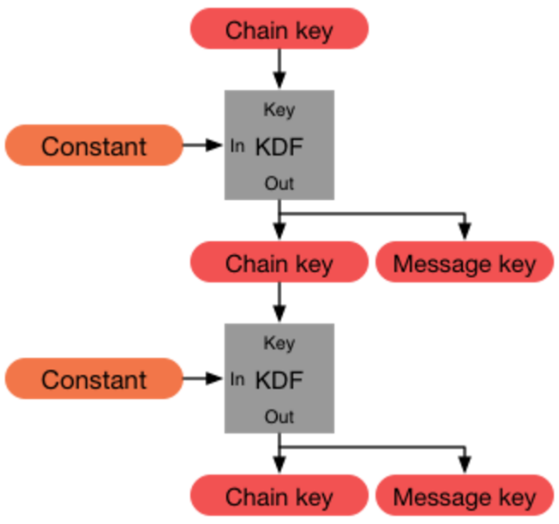


Figure 3.2: Two steps in the symmetric-key ratchet [38]

The message key that is derived from the KDF chain is not used later on to derive new message keys or chaining keys. Because of this, it is fully possible to store the message key without affecting the security of other keys, only the message that belongs to the particular message key. It is quite useful when the protocols handles out-of-order messages because a participant can store the message key and decrypt the message later when they receive the correct message for that message key. We talk more about out-of-order messages in section 3.1.5.

3.1.3 Diffie-Hellman Ratchet

The Double Ratchet is formed by combining the symmetric-key ratchet and the Diffie-Hellman ratchet. If the Double Ratchet did not use the Diffie-Hellman ratchet to compute new chain keys for the sending and receiving chain keys, an attacker could steal one of the chain keys and then compute all future message keys and decrypt all future messages [38].

For this to work, each party generates a DH key pair, a public and a private key, which will be their first ratchet key pair. When a message is sent, the header must contain the current public key. When a message is received, the receiver checks the public keys that are given with the message and do a DH ratchet step to replace the receiver's current ratchet key pair with a new one [38].

The result is a kind of "ping-pong" behavior as the two parties take turns replacing their key pairs. The attacker will have a harder time to get any valuable information from the parties, since if one of the messages gets compromised, and the attacker learns the value of the current private key, it will not make any difference since the private key will soon be replaced with a new, uncompromised key [38].

From this point onwards, this section will present an example of how the Diffie-Hellman ratchet works to get a shared sequence of DH outputs.

The initialization starts with Bob sending his ratchet public key to Alice, while Bob does not know Alice's ratchet public key. Alice will now do the initialization step by performing a DH calculation between her ratchet public key and Bobs ratchet public key.

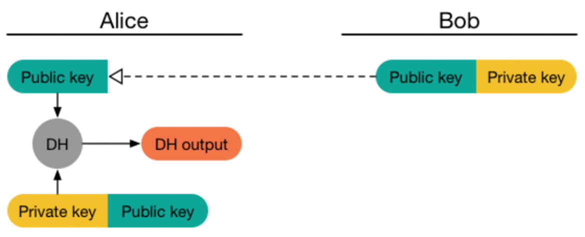


Figure 3.3: Initialization of DH Ratchet by Alice [38]

Figure 3.4 shows Alice advertise her ratchet public key to Bob after she is done with her Diffie-Hellman calculation. When Bob receives Alice's initial message, he performs a Diffie-Hellman ratchet step by calculating the new DH output between Alice's ratchet public key and his ratchet private key, which equals Alice's DH output. The DH outputs are equal because the figures are a simplification of the DH ratchet, and there is a KDF chain which uses a Root Key (shared secret between Alice and Bob) to output the same keys. He then replaces his ratchet key pair and calculates a new DH output.

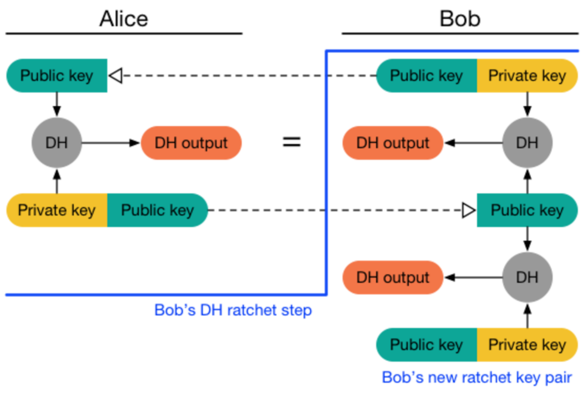


Figure 3.4: Initialization of DH Ratchet by Bob [38]

Bob sends his next message with the new ratchet public key. Eventually, Alice receives Bob's new message with his new ratchet public key. Alice derives a new DH output with her ratchet private key and Bob's new public key to get the same DH output as Bob for decrypting the message and then generates a new DH ratchet key pair to replace her old key pair. With the new DH ratchet key pair, Alice derives another DH output with her new ratchet private key and the same ratchet public key from Bob, which is used in the next message exchange. These exchanges continue for each message sent with a new ratchet public key and new DH output.

Until now the DH outputs have only been called outputs to simplify the description of the DH ratchet, but the DH outputs are used in a KDF to get the sending and receiving chain keys for the symmetric-key ratchet. Figure 3.5 shows the DH outputs are changed with sending or receiving chains. The first time Alice sends out her message with her ratchet public key and DH output, a sending chain is derived from the DH output through a KDF chain, and Bob on his side derives a receiving chain key from his DH output by using Alice's public key and his private key. Bob then derives a new sending chain key from his second DH output [38].

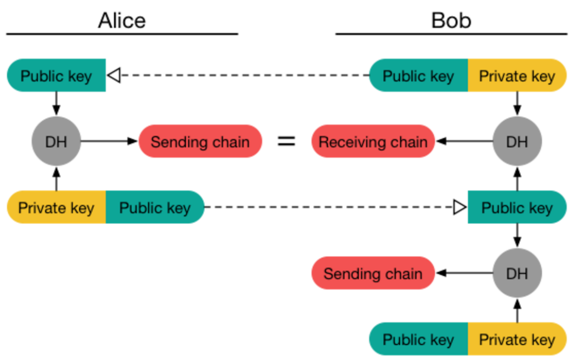


Figure 3.5: Sending and receiving chains [38]

The figures shown until now have been a simplification of the DH ratchet. The DH ratchet takes the output keys and uses them as KDF inputs to a root chain, and then the KDF outputs are used as a sending or receiving key. Using a KDF chain here improves the resilience and break-in recovery. Figure 3.6 shows that the DH ratchet updates the root KDF chain twice, and uses the KDF output keys as new sending and receiving chain keys [38].

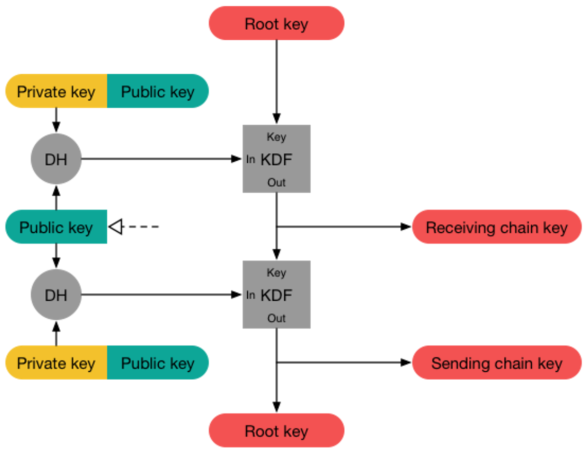


Figure 3.6: Full DH ratchet step [38]

3.1.4 Double Ratchet

Double Ratchet is the algorithm we get by combining symmetric-key ratchet and Diffie-Hellman ratchet.

- 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 [38]
- 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 [38]

This section is about how these algorithms work together to form the Double Ratchet, seen from Alice's perspective. Figure 3.7 shows after Alice has done her initialization with Bob's ratchet public key, and the Root Key (RK) is the shared secret used as the initial root key for the KDF chain. Alice generates her ratchet key pair and then sends the DH output to the root KDF chain to calculate a new root key and a sending chain key (CK) [38]. The figure is split up into four different parts. The ratchet is where the ratchet key pair changes and sends Alice's ratchet private key and Bob's ratchet public key to generate a DH output for the root KDF chain input. The sending column is where the message key for encryption generates through the symmetric-key KDF chain. The last column is where the message key for decrypting the received message gets also derived by the symmetric-key KDF chain. To ensure that the secrecy throughout the Double Ratchet is upheld, the old RK gets deleted after it has been used to derive a new RK.

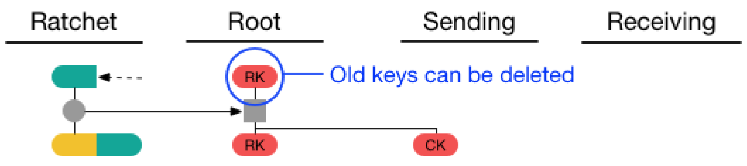


Figure 3.7: Initial Double Ratchet Step by Alice [38]

Figure 3.8 shows Alice sending her first message to Bob, message A1. The sending CK is used on a symmetric-key ratchet step to derive a new CK and a message key, A1, to encrypt her message. The new CK is stored for later use, while the old CK and the message key can be deleted since it no longer is of any use to Alice.

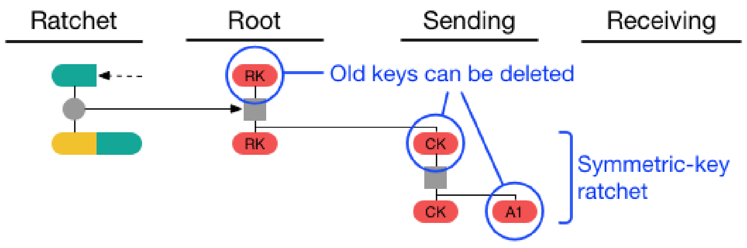


Figure 3.8: First double ratchet message A1 [38]

Figure 3.9 shows Alice receiving her first response from Bob, message B1, and he has sent his new ratchet public key, which means Alice needs to calculate a new ratchet key pair. Alice applies a new DH ratchet step to derive a new receiving and sending chain keys. Alice uses her old ratchet private key and Bob's new public key, to derive a new RK and a CK for the receiving KDF chain. The receiving chain key is used to derive a new receiving chain key and a message key to decrypt Bob's message. Then she derives a new DH output for the next root KDF chain with her new ratchet private key to derive a new RK and a sending CK.

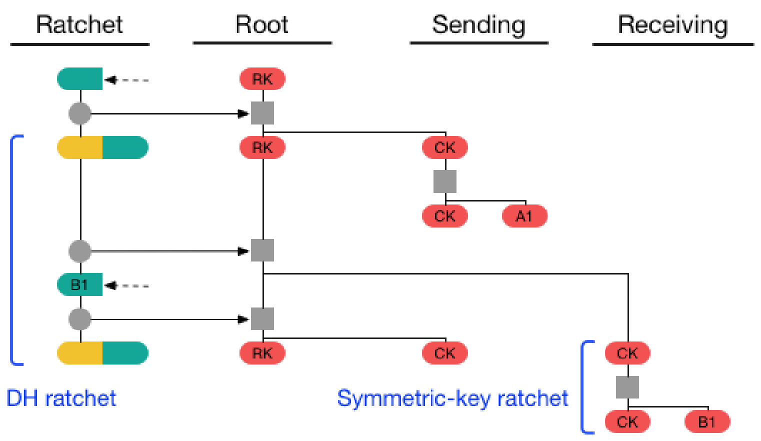


Figure 3.9: First received double ratchet message B1 [38]

The next figure, 3.10, shows how many ratchet steps Alice does when sending a message A2, receiving a message B2 with Bob's old ratchet public key, and then send two new messages A3 and A4 to Bob. Alice received a message B2 with Bob's old ratchet keys, which means that Alice only needs to do a symmetric-key ratchet step to derive a new receiving CK and a message key to decrypt message B2. Before Alice sends her second message A2, she needs to do a symmetric-key ratchet step to derive a new sending CK and a message key to encrypt her message A2. The same must be done with message A3 and A4, by ratcheting the symmetric-key ratchet two more times to derive the correct message keys.

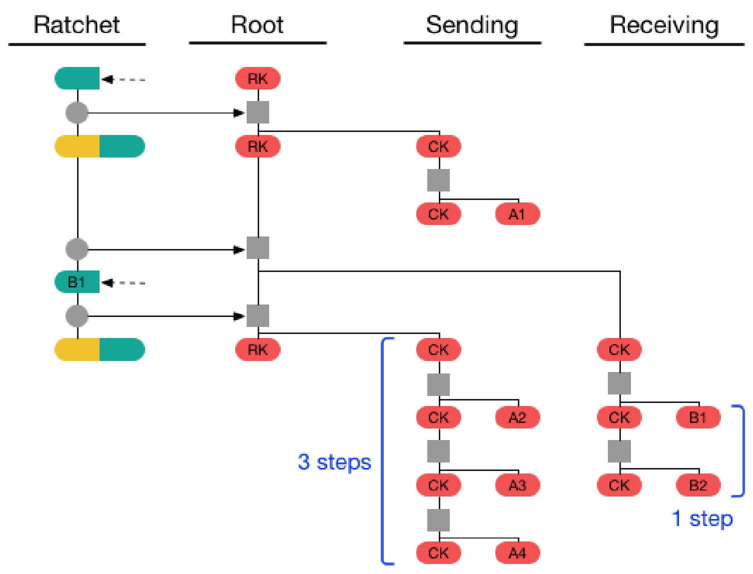


Figure 3.10: Double ratchet message A2, A3 and A4, received B2 [38]

Figure 3.11 shows the state when Alice receives messages B3 and B4 from Bob with his new ratchet public key, and the sending of Alice's last message, A5. Alice receives new messages from Bob with his new ratchet public key, and she first derives a new DH output to ratchet the root KDF chain to get a new RK and a new receiving CK to decrypt Bob's messages. The receiving CK is used to run the symmetric-key ratchet step two times, ones to derive a new CK and a new message key for decryption of message B3 and then another ratchet step to derive the second CK and message key for message B4. Alice generates a new ratchet key pair and uses her new ratchet private key to derive a new RK and a new sending CK with Bob's new ratchet public key. The sending CK is used to do a symmetric-key ratchet step to derive a new CK and a message key to encrypt her new message, A5.

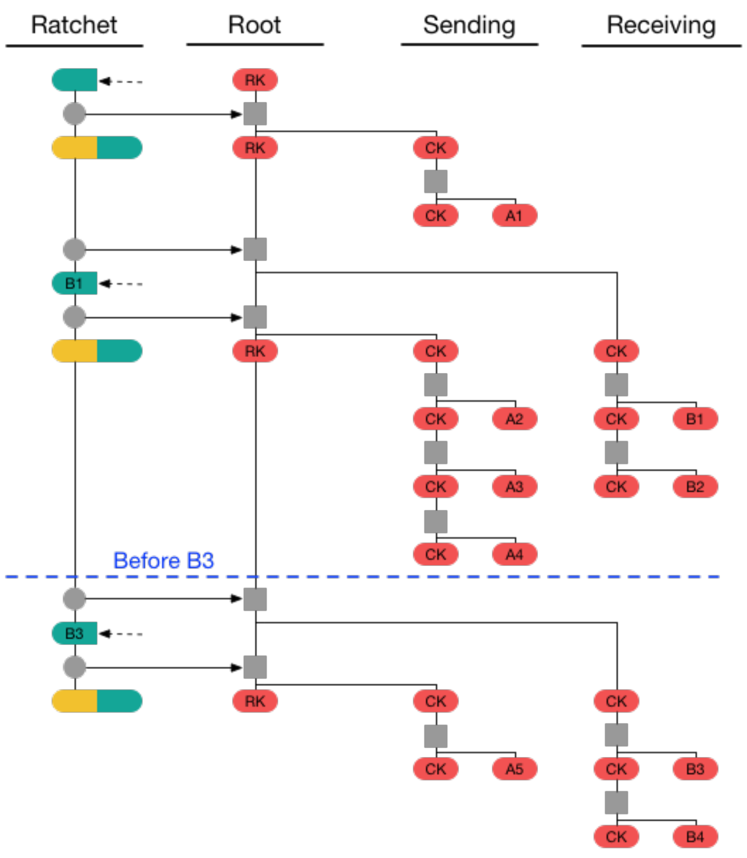


Figure 3.11: Double ratchet message A5, received B3 and B4 [38]

3.1.5 Out-of-Order Messages

The Double Ratchet handles lost or out-of-order messages by including in each message header the message's number in the sending chain (N=0,1,2,...) and the length (number of message keys) in the previous sending chain (PN) [38]. The reason for this is for the receiver to advance the keys to the relevant message key, while still storing the skipped message keys in case they receive the message at a later time.

When receiving a message, and a DH ratchet is triggered, then the received PN minus the length of the current receiving chain is the number of skipped messages in that receiving chain. The N which is received is the number of messages which are skipped in the new receiving chain after DH ratchet step.

If a received message does not trigger a DH ratchet step, then the receiver needs to minus the received N with their own receiving chain length.

To explain this with a figure, we can use the message sequence from the previous section, but here message B2 and B3 are skipped. Alice received message B4 from Bob, with the PN = 2 and N = 1. Alice sees that she would need to do a DH ratchet step, but first, she calculates how many message keys she needs to store from her current receiving chain (Bob's previous sending chain). Since PN = 2 and her current receiving chain length is 1, the number of stored keys from the current receiving chain is 1 message key (B2). Then she does a DH ratchet step where a new receiving chain is derived. Because the length of her new receiving chain is 0, she needs to store a message key from her new receiving chain (B3). After Alice has stored B2 and B3, she can derive the last message key to decrypt message B4.

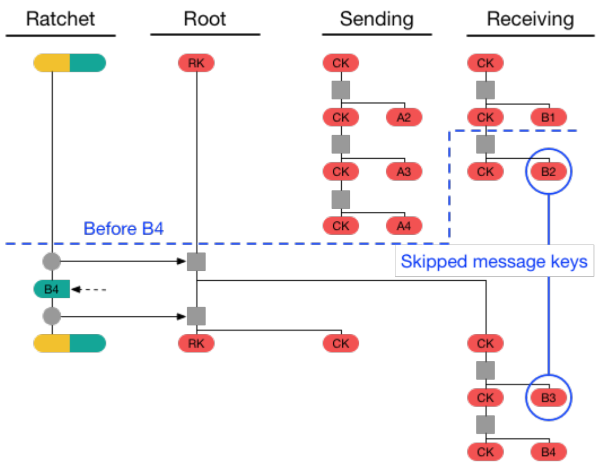


Figure 3.12: Handling of Out-of-Order Messages [38]

3.2 The X3DH Key Agreement Protocol

The Double Ratchet algorithm needs some key agreement protocol between two parties. In section 3.1 we talked about using the Extended Triple Diffie-Hellman key agreement protocol(X3DH). X3DH is used to establish a shared secret between two parties who mutually authenticate each other based on public keys [39], and at the same time provide both forward secrecy and cryptographic deniability.

This method of doing key agreement is designed for asynchronous settings where one user, Bob, is offline but has published information to a server. Another user, Alice, wants to use that information to send encrypted data to Bob, and also establishes a shared secret for future communication [39]

3.2.1 Preliminaries

There are a few specifications that are already set and chosen beforehand for someone to implement the X3DH protocol. Three parameters need to be followed by everyone; curve, hash, and info and these are used throughout this section. Curve means either elliptic curve X25519[9] or X448[10]; hash means either 256 or 512-bit hash function[11] and a parameter, info, which is an ASCII string identifying the application [39].

Throughout this section, a set of keys are used with the protocol. These keys are chosen because when Alice wants to send Bob some initial data using encryption and establish as shared secret, she asks the server for Bob's keys which he has uploaded to the server in advance to allow other parties to establish a shared secret with him. A server is used to store messages from Alice and Bob which Bob later can retrieve, and the server keeps these sets of keys for Alice and Bob to retrieve when needed [39]. These keys are elliptic curve public keys:

- IK_A: Alice's identity key
- EK_A: Alice's ephemeral key
- IK_B: Bob's identity key
- SPK_B: Bob's signed prekey
- OPK_B: Bob's one-time prekey

The public keys used within an X3DH protocol run must either all be in X25519 form or X448 [39]. Each party as a long-term identity public key, but Bob also has a signed prekey, which he changes periodically, and a set of one-time prekeys. The one-time prekeys are used in a single X3DH protocol run.

3.2.2 X3DH Protocol

The X3DH Protocol has three different phases that this sections is going through:

1. Bob publishes his identity key and prekeys to a server
2. Alice fetches a "prekey bundle" from the server, and uses it to send an initial message to Bob
3. Bob receives and processes Alice's initial message

Phase 1: Publishing keys

Phase one is about what Bob needs to do before Alice can get any information from the server about Bob. He will need to send a set of elliptic curve public keys to the server to be stored:

- Bob's identity key IK_B

[9]https://tools.ietf.org/html/rfc7748

[10]https://tools.ietf.org/html/rfc7748

[11]https://www.wikiwand.com/en/Cryptographic_hash_function

- Bob's signed prekey SPK_B
- Bob's prekey signature Sig(IK_B, Encode(SPK_B))
- A set of Bob's one-time prekeys (OBK_B^1, OBK_B^2, OBK_B^3, ...)

Identity keys need to be uploaded to the server once, while other keys, such as new one-time prekeys can be uploaded again later if the server is getting low, by informing Bob to upload more. How the one-time prekeys are used is explained thoroughly in phase two. On the other hand, new signed prekey signatures need to be uploaded periodically, and it is up to Bob when he wants to upload them, every day, once a week, or once a month [39].

If the other user has not managed to update their signed prekeys because of delay in transit, and they send messages to Bob before getting new prekeys, it is important that Bob stores previous private keys corresponding to the old signed prekey, even after uploading a new signed prekey to the server [39]. It is important to delete the private key eventually, to keep the forward secrecy.

Phase 2: Sending The Initial Message

Alice contacts the server and fetches a "prekey bundle" containing Bob's keys. She receives the same keys Bob pushed to the server in phase one. Alice now has Bob's identity key, signed prekey, prekey signature and optionally a single one-time prekey.

Alice receives Bob's one-time prekey if the server still has more of them since it will delete a one-time prekey each time it sends it to another user. If the server does not have more of them in store, the bundle will then not contain the prekey [39].

Alice tries to verify the prekey signature, but if it does not verify correctly she will need to abort the protocol. The verification process is written more in the documentation about the signatures from Signal [48]. When the verification is a success, Alice will generate an ephemeral key pair with the public key EK_A [39]. If the bundle does not contain a one-time prekey, Alice will calculate:

```
DH1 = DH(IK_A,  SPK_B)
DH2 = DH(EK_A,  IK_B)
DH3 = DH(EK_A,  SPK_B)
SK  = KDF(DH1  ||  DH2  ||  DH3)
```

If the bundle does however contain a one-time prekey, another DH calculation is added:

```
DH4 = DH(EK_A,  OPK_B)
SK  = KDF(DH1  ||  DH2  ||  DH3  ||  DH4)
```

Figure 3.13 explains the calculation process better by showing how the different keys are calculated. DH1 and DH2 provide mutual authentication, while DH3 and DH4 provide forward secrecy [39]

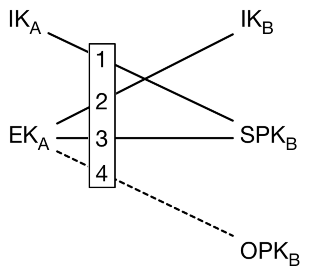


Figure 3.13: DH1...DH4 calculations [39]

It is important to know that Alice deletes her ephemeral private key and the DH outputs to preserve secrecy after she is done calculating the SK. She calculates an "associated data" byte sequence AD that contains information for both parties [39].

```
AD = Encode(IK_A) || Encode(IK_B)
```

Alice could concatenate more information to the sequence, such as usernames, certificates or other information which may be important. Alice then sends her initial message to Bob containing:

- Alice's identity key IK_A
- Alice's ephemeral key EK_A
- Identifiers stating which of Bob's prekeys Alice used
- An initial ciphertext encrypted with some AEAD encryption scheme [50] using AD as associated data and using an encryption key which is either SK or the output from some cryptographic pseudo-random function keyed by SK

Alice's initial ciphertext is typically used as the first message in a post-X3DH communication protocol, such as the Double Ratchet protocol from section 3.1. The ciphertext has two roles, serving as the first message within some post-X3DH protocol, and as part of Alice's X3DH initial message to Bob [39].

Phase 3: Receiving The Initial Message

Phase three of the X3DH is mostly the same as phase two. Bob retrieves Alice's initial message which contains Alice's identity key and ephemeral key from the message. Bob will then load his identity private key and the private key(s) corresponding to the signed prekey and one-time prekey that Alice used [39].

Bob repeats the same steps with DH and KDF calculations to derive his own SK and then deletes the DH values the same as Alice did. Afterwards, he constructs the AD byte sequence, and in the end, tries to decrypt the initial ciphertext using the SK and AD. The decryption is the only difference

Bob does to what Alice did on her side. If the decryption fails, Bob will delete the SK and the protocol aborts and the participants need to restart the protocol from the start [39].

If the decryption is successful, he gets the information that Alice had encrypted, and the protocol is complete for Bob. He deletes any one-time prekey private key that was used during the protocol, to uphold the forward secrecy and not get it compromised.

3.3 Summary

This chapter described in detail two different methods that together form the Signal Protocol. The first is an algorithm called Double-Ratchet algorithm which is combined with a Diffie-Hellman ratchet and a Symmetric-key ratchet. It explained how the protocol is set up by the two ratchets and showed how they work together.

The second method is a protocol called X3DH Key Agreement Protocol which is used to come to an agreement between parties on a shared key, to be use to encrypt messages when they want to communicate with each other.

In the next chapter, the analysis of three secure messaging protocols is conducted to see what types of security principles they provide in both one-to-one and many-to-many conversations.

Chapter 4

Analysis of Secure Messaging Protocols

This chapter presents the analysis of secure messaging protocols which are used by applications that are analyzed in Chapter 5. We discuss:

- the systematization method which is chosen,
- what kind of threat models are taken into consideration when evaluating the protocols, and
- how the table 4.1 is structured with respect to the security and privacy properties.

The structure of this chapter is based on the research paper by Nik Unger et al. [53].

After the discussion on how the analysis was done, we look in detail at three protocols which use some of the same security principles on securing message conversations with end-to-end encryption. The first is Off-the-Record, which is the baseline for the two other protocols, Signal and Matrix.

4.1 Systematization Method

4.1.1 Threat Model

When evaluating the security and privacy properties of the secure messaging protocols, we assume there are multiple threats that could occur during the evaluation. Our threat model when going through the evaluation of the protocols in this chapter may occur by the following adversaries:

- **Active adversaries:** Man-in-The-Middle attacks are possible on both local networks and global networks by adding a proxy between the applications and servers handling the messages.
- **Passive adversaries:** These adversaries log everything that is sent to and from a user and could potentially use that information to keep

track on who they talk to and when. Passive adversaries could also log information such as messages and keys, even though the contents of the messages are encrypted.

- **Service providers:** The messaging systems that require centralized infrastructure (such as Signal and Matrix), need to keep the information about users secure. The service operators could at any time become a potential adversary.

We assume the endpoints on the applications that these protocols are implemented into are secure and that the user's devices do not have any kinds of malware or exploits.

4.1.2 Structure

The structure of this section is based on the research paper by Nik Unger et al. where we take the same security and privacy properties which have been explained in the research paper [53] and in section 2.2. We only look at the conversation security of three different secure messaging applications which all support end-to-end encryption, but only two of them support end-to-end encryption for group conversations. The problem areas are divided into three separate areas; security and privacy, usability and adoption, and group chat features. Usability features of the secure messaging protocols have been left out of this chapter, because we look at the different usability features in chapter 5 and test what kind of features the applications delivers.

- **Security and Privacy properties:** We look at the underlying protocol on how it is set up and if they handle the encryption as it is meant. When going through the evaluation of the protocols, we assume the protocols are implemented the way it is designed without having any known exploits. The security and privacy properties chosen are basic properties which are achieved by using different types of cryptographic primitives.

- **Usability and Adoption:** Even though we go through usability features in the next chapter, it is important to at least look at some usability and adoption factors, such as resilience properties when evaluating the protocols.

- **Group chats:** In section 2.2.9 we discuss the group chat properties which should be provided by the protocols and applications using them.

4.2 Secure Messaging Protocols

Table 4.1: Result of the Analysis of Secure Messaging Protocols

Protocol	Client	Security and Privacy														Adoption					Group Chat				
		Confidentiality	Integrity	Authentication	Participant Consistency	Destination Validation	Forward Secrecy	Backward Secrecy	Anonymity Preserving	Speaker Consistency	Causality Preserving	Global Transcript	Message Unlinkability	Message Repudiation	Particip. Repudiation	Out-of-Order Resilient	Dropped Message Resilient	Asynchronicity	Multi-Device Support	No Additional Service	Computational Equality	Trust Equality	Subgroup Messaging	Contractable	Expandable
Off-the-Record	Pidgin	●	●	●	●	●	◑	●	●	◑	◑	-	●	●	◑	◑	◑	-	-	●					
Signal	Signal	●	●	●	●	●	●	●	-	●	●	-	●	●	●	●	●	●	◑	-	●	●	●	●	●
Matrix	Riot	●	●	●	●	●	◑	◑	-	●	●	-	●	●	●	●	●	●	●	-	●	●	●	●	●

● = provides property ◑ = partially provides property

4.2.1 Off-the-Record

The Off-the-Record protocol is based on the authenticated Diffie-Hellman key exchange, but they introduced a different ratcheting approach, by attaching new DH contributions to messages [8]. The authenticated Diffie-Hellman design, such as SIGMA (used in OTR), is when the participants generate new ephemeral session keys and authenticate the exchange with their long-term keys. The shared secret from this key exchange is used to derive a sending and receiver cipher key for each party, as well as a set of MAC keys for each party [35]. Afterwards, the protocol uses those to protect messages using an encrypt-then-MAC approach which in the end provides confidentiality, integrity, and authentication. The SIGMA protocol also adds participation consistency feature for key exchange [32].

The OTR protocol signs the messages with the shared MAC keys and not the long-term keys, but to strengthen the message unlinkability and message repudiation features, they also publish the MAC keys and at the same time use malleable encryption [8]. OTR only signs the ephemeral keys and not every parameter during the key exchange; then it provides partial participation repudiation since the conversation partners can use the signed ephemeral keys to forge transcripts.

Backward secrecy is provided when message keys are computed by new DH values which are advertised by the sender with each sent message. The security property is provided because a compromised key regularly gets replaced with new key material during the conversation [53]. Anonymity preservation is provided by OTR since the long-term public keys are never observed neither during the key exchange nor the conversation. Causality preservation is partially achieved since messages implicitly reference their causal predecessors based on which keys they use [53]. Speaker consistency is partially obtained since an adversary cannot drop messages without also dropping all future messages because then the recipients would not be able to decrypt succeeding messages [53]. The aftermath of the speaker consistency is that the recipient needs to hold on to out-of-order messages because if they do not come in order the message will be encrypted with an unexpected key, and at the same time the window of compromises enlarges, and the OTR ends up only partially providing the forward secrecy. Out-of-order and dropped messages are partially provided because if a message is out-of-order or dropped during the transmission, the protocol can store the decryption key until the participant receives it. The problem of storing the decryption key is that it raises the possibility of successful attacks by adversaries.

The Off-the-Record protocol is made for instant messaging, which does not provide asynchronous messaging between participants, but because of the synchronous capability, the OTR protocol does not rely on additional services to establish a connection between two participants.

4.2.2 Signal

The Signal protocol is made up of a Double Ratchet algorithm, 3-DH Handshake, and prekeys for the asynchronous ability it provides. The Double-Ratchet is, as we talked about in chapter 3.1, composed of a ratchet based on a KDF and a ratchet based on the Diffie-Hellman key exchange (OTR DH-ratchet), which takes the same security features as the Off-the-Record protocol, but in some cases adds stronger or new features as well. Forward secrecy is provided because the KDF ratchets, and the backward secrecy is provided since even though the KDF keys gets comprised, they eventually will be replaced by new keys.

3-DH Handshake provides the same level of authentication as the AKE from Sigma, but 3-DH manages to achieve full participation repudiation since anybody can forge a transcript between two parties [53]. It does not manage to continue providing anonymity preserving because 3-DH uses the long-term public keys during the initial key agreement.

The prekeys are used to achieve asynchronous messaging system by sending a set of prekeys (ephemeral public DH contributions) to a central server, and then a sender can request the next prekey for the receiver to compute encryption keys. By using a central server to keep the prekeys, the Signal protocol loses the no additional service property.

Out-of-order and dropped messages are fully supported on one-to-one conversations asynchronously by the use of prekeys, as it is implemented in the Double Ratchet algorithm together with the X3DH Key Agreement which we have discussed in chapter 3. A set number of prekeys are uploaded to a central server which stores them securely and sends one at a time to the user who requests a key to encrypt a message.

Group conversation is achieved by using multicast encryption, which when sending a single encrypted message to the group, it is sent to a server and then relays it to the other participants while the decryption key is sent as a standalone message to each member of the group conversation. The group conversation provides asynchronous messaging, speaker consistency and causality preservation are achieved by attaching message identifiers, of the message before, to the new messages [53], but it cannot guarantee participant consistency. Multi-device is partly provided by Signal, in that sense that only an extra computer can join in on the conversation by using the Signal Desktop application[1], which is only a Chrome Extension[2] and not its own application.

The Signal Protocol does provide computational and trust equality, subgroup messaging, contractible and expandable properties. By using pairwise group messaging and multicast encryption, Signal gets the ability to push group management into the clients themselves, which makes it easier for the users to change the group, expand it or shrink it in size, without having to restart the whole group conversation and protocol. When users want to send a group message they send a message to each of the users that are participating and adding a parameter to the header explaining that it is meant for the specific group chat. The Signal server does not know about the group conversation, since the messages are encrypted using their normal public key. The pairwise group messaging also makes the computation of new cryptographic keys and trust equally as demanding as if there only was a one-to-one conversation.

4.2.3 Matrix

The Matrix protocol consists of two different algorithms, the olm[3] for one-to-one conversations and megolm[4] for group conversations between multiple devices. The olm algorithm is based on the Signal Protocol which means they achieve the same security properties as Signal does, while the megolm algorithm is a new AES-based cryptographic ratchet developed for group conversations. The NCC Group has audited both of the algorithms [4].

While the Matrix protocol does supply the users with the same security properties as Signal, it does have some extra features which are provided by the megolm algorithm. Multiple devices are possible with Matrix because megolm implements a separate ratchet per sending device which is participating in a group conversation [43]. Each group conversations manage when the ratchet should be replaced by the senders, such as when a new device joins, leaves or after N number of messages. The protocol does not restart when the ratchet is replaced with a new one, which provides computational and trust equality, subgroup messaging, contractible and expandable properties.

The NCC Group has audited both of the algorithms [4] and found out that megolm has some security flaws about forward and future secrecy. If an attacker manages to compromise the key to Megolm sessions, then it can decrypt any future messages sent to the participants in a group conversation. The Matrix SDK, which is used in the applications that have implemented the Matrix protocol such as Riot; the Megolm keys get refreshed after a certain amount of messages sent between the participants. Forward secrecy is also partially provided since the megolm maintains a record of the ratchet value which allows them to decrypt any messages

[1]`https://whispersystems.org/blog/signal-desktop/`

[2]`https://en.wikipedia.org/wiki/Browser_extension`

[3]`https://matrix.org/docs/spec/olm.html`

[4]`https://matrix.org/docs/spec/megolm.html`

sent in the session after the corresponding point in the conversation [44]. The Matrix developers have stated that this is intentionally designed [27], but also said that it is up to the application to offer the user the option to discard old conversations [44].

4.3 Discussion

The table 4.1 shows that none of the secure messaging protocols we have gone through in this chapter can give the users every security property. There is room for improvement, but for this to work, the researchers need to work together throughout the different protocols to come up with new ways of implementing the rest of the properties that are not achieved.

While the Off-the-Record protocol does not need any additional services to function as it is designed, it cannot give the user's group conversation in the state it is now. There have been a few research papers looking at group conversations with OTR in mind [7, 33, 24], but they have not received enough traction by the users or developers because it does not support asynchronous chat conversations.

While Signal only allows for one single device to be used, it does support desktop through the Chrome Extensions, but it does not support native desktop application or multiple mobile phones to be added to a users account. They could implement the same function as group conversations with multicast encryption, where the user who sends the message has information on all of its devices and encrypts the message for each n-number of devices, which then gets forwarded to all of them at the same time. The efficiency could be a problem, and the Signal team needs to find a way of doing it efficiently because users are used to multiple devices when having a conversation.

The Matrix protocols support multiple devices, and it does not hurt the efficiency of the conversations. On the other hand, it does not achieve full forward, and backward secrecy from the protocol, but the implementation of the protocol supports it.

While Signal and Matrix are providing extremely useful secure messaging protocols, the adoption rate is going fairly slow. The Signal protocol has been audited by a couple of research groups in 2016 [10, 31] and since it is open source, the community can come together and improve on the implementations of both the protocol and implementations. They have also implemented their protocol in applications that have millions of users, such as the WhatsApp[5] and Google Allo[6]. The Matrix protocol has also been audited which means that researchers are taking them seriously and sees a healthy future for them, and at the same time strengthens the credibility of the protocol.

We consider this information throughout this chapter to be useful for other to read through and use as a building block for their research on these protocols. The usability properties for each protocol and its

[5]https://whatsapp.com/

[6]https://allo.google.com

implementations are represented in the next chapter, when we go through the applications that implement these protocols and look at how the usability is done.

4.4 Summary

This chapter went through three secure messaging protocols which promise end-to-end encrypted messaging. Off-the-Record is the protocol which only supports one-to-one synchronous conversations which mean users need to be online to chat with each other, while Signal and Matrix offer asynchronous chat conversations.

The secure messaging protocols do not provide every single security property, which means there is room for improvement for all of the protocols. Security researchers need to work together throughout the protocols to come up with ways of implementing the rest of the properties that are not achieved.

The next chapter is about the testing methodology and the different test scenarios and test cases for secure messaging applications which implement the Signal and Matrix protocol to ensure the users have one-to-one and many-to-many asynchronous chat conversations.

Chapter 5

Implementations of Signal Protocol

This chapter presents applications that have emerged after the NSA and Edward Snowden scandals, as well as old applications that implemented secure messaging capabilities in the last couple of years. It investigates a set of usability properties the application have when it comes to secure messaging.

The first section presents the testing method of the application analysis, how the case study is set up and which applications the analysis is going to test. The section presents each application and the step by step testing of each test scenario.

5.1 Testing Method

This section explains in detail what types of mobile phones were used during the test, and which applications were tested. The applications used during the testing phases are apps that advertise secure messaging conversation capabilities between one-to-one and many-to-many.

For this test, we used two separate Android mobile phones[1]. The first mobile phone is a Sony Xperia Z5 running stock Android 7.0 operating system with a December 1st security patch, and kernel 3.10.84-perf-gda8446.[2] The second phone is a Google Nexus 5X running stock Android 7.1.2 with a January 5h security patch, and kernel 3.10.73-gbc7f263.[3] Both phones have their personal phone number; the Sony phone has the contact information of the Nexus phone named Bob, while the Nexus phone has the contact details of the Sony phone named Alice. The reason behind the contact details is to quickly find each other when initiating a conversation during the testing. The mobile phones in use are:

[1]`https://www.android.com/`

[2]`https://www.sonymobile.com/us/products/phones/xperia-z5/`

[3]`https://www.google.com/intl/no_no/nexus/5x/`

Table 5.1: The phone models involved in the testing

Phone	Alice	Bob
Model	Sony Xperia Z5	Google Nexus 5X
OS	Android 7.0	Android 7.1.2
Security Patch	December 1st 2016	January 5th 2017
Kernel	3.10.84-perf-gda8446	3.10.73-gbc7f263
CPU	Qualcomm MSM8994 Snapdragon 810	Qualcomm MSM8992 Snapdragon 808
Memory	3GB	2GB

The applications used during the testing phase are locked to one version number and do not get updated to keep the research consistent. Both devices have the same applications installed and the identical version number. The apps tested with their respective version number are:

Table 5.2: Test Applications

Application	Version
Signal	3.30.4
WhatsApp	2.17.79
Wire	2.28.317
Viber	6.6.0.888
Riot	0.6.9
Telegram	3.17.1

5.1.1 Test Scenario

This section explains the test scenarios we went through to test a few security and usability steps to ensure that the security within conversations are less likely to be compromised. The test scenarios were the same for each application, and screenshots were taken during the testing phases to gather enough information for later to look through and analyze each step on how the applications handled the tests. We are going to study which properties the applications support for each test scenarios. These particular properties are chosen to ensure that the applications have both the most secure and usable conversations. The test scenarios the applications are going through are:

1. Setup and Registration
2. Initial contact
3. Message after key change
4. Key change while a message is in transit
5. Verification process between participants
6. Other security implementations

Setup and Registration

The setup and registration process is the first steps a user needs to go through after installing the application. This test checks how the applications handle the registration process, what the user needs to do to register a new account and if there are other ways to register or only with a phone number.

After a user has initiated the registration process, we look at how the user gets verified by the application and provider, does the user receive a verification code by SMS and then type the code to verify or does the verification code get sent over a phone call. If the application does not use a verification code during the registration phase, the risk of being an identity theft could be high because of using someone else's phone number to register a new account.

The properties we are looking at when going through this test scenario are:

- **Phone registration:** Register account with a phone number.
- **E-mail registration:** Register account with an e-mail address.
- **Access SMS inbox:** Access to SMS Inbox to read the verification code.
- **Contact list upload:** Upload contacts to see if others are using the same application.
- **Verification by SMS:** Receive verification code through SMS.
- **Verification by Phone Call:** Receive verification code through a phone call.

Initial Contact

This test scenario is a part of each of the other scenarios where two users have a conversation. When Bob sends Alice a message, we look how the application handles the first message sent to the other participant. Are the participants informed of the secure messaging capabilities and does the application show how the cryptographic keys are used?

The properties we are looking at when going through this test scenario are:

- **Trust-On-First-Use:** Automatically verify each other on initiation.
- **Notification About E2E Encryption:** A notification box to explain the user that messages are end-to-end encrypted.

Message After a Key Change

This test is about how the application handles changes of cryptographic keys after Bob deletes the application in the middle of a conversation with Alice. After Bob has reinstalled his application, Alice sends him a new message and examines if the application gives Alice any information about the key changes.

When a user deletes a secure messaging application, the cryptographic keys are deleted from the device to strengthen the security of the messages the participant has already sent. When a participant then reinstalls the application, a new set of cryptographic keys are generated. The test looks if the application gives Alice any information about Bob's newly generated set of cryptographic keys, or if the conversation moves along without any notification.

The properties we are looking at when going through this test scenario are:

- **Notification about key changes:** Notifies Alice that Bob has changed cryptographic keys.
- **Blocking message:** Blocks new message from sending before Alice and Bob verify each other.

Key Change While a Message Is In Transit

Cryptographic key changes while a message is in transit is mostly the same as the test scenario before, but what happens when a message is lost before new keys are generated. Bob deletes his application without telling Alice; she then sends Bob a message, but the message is lost in transit. Does the application try to re-encrypt the message after Bob has generated new cryptographic keys or does the message get lost forever?

The properties we are looking at when going through this test scenario are:

- **Re-encrypt and send message:** Re-encrypts the message when the application finds out that the receiver has changed cryptographic keys and sends it.
- **Details about transmission of message:** Users can see the difference between sent and delivered messages.

Verification Process Between Participants

In a middle of a conversation, Alice and Bob want to verify each other that they are having a conversation with honest participants. This test looks at how the verification process works and if it is a secure and usable method of doing the verification between participants. The reason to verify the conversations' recipient is to ensure that no "man in the middle" attack has occurred.

The properties we are looking at when going through this test scenario are:

- **QR-code:** Verify each other through a QR-code.
- **Verify by Phone call:** Call each other with E2E-encrypted phone call and read keys out loud.

- **Share keys through 3rd party:** Share the keys through other applications.
- **Verified check:** Users can check later if a specific user is already verified.

Other Security Implementations

What about other security implementations in the applications? Some applications may have different ways of securing the applications from intrusion or other attacks.

The properties we are looking at when going through this test scenario are:

- **Two-step verification:** When registering after a reinstall or new device, then add a second passphrase/code which only the specific user knows.
- **Passphrase/code:** Add a passphrase/code that only the user knows and enters it to gain access to the application.
- **Screen security:** The user is not allowed to screenshot within the application.
- **Clear trusted contacts:** Clear all the contacts the user has verified, which means the user needs to verify each contact once again.
- **Delete devices from account:** If the application allows multiple devices, then there should be an option to delete devices which are not in use anymore.

5.2 Running The Different Test Cases

5.2.1 Case 1: Signal

Signal is an instant messaging application, but also a voice calling application for both Android and iOS[4]. It changed the name to Signal after Open Whisper Systems[5] decided to merge their voice calling application, RedPhone[6], with their secure messenger, TextSecure[7], in 2014. A unified application was a good way to not confuse the end user by installing multiple applications when they could only have one and integrate both of them into it.

What sets the Signal application apart from the other applications, except for Riot, is that it is completely open source and that anyone who wants to contribute can do so. This reassures people who use it that it does what they are saying that it does since anyone can come and audit the source code and the cryptographic protocol that is used.

[4]`http://www.apple.com/no/ios/ios-10/`

[5]`https://whispersystems.org/`

[6]`https://whispersystems.org/blog/low-latency-switching/`

[7]`https://en.wikipedia.org/wiki/TextSecure`

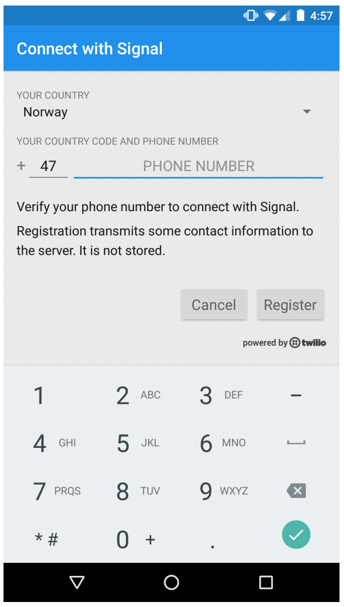
Initial Set Up

An account can only be registered to one device at a time, which means that if a user uses the same number on a second device, the first device gets deactivated automatically, for security reasons and to keep the private cryptographic keys on one device.

Figure 5.1a shows the first view a user gets when opening the app for the first time. Twilio[8] is used for the handling of SMS verification process with the Signal server when registering an account and some contact information may be transmitted to the server, but it is not stored.

Figure 5.1b explains the different steps the Signal app goes through to register and verify a new user account. The verification code is sent as an SMS, and the app reads the SMS automatically to verify the new user. After the verification, the app generates new device cryptographic keys used in conversations between participants for end-to-end encryption and at the end registers the account with the server.

If the user does not give the application access to their SMS inbox, then it has to wait for the SMS verification timer to time out, as shown at the bottom of figure Figure 5.1b. When the timer has timed out, the Signal application calls the user and gives out a verification number to be typed in manually.


4:57

Connect with Signal

YOUR COUNTRY
Norway

YOUR COUNTRY CODE AND PHONE NUMBER
+ 47 PHONE NUMBER

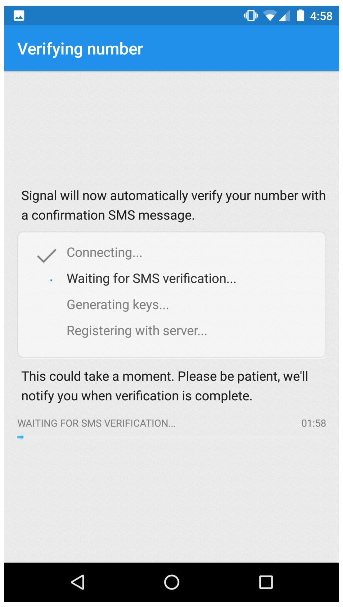
Verify your phone number to connect with Signal.

Registration transmits some contact information to the server. It is not stored.

Cancel Register

powered by twilio

(a) Phone number registration


4:58

Verifying number

Signal will now automatically verify your number with a confirmation SMS message.

Connecting...
Waiting for SMS verification...
Generating keys...
Registering with server...

This could take a moment. Please be patient, we'll notify you when verification is complete.

WAITING FOR SMS VERIFICATION... 01:58

(b) Verifying the phone number

Figure 5.1: Signal: Registration process

Message after key change

This test is about checking what happens when the cryptographic keys change a user in a conversation deletes and then reinstalls the Signal app.

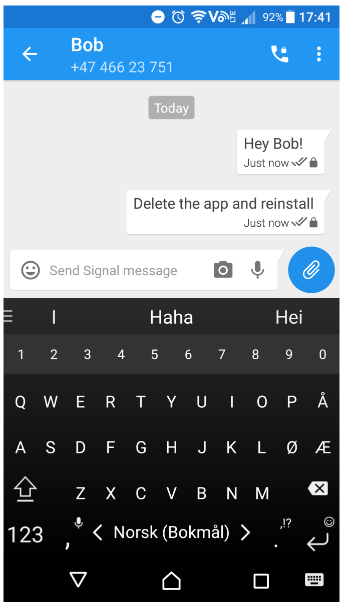
[8] https://www.twilio.com/

Figure 5.2a shows the first two messages that Alice has sent to Bob asking him to delete and reinstall the app for Bob's cryptographic keys to change. The double checkmark shown on each message in figure 5.2a, means it has been received and read by Bob. The lock on each message presents to both participants that the message is encrypted from one end to the other end and nobody in between can read it.

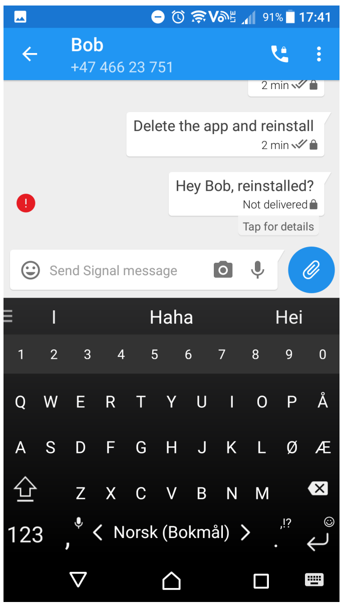
Figure 5.2b shows when Alice sends Bob another message after he has deleted and reinstalled his Signal application. The application notifies Alice that the message has not been delivered with a red notification icon on the left of the message. It also gives information to press on the message to get more details about the notification.

Figure 5.2c is the view the user gets when pressing the message that was not delivered in figure 5.2b. Alice gets presented with information that Bob has a new security number (cryptographic keys), and she needs to verify the new keys to get the ability to send Bob new messages. How the verification process is handled between Alice and Bob is shown in a later test scenario.

After the verification process between Alice and Bob is done, they can continue the conversation, and a notification gets posted in the conversation that Bob has changed his security number, as shown in figure 5.2d.

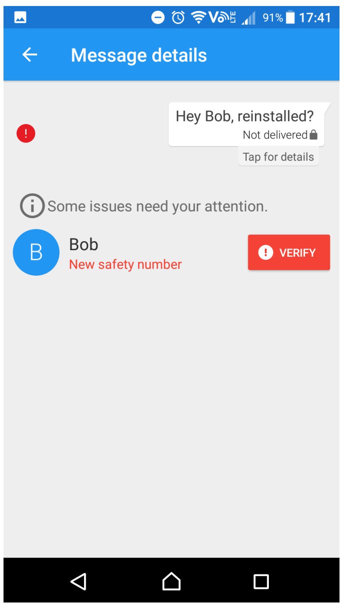

17:41
Bob
+47 466 23 751
Today
Hey Bob!
Just now
Delete the app and reinstall
Just now

(a) Alice's first message


17:41
Bob
+47 466 23 751
2 min
Delete the app and reinstall
2 min
Hey Bob, reinstalled?
Not delivered
Tap for details

(b) Message after reinstall

Figure 5.2: Signal: Message after key change

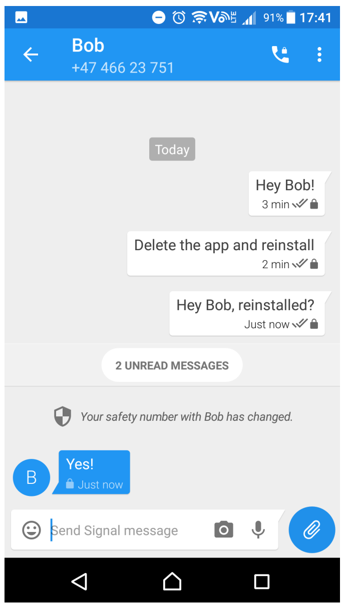

Message details

Hey Bob, reinstalled?
Not delivered
Tap for details

Some issues need your attention.

Bob
New safety number

(c) Verifying Bob again


Bob
+47 466 23 751

Today

Hey Bob!
3 min

Delete the app and reinstall
2 min

Hey Bob, reinstalled?
Just now

2 UNREAD MESSAGES

Your safety number with Bob has changed.

Yes!
Just now

(d) Message after verification

Figure 5.2: Signal: Message after key change (cont.)

Key Change While a Message Is In Transit

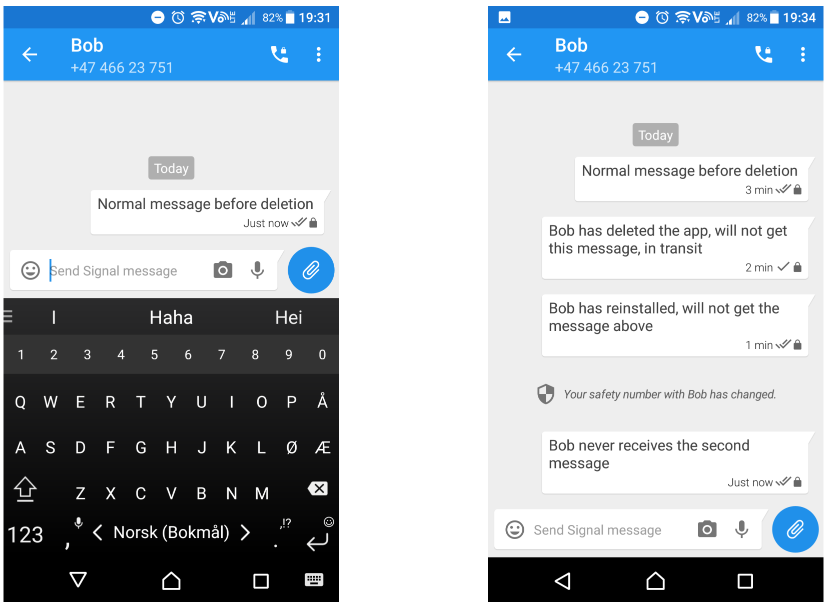
This test scenario is mostly the same as the previous test scenario, but the difference here is to check what the Signal app does when a message is sent before Bob has managed to reinstall his Signal application.

Figure 5.3a shows the initialization of the conversation between Alice and Bob, and then Alice asks Bob to delete the application to see how the Signal app handles lost messages in transit.

Figure 5.3b shows the conversation after a couple of messages from Alice to Bob. The second message is sent after Bob has deleted his application, and it shows that there is only a single checkmark on that message, which means the message has been sent, but not received by Bob.

The third message sent by Alice explains that Bob has finally reinstalled, but he never receives the second message which was sent before he reinstalled. After Alice and Bob verified their new security number between themselves, all new messages are received and encrypted by both sides, but the second message is never received.

The reason for never receiving the second message in figure 5.3b is because the Signal application never stores messages that are encrypted after they are sent to the server, and the messages are never re-encrypted by Alice when Bob has changed his cryptographic keys.



Today

Normal message before deletion
Just now

Bob
+47 466 23 751

Today

Normal message before deletion
3 min

Bob has deleted the app, will not get this message, in transit
2 min

Bob has reinstalled, will not get the message above
1 min

Your safety number with Bob has changed.

Bob never receives the second message
Just now

(a) Message before key change (b) Message after key change

Figure 5.3: Signal: Key change while message in transit

Verification Process Between Participants

Signal has some great methods of letting the users verify each other. The user has three different options on verifying the other participant.

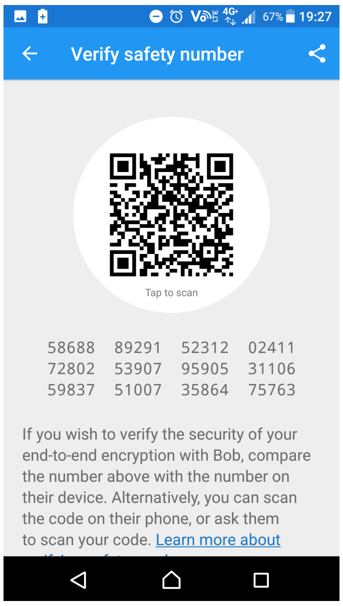
The first verification process is using the built-in calling option in Signal which is end-to-end encrypted and then read out loud to the other participant the security numbers that are shown in figure 5.4a. If the Signal calling is not secure enough for the participants, they can meet up and read the numbers out loud to each other.

The second method is using the QR-codes[9] that is shown in figure 5.4a. The Signal app has a built-in QR-code scanner as well, which means that the participants can use that to scan the other participants QR-code to verify it is the same person in the chat.

The third option to verify the other user is if the users do not trust the Signal application in handling the verification process. It is possible to share the security numbers to other applications on the user's phone. The user may have PGP[10] enabled e-mail on their phone, and they trust it more than the Signal application, then this method is a better way of verifying the other user.

[9] `http://www.qrcode.com/en/index.html`

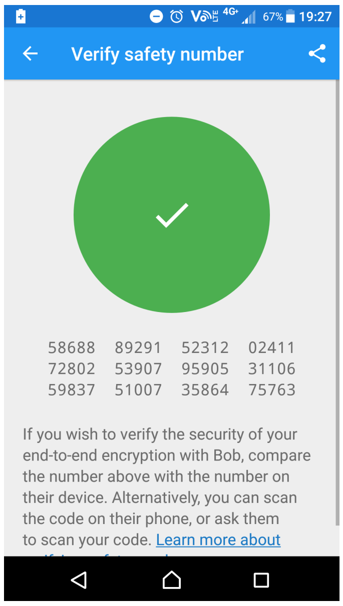
[10] `https://en.wikipedia.org/wiki/Pretty_Good_Privacy`



58688 89291 52312 02411
72802 53907 95905 31106
59837 51007 35864 75763

If you wish to verify the security of your end-to-end encryption with Bob, compare the number above with the number on their device. Alternatively, you can scan the code on their phone, or ask them to scan your code. Learn more about

(a) Verification page



58688 89291 52312 02411
72802 53907 95905 31106
59837 51007 35864 75763

If you wish to verify the security of your end-to-end encryption with Bob, compare the number above with the number on their device. Alternatively, you can scan the code on their phone, or ask them to scan your code. Learn more about

(b) Verified

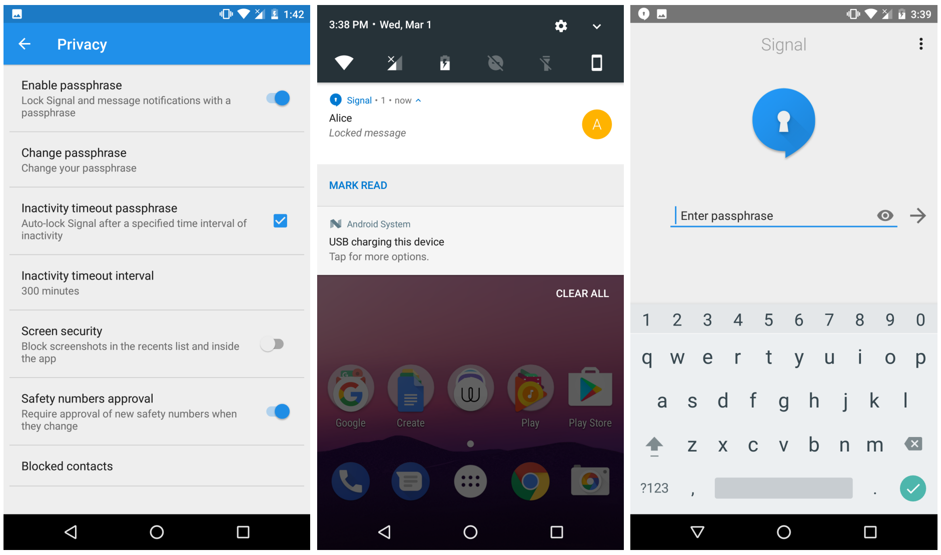
Figure 5.4: Signal: Verification process

Other Security Implementations

The Signal application has some extra privacy settings if a user feels it needs them. The first extra privacy setting is the ”Safety numbers approval” as seen in figure 5.5a. The setting is activated by default which is an important setting to have activated from the start. When a user changes it safety numbers (cryptographic keys) by deleting and reinstalling the app, it will change device keys. When the device keys change, the messages will not be shown to the user receiving the messages from the new user device, until the new safety numbers are approved.

The second privacy setting is not that important for the majority of users, but a good implementation none the less. ”Screen security” does not allow the user to screenshot as long they are inside the application.

The last privacy setting is the ability to enable a passphrase. The passphrase locks the Signal application and all message notifications with a passphrase which hinders other users than the owner of the application to get access to the notifications and Signal. It is possible to add an inactivity timeout passphrase which locks the application after some given time. Figure 5.5b shows the notification which is locked and when the user tries to open the application, the screenshot in figure 5.5c shows that the user needs to enter their passphrase they chose when the setting was activated.


Privacy

Enable passphrase
Lock Signal and message notifications with a passphrase

Change passphrase
Change your passphrase

Inactivity timeout passphrase
Auto-lock Signal after a specified time interval of inactivity

Inactivity timeout interval
300 minutes

Screen security
Block screenshots in the recents list and inside the app

Safety numbers approval
Require approval of new safety numbers when they change

Blocked contacts

3:38 PM • Wed, Mar 1

Signal • 1 • now
Alice
Locked message

MARK READ

Android System
USB charging this device
Tap for more options.

(a) Privacy settings (b) Notification locked (c) Passphrase input

Figure 5.5: Signal: Other security implementations

5.2.2 Case 2: WhatsApp

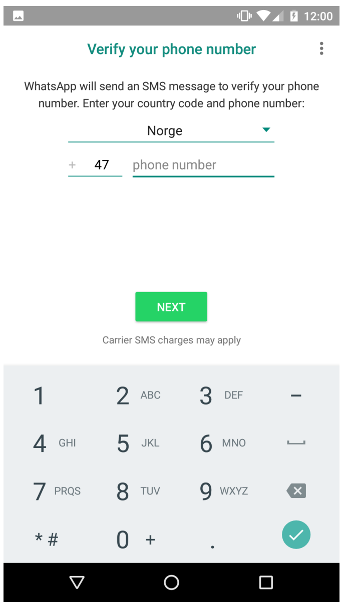
WhatsApp started as a small company in 2009, bought by Facebook in 2014 when it had 465 million monthly active users [49] and in 2017 that number has grown to 1.2 billion [49]. It started with only doing cross-platform non-secure instant messaging, but by the end of 2014 they announced that every user was going to start sending end-to-end encrypted messages using the Signal protocol [36]. This was a huge step for Open Whisper Systems who made the Signal Protocol since they would now have made the biggest instant messaging application use their protocol, and in April 2016 they made the complete transition from non-secure messaging to fully support end-to-end encryption with the Signal Protocol [37].

Initial Set Up

An account on WhatsApp works the same way as the Signal application, where the account only works on one device at a time.

Figure 5.6a shows the first page a user gets greeted with when starting the application for the first time. WhatsApp uses their infrastructure to handle the SMS verification process instead of any other 3rd party such as Twilio that Signal uses.

Figure 5.6b shows the verification page after the user has entered their phone number. WhatsApp automatically enters the verification code that is sent to the user's SMS inbox, but if the user has not given the app access to the inbox, they can enter the verification code manually. If for some reason the verification code does not arrive, the user has the option to either resend the SMS or notify WhatsApp to call the user to receive the verification code through a phone call.


12:00
Verify your phone number
WhatsApp will send an SMS message to verify your phone number. Enter your country code and phone number:
Norge
+ 47
phone number
NEXT
Carrier SMS charges may apply
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PRQS
8 TUV
9 WXYZ
* #
0 +
.
(a) Phone number registration

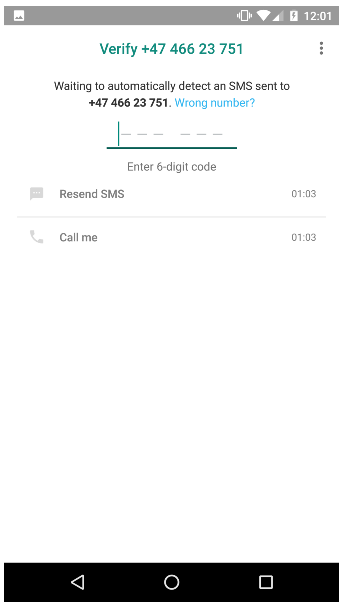

12:01
Verify +47 466 23 751
Waiting to automatically detect an SMS sent to +47 466 23 751. Wrong number?
Enter 6-digit code
Resend SMS 01:03
Call me 01:03
(b) Verifying the phone number

Figure 5.6: WhatsApp: Registration process

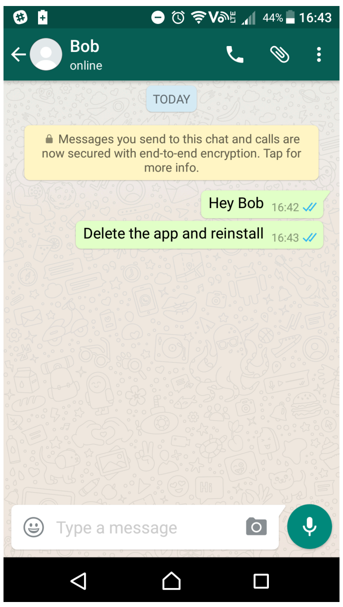
Message after key change

Figure 5.7a shows when Alice sends her first and second message to Bob, for them to initiate a conversation together. The yellow notification box at the top of the conversation is WhatsApp notifying both participants that the conversation is end-to-end encrypted and they can read more about the encryption by pressing the box. Each message is shown with a double checkmark which means that the message from Alice is received and read by Bob. Alice's second message is her asking Bob to deleted and then reinstall the application to see how WhatsApp handles new cryptographic keys.

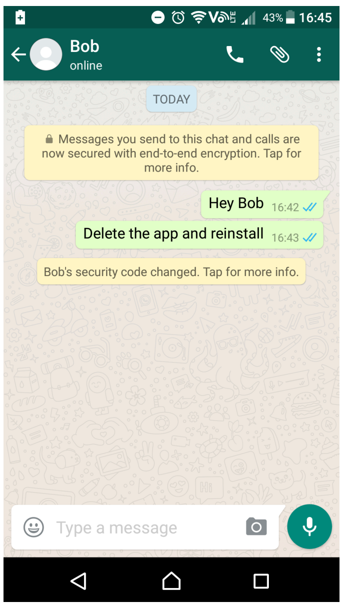
Figure 5.7b shows a new notification box has appeared on Alice's conversation page with Bob after he has reinstalled his application. WhatsApp automatically checks if new cryptographic keys (security code) are changed even though she has not sent him any message asking if he has reinstalled his application.

When Alice taps the notification box from figure 5.7b, a popup as shown in figure 5.7c, informs Alice why Bob's cryptographic keys have changed and the option to verify him before she sends him new messages. How the verification process works in WhatsApp is described in a later test scenario about the verification process.

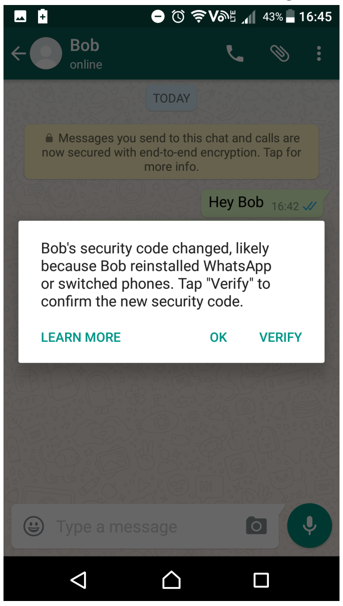
After Alice has verified Bob's new cryptographic keys, she sends him a new message and asks if he has reinstalled the application. The message has again the double checkmark that the message is received and is end-to-end encrypted.


Bob
online
TODAY
Messages you send to this chat and calls are now secured with end-to-end encryption. Tap for more info.
Hey Bob 16:42
Delete the app and reinstall 16:43
Type a message

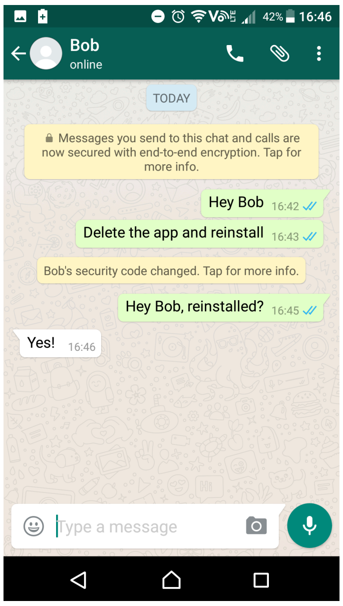
(a) Alice's first message


Bob
online
TODAY
Messages you send to this chat and calls are now secured with end-to-end encryption. Tap for more info.
Hey Bob 16:42
Delete the app and reinstall 16:43
Bob's security code changed. Tap for more info.
Type a message

(b) After Bob has reinstalled


Bob
online
TODAY
Messages you send to this chat and calls are now secured with end-to-end encryption. Tap for more info.
Hey Bob 16:42
Bob's security code changed, likely because Bob reinstalled WhatsApp or switched phones. Tap "Verify" to confirm the new security code.
LEARN MORE OK VERIFY
Type a message

(c) Info about Bob's new keys


Bob
online
TODAY
Messages you send to this chat and calls are now secured with end-to-end encryption. Tap for more info.
Hey Bob 16:42
Delete the app and reinstall 16:43
Bob's security code changed. Tap for more info.
Hey Bob, reinstalled? 16:45
Yes! 16:46
Type a message

(d) Message after verification

Figure 5.7: WhatsApp: Message after key change

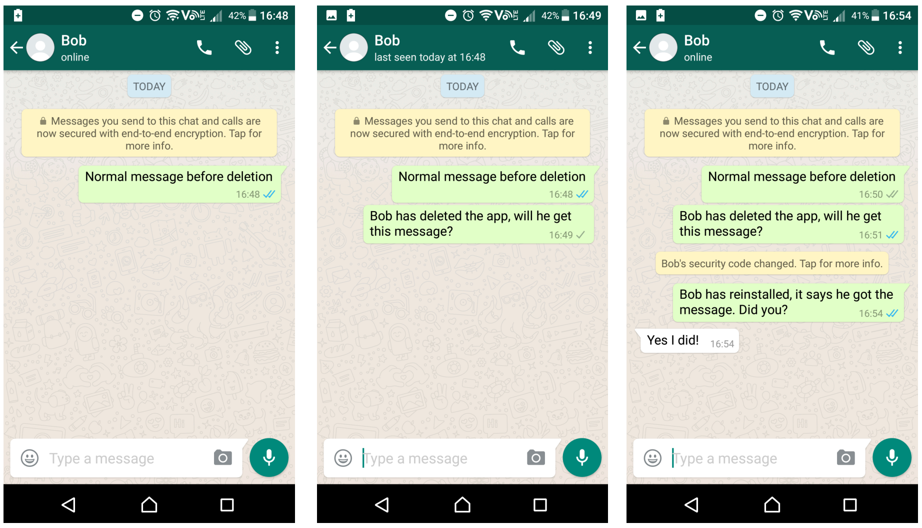
Key change while a message is in transit

This test scenario is mostly the same as the previous one, but here we look at how the WhatsApp application handles messages sent before Bob has managed to reinstall.

Figure 5.8a shows the initial message Alice sends to Bob asking him to delete the application.

Figure 5.8b shows Alice sending a second message to Bob after he has deleted his application. The single checkmark on the message means that the message has been sent, but not received and read by Bob.

When Bob reinstalls the application, both the second message Alice sent and the same yellow notification box is added to the conversation. Figure 5.8b shows the conversation after Alice sends a third message asking about her second message, if Bob actually received it without her re-encrypting and sending it a second time. Bob does receive the message which was sent before he reinstalled his application, which means that WhatsApp re-encrypts messages when the receiver has gotten new cryptographic keys, without Alice verifying the keys first.


Bob
online
TODAY
Messages you send to this chat and calls are now secured with end-to-end encryption. Tap for more info.
Normal message before deletion 16:48

Bob
last seen today at 16:48
TODAY
Messages you send to this chat and calls are now secured with end-to-end encryption. Tap for more info.
Normal message before deletion 16:48
Bob has deleted the app, will he get this message? 16:49

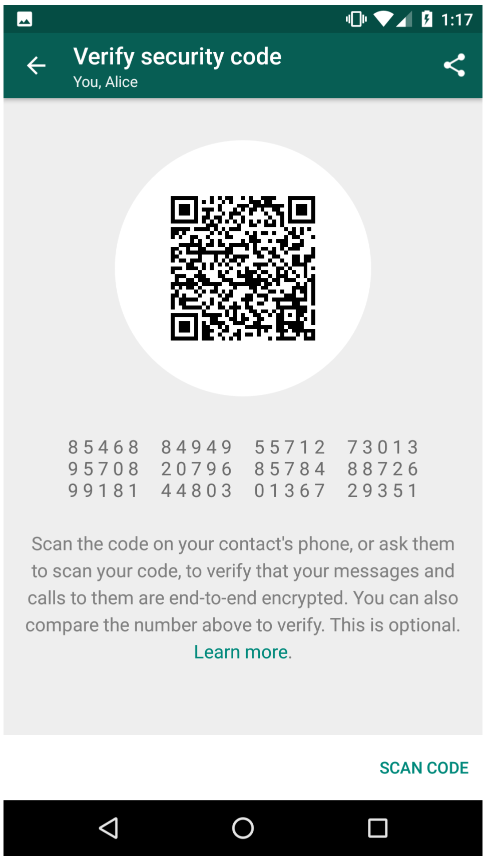
Bob
online
TODAY
Messages you send to this chat and calls are now secured with end-to-end encryption. Tap for more info.
Normal message before deletion 16:50
Bob has deleted the app, will he get this message? 16:51
Bob's security code changed. Tap for more info.
Bob has reinstalled, it says he got the message. Did you? 16:54
Yes I did! 16:54

(a) Alice's first message (b) Bob deletes his app (c) Bob reinstalled

Figure 5.8: WhatsApp: Key change while message in transit

Verification Process Between Participants

Whatsapp have implemented the same verification process as Signal has done. It uses the Signal numerical format for verification, a QR-code for scanning with the built-in scanner and the user can choose if they want to copy the security numbers outside of the Whatsapp application. The reason for this may be that when they decided to implement the Signal end-to-end security protocol, they implemented every single step of the Signal implementation to uphold the specifications. WhatsApp does also have end-to-end encrypted calling, which means that the users can call each other to read the security code and verify.


Verify security code

You, Alice

85468 84949 55712 73013
95708 20796 85784 88726
99181 44803 01367 29351

Scan the code on your contact's phone, or ask them to scan your code, to verify that your messages and calls to them are end-to-end encrypted. You can also compare the number above to verify. This is optional. Learn more.

Figure 5.9: WhatsApp: Verification process

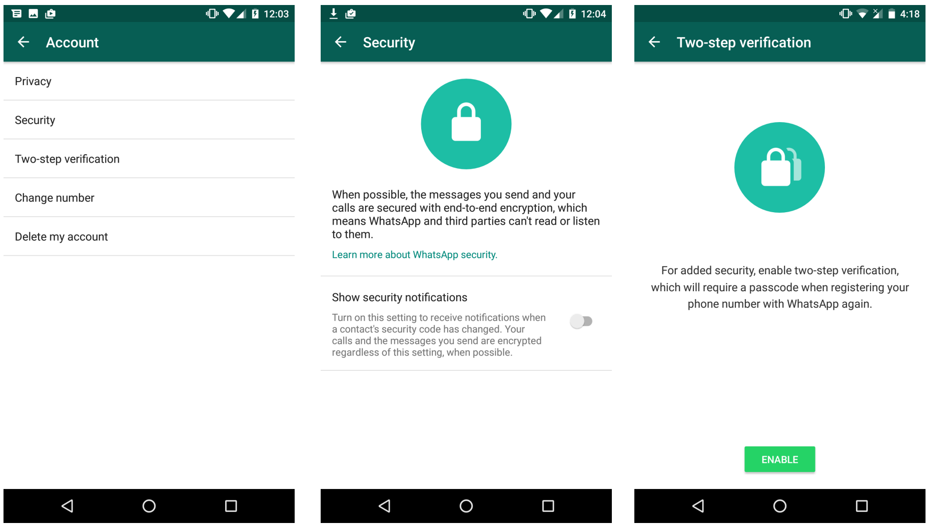
Other security implementations

WhatsA has a few different settings to strengthen the application's security.

Figure 5.10 shows the settings page for the user's account, where we can see there are settings to change the number of the account or delete the account.

When a user goes to the security page, the figure 5.10b is shown to the user. The "show security notification" option triggers a notification to Alice when Bob has either reinstalled the application or received a new device. When the option is turned off Alice does not receive any notification the way our earlier test scenarios about key changes were handled.

Figure 5.10c shows the two-step verification settings that WhatsApp has implemented, where the user needs to enter an additional passphrase when registering the account with the same number on a new device or after a fresh reinstall.


Account

Privacy

Security

Two-step verification

Change number

Delete my account

Security

When possible, the messages you send and your calls are secured with end-to-end encryption, which means WhatsApp and third parties can't read or listen to them.

Learn more about WhatsApp security.

Show security notifications

Turn on this setting to receive notifications when a contact's security code has changed. Your calls and the messages you send are encrypted regardless of this setting, when possible.

Two-step verification

For added security, enable two-step verification, which will require a passcode when registering your phone number with WhatsApp again.

ENABLE

(a) Privacy settings (b) Security notification (c) Two step verification

Figure 5.10: WhatsApp: Other security implementations

5.2.3 Case 3: Wire

Wire is an application that utilizes end-to-end encryption with the protocol Proteus, that is heavily based on the Signal Protocol, but implemented by themselves [21]. It started in 2012 by developers who previously worked at Microsoft[11] and Skype[12], and finally released their own instant messaging application in 2014 [47]. The first version did not offer end-to-end encryption between people until March 2016, when they launched the encryption on instant messaging and their video calling feature [2].

Wire offers the same features as the other applications, such as text, video, voice, photo and music messages, and does it by still providing end-to-end encrypted messages. It is supported on multiple platforms, from smartphones to personal computers, by also having every platform open sourced.[13]

Initial Set Up

The Wire app has a little different registration process than the other applications. It starts the same as the others, as can be seen in figure 5.11a, with a phone number registration. Wire allows users to register with a phone number, or an email address, which is not the same as the other applications. The latter is only supported by registering through their web application, not the phone.

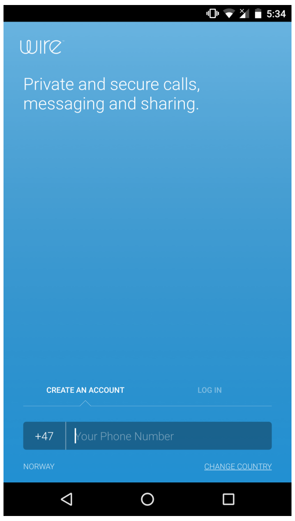
Figure 5.11b shows the verification process, which the user needs to enter the verification code manually, which they get as an SMS. If the user does not get any verification code through the SMS, they can register the

[11]https://www.microsoft.com/nb-no/
[12]https://www.skype.com/en/
[13]https://github.com/wireapp

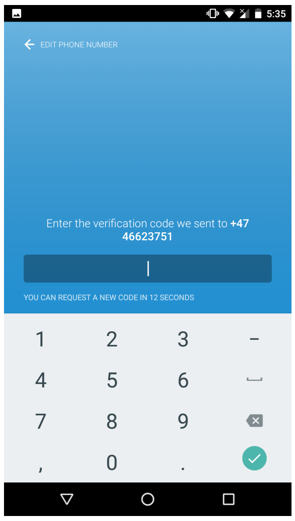
account through the Wire web application with their email address. If the user never receives the verification code, it can ask Wire to call the user to receive it, but it is not entered automatically by the application.

When a user reinstalls the application or gets a new device, they do not need to go through the registration again. Wire gives the users the option to log into the application with their phone number or email address if they have registered an email to their account profile. Users get the option to register an email and password the first time they register their phone number.

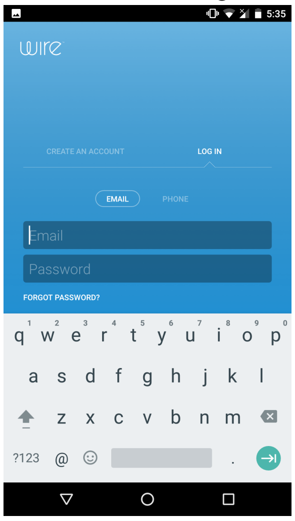
Figure 5.11b shows the option to log in with an email and password, and figure 5.11c illustrates the log in function with a phone number.


Private and secure calls, messaging and sharing.

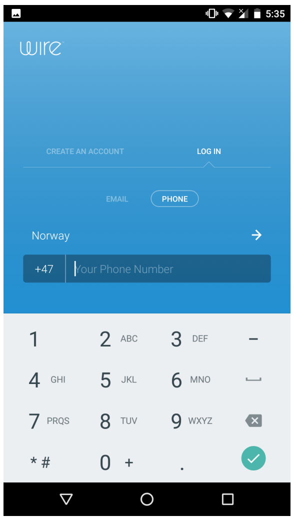
(a) Phone number registration


Enter the verification code we sent to +47 46623751
YOU CAN REQUEST A NEW CODE IN 12 SECONDS

(b) Phone verification



(c) User login with mail



(d) Login with phone number

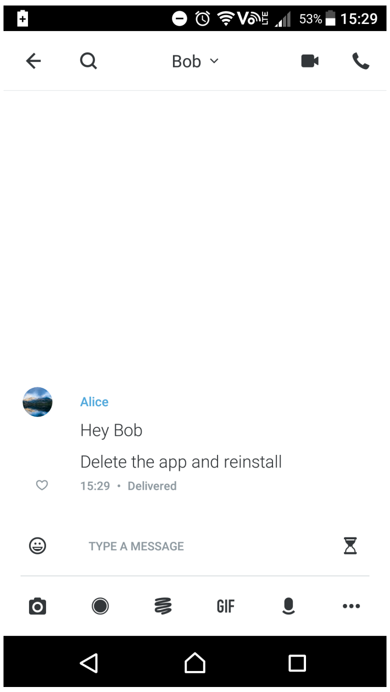
Figure 5.11: Wire: Registration process

Message after key change

Figure 5.12a shows Alice's initial contact with Bob, telling him to delete and then reinstall the application to see how Wire handles key changes. Wire uses text under each message to explain if the message was delivered to Bob.

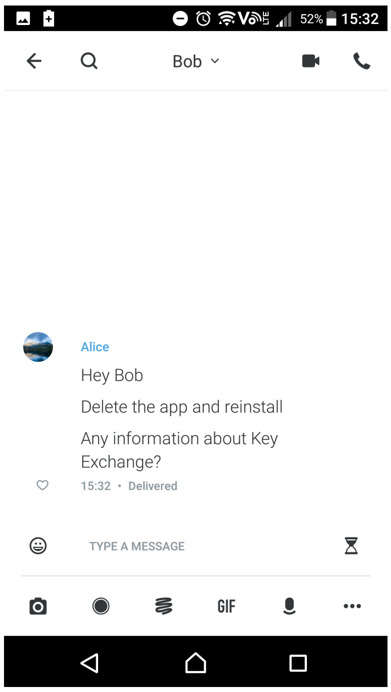
Figures 5.12b and 5.12c illustrates Alice sending a third and fourth message to Bob after he has reinstalled his application. Alice does not get any notification by Wire that Bob has gotten new cryptographic keys; it may look to Alice that Bob has the same keys as before.

Alice can check Bob's account information to see if he has got new cryptographic keys. Figure 5.12d shows Bob's device keys under his account that he has new device keys after reinstalling the application, but Wire does not give any information to Alice about this. There are three different device keys under his account because Wire allows for multiple devices to one account, which means that Alice needs to verify each device to know that the conversation is secure with end-to-end encryption. The two top devices have a full blue shield which means they are verified, while the bottom device is only has a half full shield because it has not been verified yet.



Alice
Hey Bob
Delete the app and reinstall
15:29 • Delivered

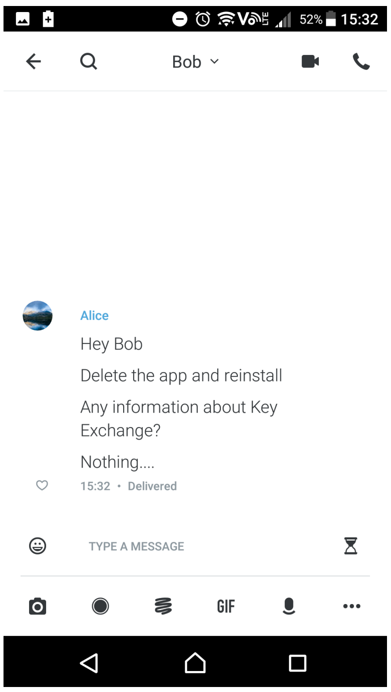
(a) Alice's first message



Alice
Hey Bob
Delete the app and reinstall
Any information about Key Exchange?
15:32 • Delivered

(b) After Bob has reinstalled

Figure 5.12: Wire: Message after key change


Alice

Hey Bob

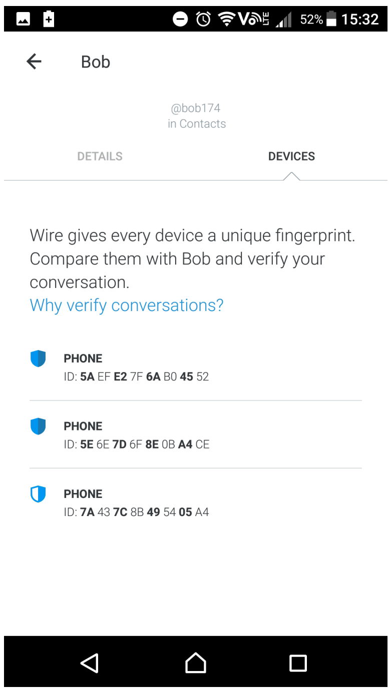
Delete the app and reinstall

Any information about Key Exchange?

Nothing....

15:32 • Delivered

(c) No notification about key change



Wire gives every device a unique fingerprint. Compare them with Bob and verify your conversation.

Why verify conversations?

PHONE
ID: 5A EF E2 7F 6A B0 45 52

PHONE
ID: 5E 6E 7D 6F 8E 0B A4 CE

PHONE
ID: 7A 43 7C 8B 49 54 05 A4

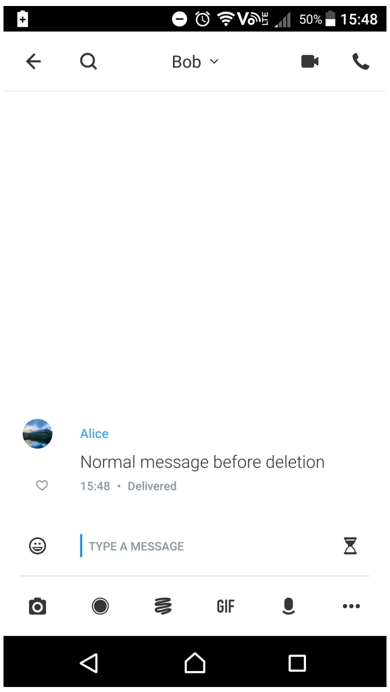
(d) Bob's device keys

Figure 5.12: Wire: Message after key change (cont.)

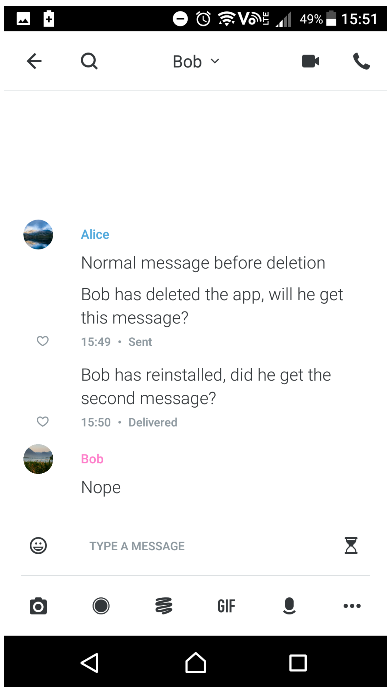
Key change while a message is in transit

Figure 5.13a shows the initial message from Alice to Bob telling him to delete the message to see how Wire handles messages lost in transit.

Figure 5.13a is after Alice has sent a few messages to Bob to see if lost messages are sent after Bob has reinstalled or if they are lost. The second message is the message before Bob has reinstalled, showing that the message is only "sent" and not delivered as the third message which was sent after Bob had reinstalled. This shows that Wire does not notify Alice about Bob's new keys, and at the same time does not deliver messages with old cryptographic keys to devices with new keys.



(a) Alice's first message before deletion of Bobs app


49% 15:51
Bob
Alice
Normal message before deletion
Bob has deleted the app, will he get this message?
15:49 • Sent
Bob has reinstalled, did he get the second message?
15:50 • Delivered
Bob
Nope
TYPE A MESSAGE
GIF

(b) No notification after Bob had reinstalled

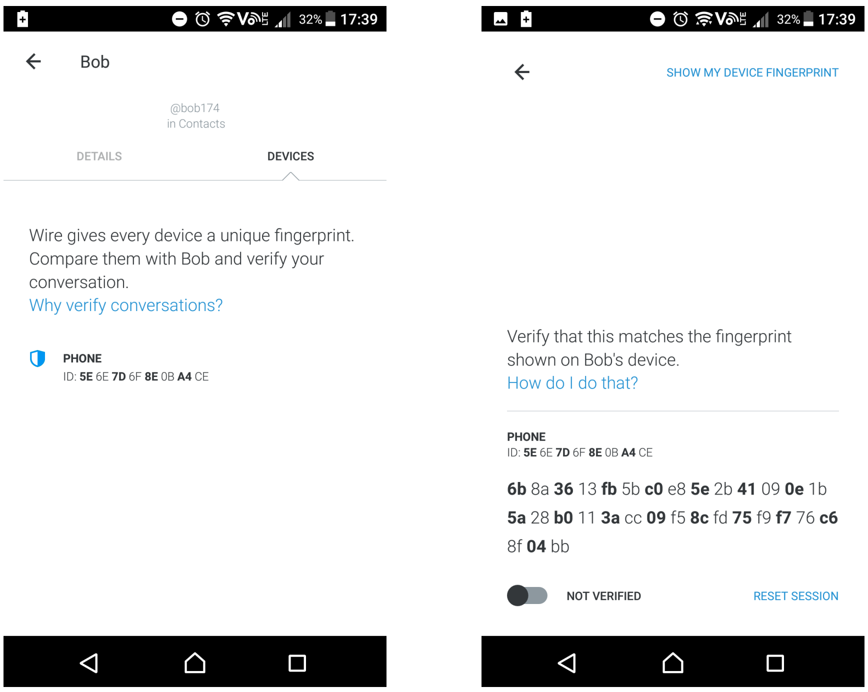
Figure 5.13: Wire: Key change while message in transit

Verification Process Between Participants

Wire's verification process does not have the same options for verifying each participant as the other applications, but it does have a useful option after the verification is done.

When Alice wants to verify one of Bob's devices, she goes into Bob's profile, and there is a dedicated tab for all of his devices. Figure 5.14a shows Bob's list of all his devices, where he at the beginning of the testing only has one set of device keys.

Figure 5.14b shows the pager after Alice taps on one of the devices from the list. Here it is possible to see the phone's ID number and the public device keys. Alice can either call Bob and verify over the phone that the keys are Bob's keys, or meet up with him in person to confirm the keys. When the verification is done, Alice needs to toggle the "not verified" switch to know that this particular device is verified.


17:39 32%

Bob

@bob174
in Contacts

DETAILS DEVICES

Wire gives every device a unique fingerprint. Compare them with Bob and verify your conversation.
Why verify conversations?

PHONE
ID: **5E** 6E **7D** 6F **8E** 0B **A4** CE

17:39 32%

SHOW MY DEVICE FINGERPRINT

Verify that this matches the fingerprint shown on Bob's device.
How do I do that?

PHONE
ID: **5E** 6E **7D** 6F **8E** 0B **A4** CE

6b 8a **36** 13 **fb** 5b **c0** e8 **5e** 2b **41** 09 **0e** 1b
5a 28 **b0** 11 **3a** cc **09** f5 **8c** fd **75** f9 **f7** 76 **c6**
8f **04** bb

NOT VERIFIED RESET SESSION

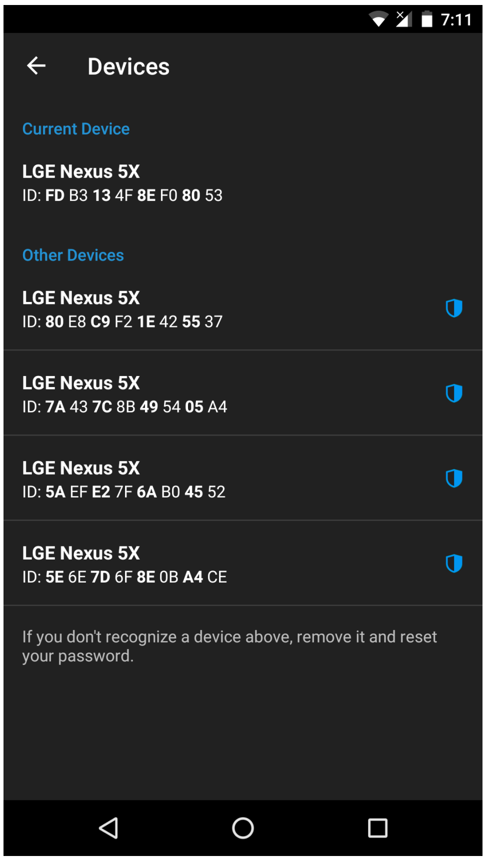
(a) List of Bob's devices (b) Bob's public keys for one device

Figure 5.14: Wire: Verification process

Other security implementations

Wire does not have any extra security implementations such as Signal or WhatsApp. The application does have some options inside the settings to change how the message conversation looks or adding email to the account for easier login.

The user can look at the devices which have been used with the account(5.15), and if there are any devices which the user does not own or recognize, they can delete the specific device. After the user has deleted a specific device, they are prompted to change the password for that account, in case someone has managed to compromise it.


7:11

Devices

Current Device

LGE Nexus 5X
ID: **FD** B3 **13** 4F **8E** F0 **80** 53

Other Devices

LGE Nexus 5X
ID: **80** E8 **C9** F2 **1E** 42 **55** 37

LGE Nexus 5X
ID: **7A** 43 **7C** 8B **49** 54 **05** A4

LGE Nexus 5X
ID: **5A** EF **E2** 7F **6A** B0 **45** 52

LGE Nexus 5X
ID: **5E** 6E **7D** 6F **8E** 0B **A4** CE

If you don't recognize a device above, remove it and reset your password.

Figure 5.15: Wire: Other security implementations

5.2.4 Case 4: Viber

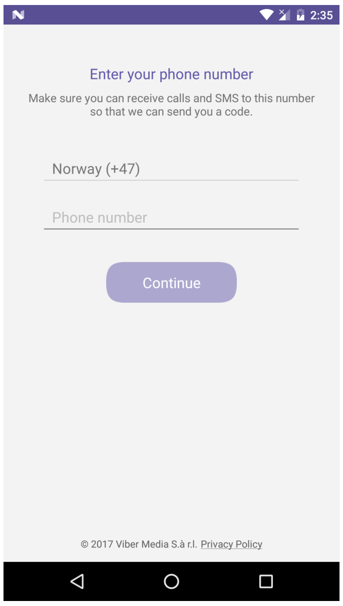
Viber is yet another instant messaging application that launched in 2010, and has managed to become quite popular throughout the world with its 800 million overall users and 266 million monthly active users [30]. It has the same properties as the other applications, where users are capable of forming groups, send messages, call each other and sending pictures, videos or voice messages to other users of Viber [29]. It works on smartphones and personal computers, which means it is cross-platform.

Viber did not have end-to-end encryption in the beginning, but implemented it on April 2016, for both one-to-one and group conversations [41]. It does not use the Signal Protocol, but they have stated it has the same concepts of the “double-ratchet” protocol used in Signal, but rather implemented their own implementation from scratch [41].

Initial Set Up

The user registration process on Viber is the same as the other applications we have looked through in the previous test cases. Figure 5.16a shows the user input for the user’s phone number for the registration.

Figure 5.16b shows the activation process of the user account. The user can either give Viber access to the SMS inbox to enter the verification code automatically or do it manually if the user does not want to give access. If the SMS with the verification code does not arrive within a minute, the user can ask the application to either resend a new verification code or get the code through a phone call.


2:35

Enter your phone number

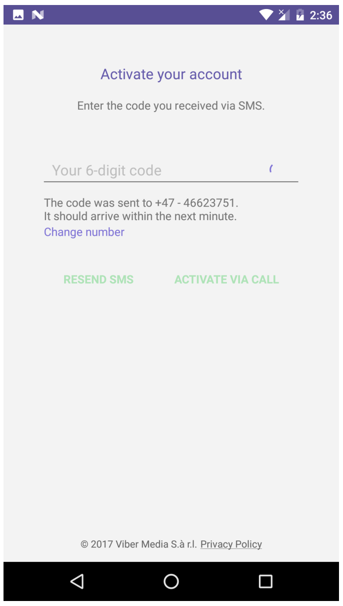
Make sure you can receive calls and SMS to this number so that we can send you a code.

Norway (+47)

Phone number

© 2017 Viber Media S.à r.l. Privacy Policy

(a) Registration with phone number


2:36

Activate your account

Enter the code you received via SMS.

Your 6-digit code

The code was sent to +47 - 46623751.
It should arrive within the next minute.

© 2017 Viber Media S.à r.l. Privacy Policy

(b) Verification of phone number

Figure 5.16: Viber: Registration process

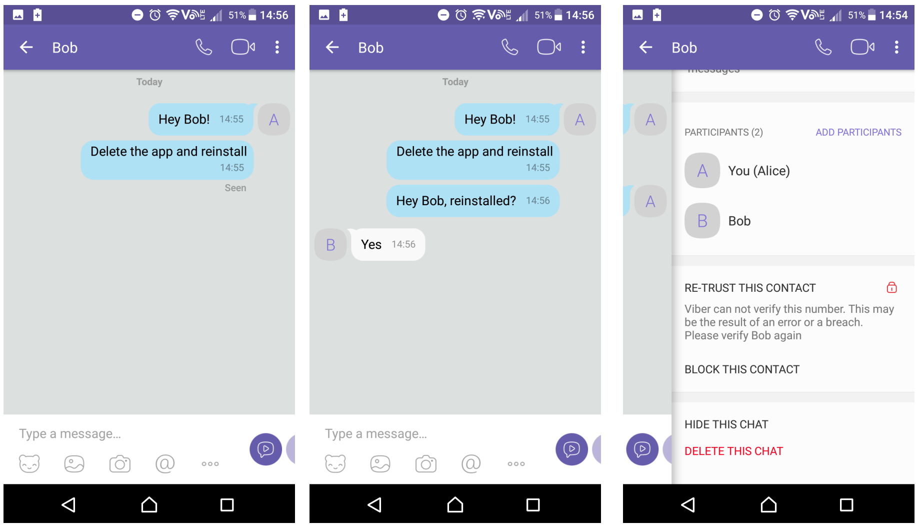
Message after key change

Viber does not notify either Alice or Bob when the cryptographic keys change during a conversation. After a few tests, the only way of knowing if the cryptographic keys changed was if the participants within a conversation verified each other first and then started testing the scenario. The verification process is showed in a later test scenario.

Figure 5.17a shows Alice initiating the conversation between her and Bob, and asks him to delete the application and then reinstall to see if any notification is posted within the conversation. The last message Alice has sent is shown as "seen" if Bob has received and read the message. Viber does not give any other notification if the message is sent or not, only if the message is read.

After Bob has reinstalled the application, Alice sends him a third message asking if he has reinstalled (5.17b). Viber does not give any notification to Alice that Bob has generated new cryptographic keys when he answers.

The only way for Alice to know that Bob has new cryptographic keys is to check the details of the conversation by swiping from right to left (5.17c) and then check if "Trust this contact" tab has changed to "Re-trust this contact". Alice needs to re-verify Bob if she wants to make sure the conversation is end-to-end encrypted.



Hey Bob! 14:55
Delete the app and reinstall 14:55
Seen

Hey Bob! 14:55
Delete the app and reinstall 14:55
Hey Bob, reinstalled? 14:56
Yes 14:56

You (Alice)
Bob
RE-TRUST THIS CONTACT
Viber can not verify this number. This may be the result of an error or a breach. Please verify Bob again
BLOCK THIS CONTACT
HIDE THIS CHAT
DELETE THIS CHAT

(a) Alice's initial message to Bob (b) No notification about key changes (c) Bob needs to be re-trusted

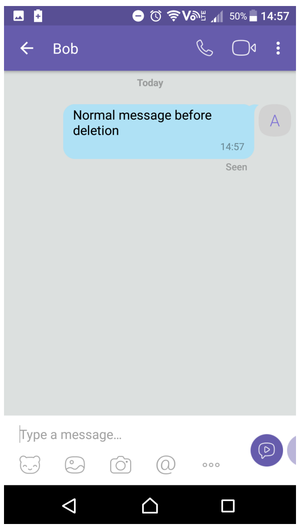
Figure 5.17: Viber: Message after key change

Key change while a message is in transit

Key changes in transit are handled the same way as key changes after the reinstall of the application, which we went through in the last test scenario. Alice initiates the conversation by sending a message to Bob right before he deletes the application (5.18a).

Figure 5.18b shows Alice sending a second message to Bob before he has reinstalled his application, and there is no information given to Alice if the message is sent or read by Bob.

Figure 5.18c shows when Alice sends her third message to Bob after he is done reinstalling his application. Alice never receives any notification by Viber that Bob has new cryptographic keys or that he has not received the second message Alice sent to Bob. There is no information on the second message if the message is sent, and Viber does not re-encrypt messages and re-sends them later on.

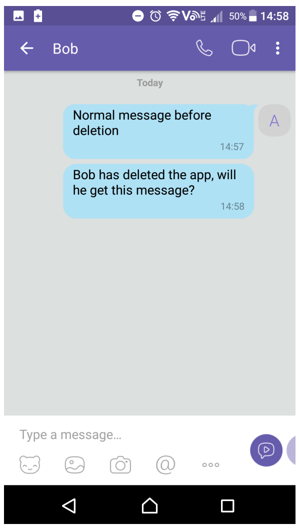

Bob

Today

Normal message before deletion 14:57

Seen

(a) Alice's initial message to Bob

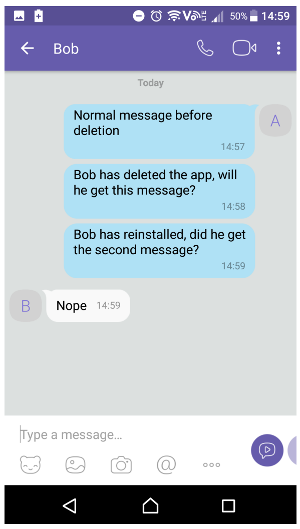

Bob

Today

Normal message before deletion 14:57

Bob has deleted the app, will he get this message? 14:58

(b) Message after Bob has deleted


Bob

Today

Normal message before deletion 14:57

Bob has deleted the app, will he get this message? 14:58

Bob has reinstalled, did he get the second message? 14:59

Nope 14:59

(c) Bob did not get the second message

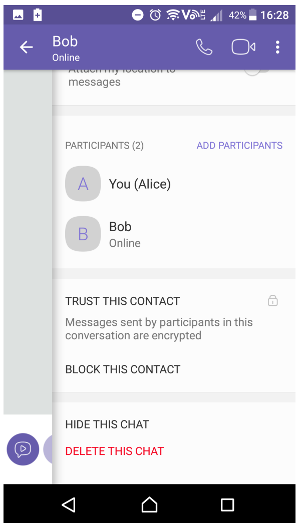
Figure 5.18: Viber: Key change while message in transit

Verification Process Between Participants

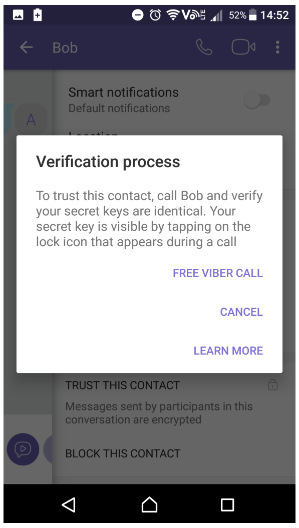
The verification process of verifying a contact in Viber is quite straight forward. If Alice wants to verify Bob she needs to go to one of their conversations, swipes from left to right to get the information tab and then go down to the "Trust this contact" option (5.19a).

Figure 5.19b shows the popup notification box after Alice clicks on the "Trust this contact" option. The only verification option Alice can use to verify Bob is by calling Bob and then read the cryptographic keys over the phone.

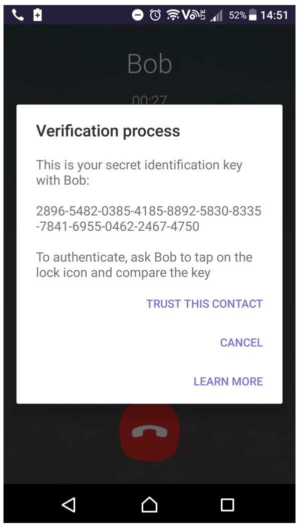
Figure 5.19c shows when Alice calls Bob and wants to verify, the popup message shows the cryptographic keys both Alice and Bob share. When they have verified each other, they press the "Trust this contact" button, and they are then verified.


Bob
Online
messages
PARTICIPANTS (2) ADD PARTICIPANTS
You (Alice)
Bob
Online
TRUST THIS CONTACT
Messages sent by participants in this conversation are encrypted
BLOCK THIS CONTACT
HIDE THIS CHAT
DELETE THIS CHAT

(a) Conversation info


Smart notifications
Default notifications
Verification process
To trust this contact, call Bob and verify your secret keys are identical. Your secret key is visible by tapping on the lock icon that appears during a call
FREE VIBER CALL
CANCEL
LEARN MORE
TRUST THIS CONTACT
Messages sent by participants in this conversation are encrypted
BLOCK THIS CONTACT

(b) Verify a contact

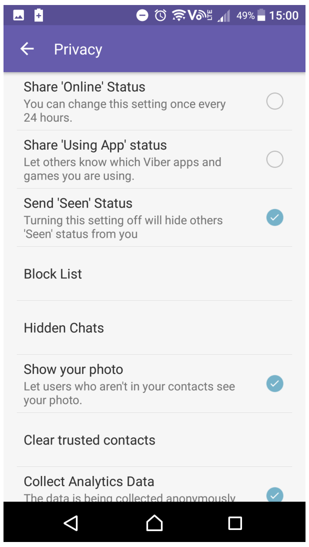

Bob
Verification process
This is your secret identification key with Bob:
2896-5482-0385-4185-8892-5830-8335
-7841-6955-0462-2467-4750
To authenticate, ask Bob to tap on the lock icon and compare the key
TRUST THIS CONTACT
CANCEL
LEARN MORE

(c) Alice verifying Bob

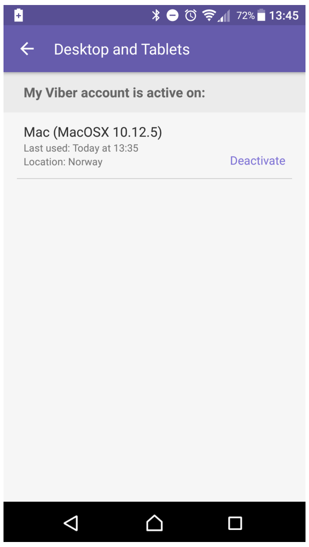
Figure 5.19: Viber: Verification process

Other security implementations

Viber does not have any extra security implementations in their application. Figure 5.20 shows the privacy settings page where the only security implementation is the “Clear trusted contacts” which clears all the contacts the Alice or Bob have verified throughout the time they have had an account.


Privacy
Share 'Online' Status
You can change this setting once every 24 hours.
Share 'Using App' status
Let others know which Viber apps and games you are using.
Send 'Seen' Status ✓
Turning this setting off will hide others 'Seen' status from you
Block List
Hidden Chats
Show your photo ✓
Let users who aren't in your contacts see your photo.
Clear trusted contacts
Collect Analytics Data ✓
The data is being collected anonymously

(a) Viber: Privacy settings


Desktop and Tablets
My Viber account is active on:
Mac (MacOSX 10.12.5)
Last used: Today at 13:35
Location: Norway
Deactivate

(b) Viber: Multiple Device List

Figure 5.20: Viber: Other security implementations

5.2.5 Case 5: Riot

Riot is a new chat client that does not utilize the Signal Protocol directly for its end-to-end encrypted capabilities, but it is built on top of Matrix[14]. Matrix is an open network for secure, decentralized communication platform which uses bridged networks and cross-platform possibilities plus full end-to-end encryption that is based on the Double Ratchet Algorithm from the Signal Protocol.

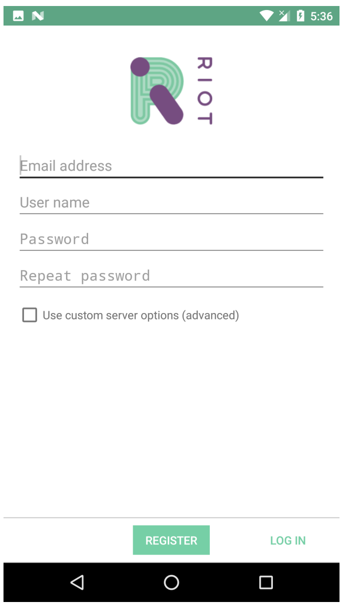
The benefit with Riot is you do not need to rely on servers by the Matrix team, the way Signal works. Riot and Matrix is open source, which means that anyone can set up their own servers with the Matrix implementation and use its end-to-end encryption. This is an excellent way for companies who want to have secure chats between employees and not have anything to do with others outside their own network. Riot does also have the same capabilities as other instant messaging applications, such as group chat, VoIP and video calling, file transfer and integrations with other applications such as Slack[15] or IRC[16] for a better and more seamless integration with others.

Initial Set Up

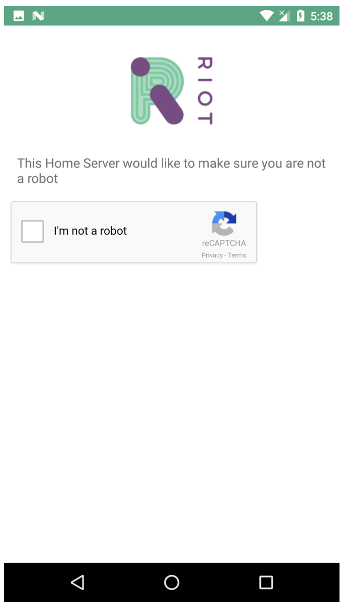
The Riot application is the only messaging client we test that does not rely on a phone number, but a user registers an account with an email and username (5.21a).

When a user registers through the app, they are instructed to check their email to continue with the registration, because Riot sends a confirmation link which the user needs to click. Figure 5.21b shows after the user has clicked the confirmation link and are then presented with a captcha verification for an extra layer of security.

[14]https://matrix.org/
[15]https://slack.com/
[16]https://www.wikiwand.com/en/Internet_Relay_Chat



(a) Registration by email



This Home Server would like to make sure you are not a robot

(b) Captcha code

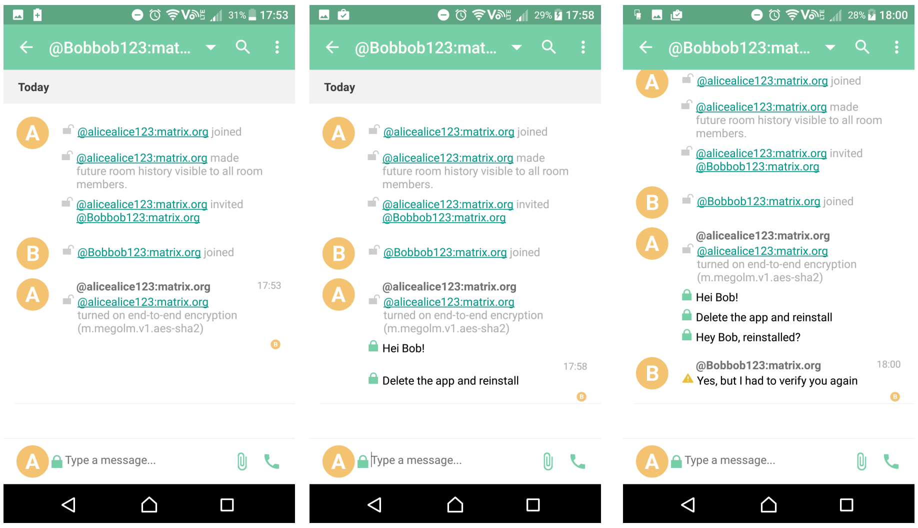
Figure 5.21: Riot: Registration process

Message after key change

Riot is not the typical instant messaging application such as Signal or WhatsApp. Their vision is to make an application which works the same way as Slack or IRC, where there are chat rooms to join and talk to others.

Alice starts a chat room, invites Bob and then activates end-to-end encryption. It is not on by default because the end-to-end encryption is still in beta and will be toggled on by default when it is out of the beta. Figure 5.22a shows the chat room, and as one can see, there are some open locks on each of the messages from Alice and Bob in the beginning. This is because those are sent before the encryption was toggled on and how the end-to-end encryption is toggled on is shown in the test scenario about other security implementations. When Alice sends her initial message to Bob about deleting the app and reinstalling it, as shown in figure 5.22b, the lock is changed to closed since the end-to-end encryption is on.

Figure 5.22c shows that Alice sends her third message to Bob and asks if he has reinstalled, and Bob answers that he has reinstalled, but he had to re-verify Alice because his device keys were changed during the reinstall. Alice can also see that she has to re-verify Bob because his message has a yellow notification triangle showing that his new keys have not been verified and should not be trusted until that is done.


17:53
@Bobbob123:mat...
Today
@alicealice123:matrix.org joined
@alicealice123:matrix.org made future room history visible to all room members.
@alicealice123:matrix.org invited @Bobbob123:matrix.org
@Bobbob123:matrix.org joined
@alicealice123:matrix.org 17:53
@alicealice123:matrix.org turned on end-to-end encryption (m.megolm.v1.aes-sha2)
Type a message...

17:58
@Bobbob123:mat...
Today
@alicealice123:matrix.org joined
@alicealice123:matrix.org made future room history visible to all room members.
@alicealice123:matrix.org invited @Bobbob123:matrix.org
@Bobbob123:matrix.org joined
@alicealice123:matrix.org
@alicealice123:matrix.org turned on end-to-end encryption (m.megolm.v1.aes-sha2)
Hei Bob!
17:58
Delete the app and reinstall
Type a message...

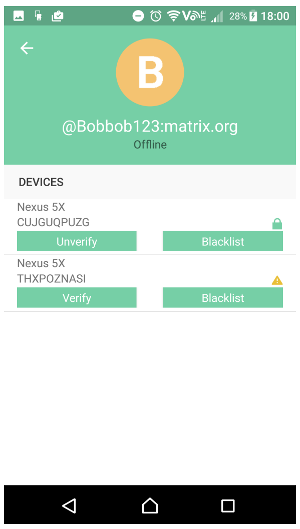
18:00
@Bobbob123:mat...
@alicealice123:matrix.org joined
@alicealice123:matrix.org made future room history visible to all room members.
@alicealice123:matrix.org invited @Bobbob123:matrix.org
@Bobbob123:matrix.org joined
@alicealice123:matrix.org
@alicealice123:matrix.org turned on end-to-end encryption (m.megolm.v1.aes-sha2)
Hei Bob!
Delete the app and reinstall
Hey Bob, reinstalled?
@Bobbob123:matrix.org 18:00
Yes, but I had to verify you again
Type a message...

(a) Initial conversation, Alice and Bob (b) Alice's initial message connecting to Bob (c) Message to Bob after he reinstalled

Figure 5.22: Riot: Message after key change

Alice needs to verify Bob's new device keys, and she can do that by going into his profile account and look at the devices he has. Figure 5.23a shows how the devices are listed on Bob's profile account, where the yellow notification triangle is there to illustrate which devices are not yet verified. How Alice verifies Bob will be shown in a later test scenario. After Alice verifies Bob, the messages are then listed with a correct closed lock, which entails that the messages are encrypted correctly (5.23b).

When a new user or a previous user, but with new keys, enters the chat room, there should not be a possibility of reading previously sent messages. Riot does this correctly, by not showing the previous messages when Bob enters the room after he reinstalled the application, but it does show him that there have been some messages exchanged between Alice and Bob earlier (5.23c).


28% 18:00

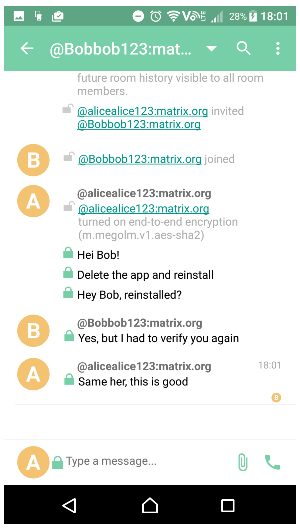
@Bobbob123:matrix.org
Offline

DEVICES

Nexus 5X
CUJGUQPUZG
Unverify | Blacklist

Nexus 5X
THXPOZNASI
Verify | Blacklist

(a) Verifying Bob the 2nd time


28% 18:01

@Bobbob123:mat...

future room history visible to all room members.

@alicealice123:matrix.org invited @Bobbob123:matrix.org

@Bobbob123:matrix.org joined

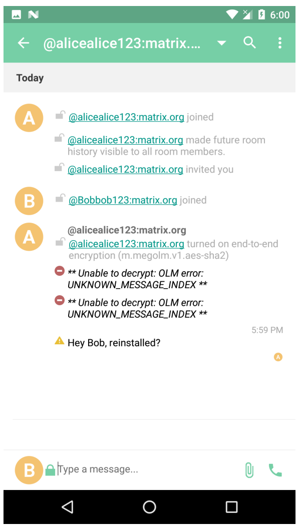
@alicealice123:matrix.org
@alicealice123:matrix.org turned on end-to-end encryption (m.megolm.v1.aes-sha2)
Hei Bob!
Delete the app and reinstall
Hey Bob, reinstalled?

@Bobbob123:matrix.org
Yes, but I had to verify you again

@alicealice123:matrix.org 18:01
Same her, this is good

Type a message...

(b) New messages are verified


6:00

@alicealice123:matrix....

Today

@alicealice123:matrix.org joined
@alicealice123:matrix.org made future room history visible to all room members.
@alicealice123:matrix.org invited you

@Bobbob123:matrix.org joined

@alicealice123:matrix.org
@alicealice123:matrix.org turned on end-to-end encryption (m.megolm.v1.aes-sha2)
*** Unable to decrypt: OLM error: UNKNOWN_MESSAGE_INDEX ***
*** Unable to decrypt: OLM error: UNKNOWN_MESSAGE_INDEX ***
5:59 PM
Hey Bob, reinstalled?

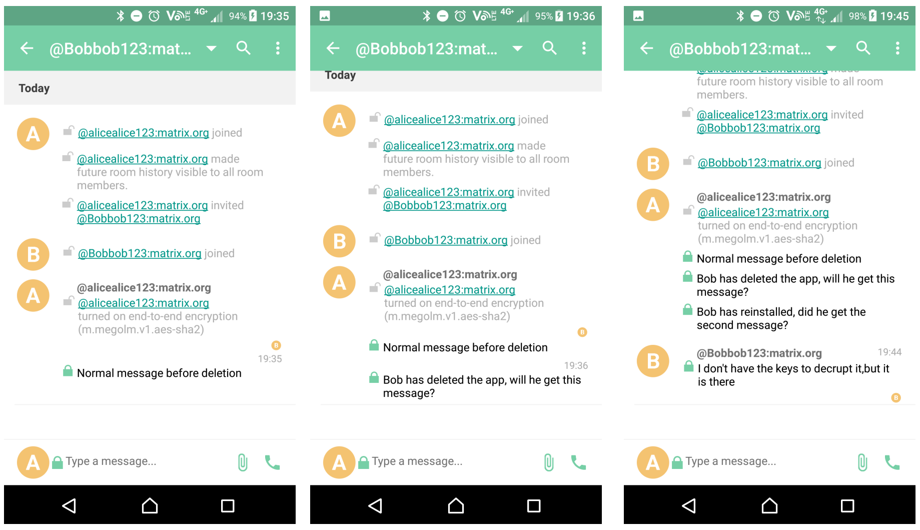
Type a message...

(c) Bob's view of previous messages

Figure 5.23: Riot: Message after key change (cont.)

Key change while a message is in transit

Riot handles key changes the same way if it is in transit or not. Figure 5.24a shows Alice's initial message before Bob deletes his Riot application. After Bob has uninstalled, Alice sends her second message (5.24b) to him to see what happens after he has reinstalled; if he can read or at least see it on his chat conversation. The last figure, 5.24c, shows after Bob has reinstalled the application and Alice sent her third message to ask if he could read the second message. Bob responds with that he could not read the message since it showed the same as with the last test, he could not decrypt them. This shows that Riot handles the key changes the same way as before, Bob can see there were some messages sent, but could not decrypt since he had new encryption keys and lost the old ones.


@Bobbob123:mat...

Today

@alicealice123:matrix.org joined

@alicealice123:matrix.org made future room history visible to all room members.

@alicealice123:matrix.org invited @Bobbob123:matrix.org

@Bobbob123:matrix.org joined

@alicealice123:matrix.org

@alicealice123:matrix.org turned on end-to-end encryption (m.megolm.v1.aes-sha2)

19:35

Normal message before deletion

Type a message...

(a) Alice's initial Message to Bob

@Bobbob123:mat...

Today

@alicealice123:matrix.org joined

@alicealice123:matrix.org made future room history visible to all room members.

@alicealice123:matrix.org invited @Bobbob123:matrix.org

@Bobbob123:matrix.org joined

@alicealice123:matrix.org

@alicealice123:matrix.org turned on end-to-end encryption (m.megolm.v1.aes-sha2)

Normal message before deletion

19:36

Bob has deleted the app, will he get this message?

Type a message...

(b) Message before Bob re-installed

@Bobbob123:mat...

future room history visible to all room members.

@alicealice123:matrix.org invited @Bobbob123:matrix.org

@Bobbob123:matrix.org joined

@alicealice123:matrix.org

@alicealice123:matrix.org turned on end-to-end encryption (m.megolm.v1.aes-sha2)

Normal message before deletion

Bob has deleted the app, will he get this message?

Bob has reinstalled, did he get the second message?

@Bobbob123:matrix.org 19:44

I don't have the keys to decrupt it,but it is there

Type a message...

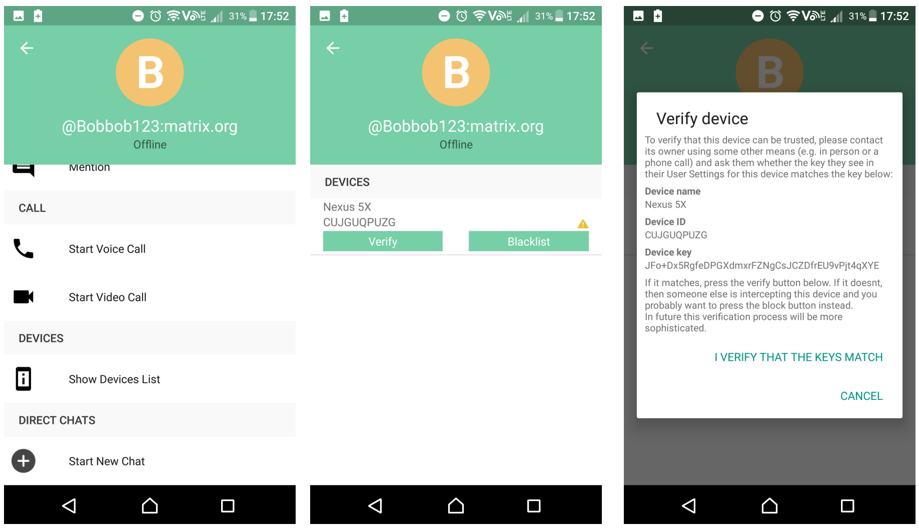
(c) Bob's message after re-install

Figure 5.24: Riot: Key change while message in transit

Verification Process Between Participants

The verification process is quite easy on the Riot application, and they give lots of information to the user about the users they interact with. When Alice wants to verify Bob's devices, she needs to look at his profile account to find Bob's list of devices to verify, by clicking on the "Device" tab as shown in figure 5.25a. Alice gets a list of Bob's devices in figure 5.25b, and if one of the devices has the yellow notification triangle, then that specific device has not been verified earlier by Alice. She can either verify the device or blacklist it, which means she does not receive any messages or invites from that specific device.

Alice decides to verify Bob's device, clicks on the verify button and gets a verification popup (5.25c). The popup is informative for users, but may have a little too much information, and could look cluttered for end-users. It is stated on the popup that the verification information and process will be more sophisticated later when the application starts to reach the end of the beta period. For Alice to verify Bob, she would need to either call Bob and exchange the device keys, or meet in public to exchange them there and in the end press the "I verify that the keys match" button.


17:52

@Bobbob123:matrix.org
Offline
Mention
CALL
Start Voice Call
Start Video Call
DEVICES
Show Devices List
DIRECT CHATS
Start New Chat

(a) Profile details

17:52

@Bobbob123:matrix.org
Offline
DEVICES
Nexus 5X
CUJGUQPUZG
Verify
Blacklist

(b) List of Bob's devices

17:52

Verify device

To verify that this device can be trusted, please contact its owner using some other means (e.g. in person or a phone call) and ask them whether the key they see in their User Settings for this device matches the key below:

Device name
Nexus 5X

Device ID
CUJGUQPUZG

Device key
JFo+Dx5RgfeDPGXdmxrFZNgCsJCZDfrEU9vPjt4qXYE

If it matches, press the verify button below. If it doesnt, then someone else is intercepting this device and you probably want to press the block button instead.
In future this verification process will be more sophisticated.

I VERIFY THAT THE KEYS MATCH

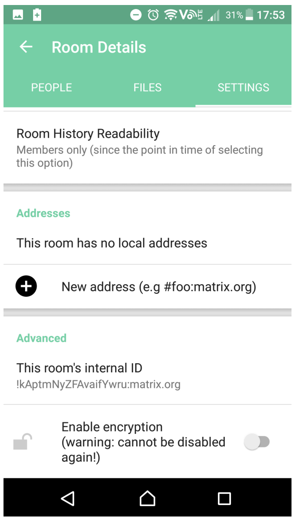
CANCEL

(c) Verifying Bob

Figure 5.25: Riot: Verification process

Other security implementations

Riot does not have that many extra security implementations, but since the application is only in beta, they may be implemented during the beta period. Figure 5.26 shows the settings page for details about a room. The administrator of the room (the one who initialized the room) is the only user who can change the settings of the room. The last setting shown in the figure is the option to enable encryption in that specific room. When encryption is first enabled, it can not be disabled throughout the conversation.


17:53

Room Details

PEOPLE FILES SETTINGS

Room History Readability
Members only (since the point in time of selecting this option)

Addresses
This room has no local addresses
New address (e.g #foo:matrix.org)

Advanced
This room's internal ID
!kAptmNyZFAvaifYwru:matrix.org

Enable encryption
(warning: cannot be disabled again!)

Figure 5.26: Riot: Other security implementations

5.2.6 Case 6: Telegram

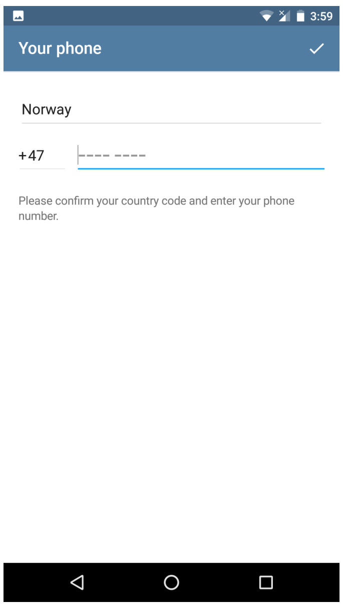
Telegram is an instant messaging platform which was developed in 2013 after the NSA scandal. It has been developed for smartphones, tablets and even computers [52]. It is possible to talk to a single person, group communication, and the possibility to send files to other people in your contacts. The difference between Telegram and the other secure instant messaging applications is that it does only offer opt-in secure messages, while normal conversations are cloud chats that are not end-to-end encrypted. It would be better if they could have secret chats as default, the same way Signal, and others do it, but they want to offer seamless cloud chat synchronization between all connected devices [17].

When it comes to the secure chats, they have implemented their own version of a cryptographic protocol that they have named the MTProto Protocol [16]. The same protocol is also used for normal cloud chats to encrypt the communication between the server and the client.

The end-to-end encrypted chats that Telegram provides does not allow users to screenshot inside the secret chat conversation. Therefore, the images within the conversation are shot with an external camera.

Initial Set Up

The initial setup of the Telegram application and user registration is the same the other applications. Figure 5.27a shows where the user inputs their phone number for the registration, and figure 5.27b shows the activation process. Telegram sends an activation code with SMS, can either input manually or give Telegram access to do it automatically, and if the message does not get received by the user in the next two minutes, it sends a new message. If the user never receives the verification code by SMS, then it can ask Telegram to call the user and activate it through phone call.



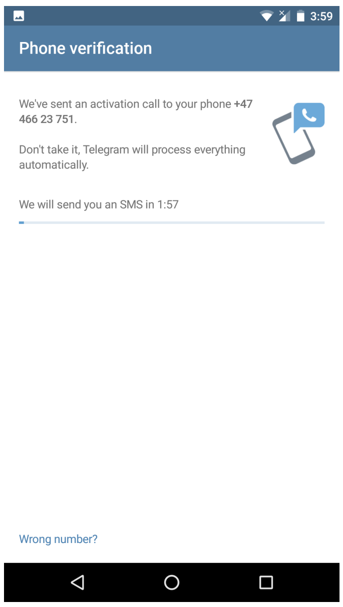
Your phone

Norway

+47 ---- ----

Please confirm your country code and enter your phone number.

(a) Phone number registration



Phone verification

We've sent an activation call to your phone +47 466 23 751.

Don't take it, Telegram will process everything automatically.

We will send you an SMS in 1:57

Wrong number?

(b) Verification of phone number

Figure 5.27: Telegram: Registration process

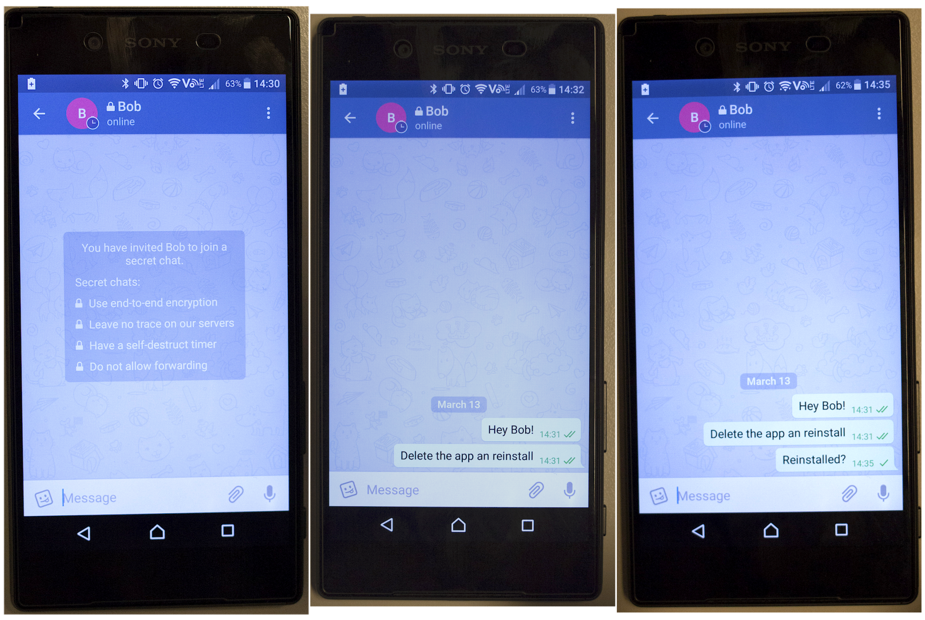
Message after key change

Telegram does not have end-to-end encryption on by default, which means that Alice needs to start an explicit secret chat with Bob. The normal messages, which are called cloud chats [52] on Telegram, do not have any encryption. Figure 5.28a shows the first view Alice gets when initiating a conversation with Bob. Telegram gives information about the secret chat that it is end-to-end encrypted and that it does not allow forwarding of messages for security reasons.

Figure 5.28b shows Alice initiating a secure chat conversation with Bob, and the double checkmarks illustrate that Bob has received and read the message, while a single checkmark illustrates the message has been sent. Alice tells Bob to reinstall the application to test if the key changes have any impact on the conversation between them.

Alice knows Bob has reinstalled his application and tries to send a new message to Bob, as shown in figure 5.28c. Bob never received the message, even after he has reinstalled the application. This shows that Telegram only uses the same keys while the application is installed, and if it is uninstalled, the device loses its keys. Alice is sending messages with the old device keys and will need to generate new ones by initiating a new secret chat with Bob for that to work, and exchanging new security keys.

Telegram does not store keys anywhere, or any other information that two users have had a secret chat, to check if one of the users have reinstalled the application or not. It is all upto the users to initiate new secret chats if the keys are lost or deleted securely by one of the participants.



You have invited Bob to join a secret chat.

Secret chats:

- Use end-to-end encryption
- Leave no trace on our servers
- Have a self-destruct timer
- Do not allow forwarding

March 13

Hey Bob! 14:31

Delete the app an reinstall 14:31

March 13

Hey Bob! 14:31

Delete the app an reinstall 14:31

Reinstalled? 14:35

(a) Initial contact (b) Alice: Initial message (c) Message after reinstall

Figure 5.28: Telegram: Message after key change

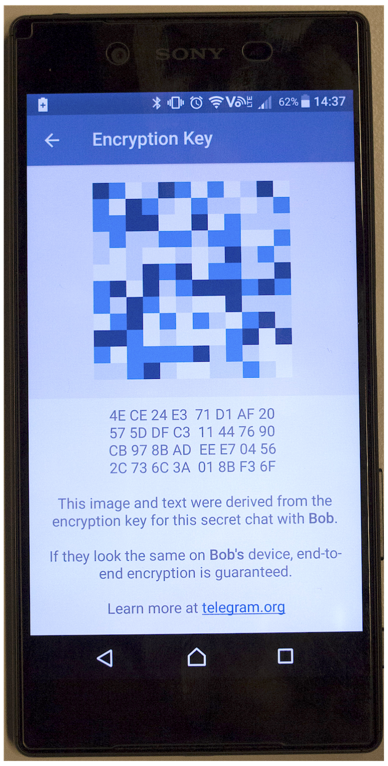
Key change while a message is in transit

This section does not have any value when it comes to the way Telegram handles messages that have been sent while Bob was reinstalling the application. As we described in the last section about key changes in Telegram, it does not store any information that Alice or Bob were having a secret chat, it is all done by the client and nothing is sent to the server saying they were having a conversation between them.

Verification Process Between Participants

The verification process between Alice and Bob is rather difficult when using secret chats in Telegram. If Alice wants to verify Bob's encryption key she needs to go the specific secure chats settings page and click on the "Encryption Key" button, then the verification page will show. Telegram does not support calling, only messaging, which makes it harder for Alice to verify Bob. Figure 5.29 shows the verification page, with an image derived from the encryption key, and the encryption key below. There is no way for Alice to verify the conversation only by looking at it and getting to the conclusion that it is the right one.

The verification page should have used a real QR-code, the same way Signal and WhatsApp do it, and implement a QR-code scanner which can scan the code and verify it is the right encryption key.



Encryption Key

```
4E CE 24 E3  71 D1 AF 20
57 5D DF C3  11 44 76 90
CB 97 8B AD  EE E7 04 56
2C 73 6C 3A  01 8B F3 6F
```

This image and text were derived from the encryption key for this secret chat with **Bob**.

If they look the same on **Bob's** device, end-to-end encryption is guaranteed.

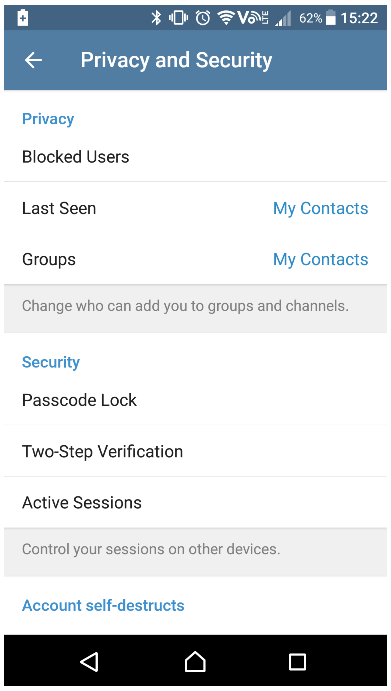
Learn more at telegram.org

Figure 5.29: Telegram: Verification process

Other security implementations

Telegram supports quite a few other security implementations. Inside the settings page, there is an option to look at the ”Privacy and Security” settings for the application. Figure 5.30a shows the settings which are possible to change within the application. Telegram supports two-step verification, which is when a user wants to log in on another device or after a reinstall, then they need to write a second, personally chosen, password after the activation code received by SMS. ”Active sessions” is a list of devices the user has logged into. The last option, ”Account self-destructs,” is a security measurement where if the user has not used their account in the last six months, the account gets deleted by Telegram. The length of the counter for self-destructing can be changed to one month, three months, six months or one year.

Figure 5.30b shows options the user gets when clicking on the ”Passcode Lock” on the ”Privacy and Security” settings list. This function locks the whole application with a passcode the user chooses. Telegram has implemented the possibility to unlock the application by fingerprint if the user has added a fingerprint in the operating system, in our case Android system. A user has the chance to change when the application should auto-lock, from one minute to five hours. The last option shown is the ”Allow screen capture” which if enabled allows users to screenshot anything inside the application, and if it is not enabled, they do not have access to do so, except secure chats that are never allowed to screenshot.



62% 15:22

Privacy and Security

Privacy

Blocked Users

Last Seen — My Contacts

Groups — My Contacts

Change who can add you to groups and channels.

Security

Passcode Lock

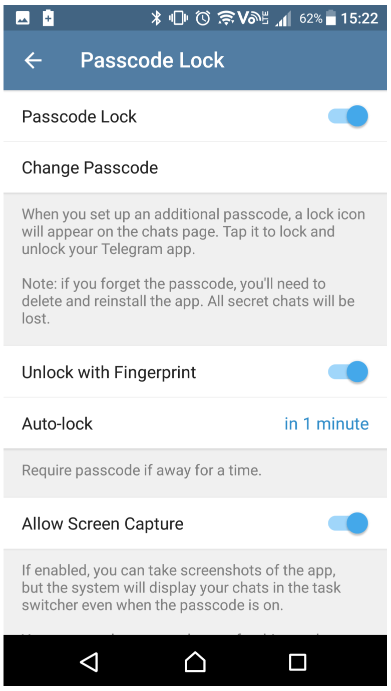
Two-Step Verification

Active Sessions

Control your sessions on other devices.

Account self-destructs

(a) Privacy and Security settings



62% 15:22

Passcode Lock

Passcode Lock ●

Change Passcode

When you set up an additional passcode, a lock icon will appear on the chats page. Tap it to lock and unlock your Telegram app.

Note: if you forget the passcode, you'll need to delete and reinstall the app. All secret chats will be lost.

Unlock with Fingerprint ●

Auto-lock — in 1 minute

Require passcode if away for a time.

Allow Screen Capture ●

If enabled, you can take screenshots of the app, but the system will display your chats in the task switcher even when the passcode is on.

(b) Passcode settings

Figure 5.30: Telegram: Other security implementations

5.3 Summary

This chapter went through the testing methodology and explained the different test scenarios and test cases. It illustrated the steps such that the reader can do the same when going through the tests.

The next chapter gathers all the information from this chapter and presents the results for each test scenario.

Chapter 6

Results

In this chapter we present the results from our testing in chapter 5 of the secure messaging applications. We show the results for each test scenario and display what types of properties each application provide and which properties are missing. We discuss the results in chapter 7 for each application as a whole.

6.1 Setup and Registration

Table 6.1: Results: Setup and Registration

Application	**Properties**					
	Phone Registration	E-mail Registration	Access SMS Inbox	Contact list Upload	Verification by SMS	Verification by Phone Call
Signal	✓	✗	✓	✓	✓	✓
WhatsApp	✓	✗	✓	✓	✓	✓
Wire	✓	✓	✗	✓	✓	✓
Viber	✓	✗	✓	✓	✓	✓
Riot	✗	✓	✗	✓	✗	✗
Telegram	✓	✗	✓	✓	✓	✓

✓= Provides property ✗= Does not provide property

The applications provided mostly the same properties when it came to the setup and registration test scenario. All the applications except Riot are created to register with the user's phone numbers while Riot needs an e-mail address. When testing the Wire application, we found out that it is the only application that supports both phone number and e-mail address to register an account and later use it to log in.

Access to the SMS inbox is not obligatory by any of the applications, but it is set up to be easier for the user because then they would not need to enter the verification code manually. While Riot does not use a phone number for verification which means it does not need access at all, Wire has decided to not ask for access because they do not see any reason that the user cannot enter the verification code themselves.

The contact list is asked to be uploaded to the server by all the applications because it is easier to find out if any of their contacts are already using a given application. If any of the users do not want to upload it to the server, they could be 100% anonymous and only give out the number to particular persons.

All the applications have the same properties when it comes to verification by SMS and phone call. They all first give the user the chance to verify by reading the SMS and if the user never receives the SMS, then they can ask the application to call them.

6.2 Initial Contact

Table 6.2: Results: Initial Contact

Application	**Properties**	
	Trust-On-First-Use	Notification About E2E Encryption
Signal	✓	✗
WhatsApp	✓	✓
Wire	✗	✗
Viber	✗	✗
Riot	✗	✓
Telegram	✗	✓

✓= Provides property ✗= Does not provide property

The initial contact is when Alice and Bob decide to start a conversation and the test checks if the applications have a Trust-On-First-Use (TOFU) and if they give any notification that the conversation is end-to-end encrypted. Signal and WhatsApp have a TOFU method, where both trust the participants in a conversation without verifying first. The other applications need to verify each other first to be assured that the conversation is secure.

A notification at the start of the conversation would be useful to a new user who does not know what end-to-end encryption is, and it only needs to be at the beginning such that users do not get bothered. Only half of the applications have this notification implemented in their applications, and the others should do an internal testing if it helps users understand end-to-end encryption.

6.3 Message After A Key Change

Table 6.3: Results: Message After a Key Change

Application	**Properties**	
	Notification about key changes	Blocking message
Signal	✓	✓
WhatsApp	✓	✗
Wire	✗	✗
Viber	✗	✗
Riot	✓	✗
Telegram	✗	✗

✓= Provides property ✗= Does not provide property

When looking at the way applications handled key changes, it became apparent that the difference in handling between the applications is quite big. Only the Signal application had both blocking messages and showed a notification that the other user in the conversation did not have the same cryptographic keys after a reinstall. A blocking message is when the sender cannot send the message before they verify the new cryptographic keys of the receiver if the receiver has generated new cryptographic keys during the conversation.

The other applications which did not give any notification or blocked a sending message could become a security issue if a man-in-the-middle attack has happened since one of the participants would never get the notification of key changes. Wire and Viber should modify the way they handle the key changes because this could become quite ugly if somehow there is an exploit in the application and man-in-the-middle attacks are possible. Telegram has the secret chats that do not work if cryptographic keys change because the application does not store any information about secret chats and then the participants would need to start a new conversation with each other to continue the conversation.

The blocking messages should be tested with users in a case study, where they go through each application and see if it is a good choice to block messages when keys change for usability. There could maybe be some problems where the users do not look at it after a while and verify new users without actually verifying the right person. This is out of our scope but could be relevant for future work.

6.4 Key Change While a Message Is In Transit

Table 6.4: Results: Key Change While a Message Is In Transit

Application	**Properties**	
	Re-encrypt and Send Message	Details About Transmission of Message
Signal	✗	✓
WhatsApp	✓	✓
Wire	✗	✓
Viber	✗	✗
Riot	✗	✓
Telegram	✗	✓

✓= Provides property ✗= Does not provide property

The only application to re-encrypt the messages and send it once again after the receiver got new cryptographic keys was WhatsApp, which is a useful usability property to have, and hopefully, the rest of the applications also implement this type of mechanic to their conversations. There is one problem with the way WhatsApp re-encrypts and sends the message again; it never asks the user if it is the correct receiver because the keys have changed. WWhen an application re-encrypts without asking and verifying that the receiver is correct, then it could become a security issue if someone has managed to get access to the receiver's account and impersonate them. If the impersonator has access, and the application does not try to verify, then the messages are sent to the wrong user.

The "details about the transmission of a message" property is if the applications show the sender that the message is either sent, delivered or seen by the receiver. If the message is never delivered because of changes to the receivers cryptographic keys, then the message only shows "sent" for the sender, but if the message is re-encrypted and sent correctly, then the message detail says "seen" by the receiver. The only application that does not show any information if past messages are sent or delivered is the Viber app. Viber does not show any details about the transmission of the message, only if the receiver reads the last message.

6.5 Verification Process

Table 6.5: Results: Verification Process

Application	**Properties**			
	QR-Code	Verify By Phone Call	Share Keys Through 3rd Party	Verified Check
Signal	✓	✓	✓	✗
WhatsApp	✓	✓	✓	✗
Wire	✗	✓	✗	✓
Viber	✗	✓	✗	✓
Riot	✗	✓	✗	✓
Telegram	✓	✗	✗	✗

✓= Provides property ✗= Does not provide property

The user experience when verifying two participants was good throughout all the applications, but at the same time, they could implement each other's way of confirming participants.

Signal and WhatsApp show the easiest verification by using a QR-code with their built-in barcode scanner, but both had shortcomings because they did not have a check which revealed that the particular user is already verified. Wire, Viber, and Riot confirmed when a user is already verified, but they did not have the useful QR-code nor any way of sharing the keys outside of the application.

Telegram was the only application which only offered a QR-code but no way of actually scanning the code. Users had to read the secret keys which are shared between them, and the QR-code picture is made to compare between participants without any technical way of doing it, only by looking at it.

6.6 Other Security Implementations

Table 6.6: Results: Other Security Implementations

Application	Properties				
	Two-Step Verification	Passphrase/Code	Screen Security	Clear Trusted Contacts	Delete Devices From Account*
Signal	✗	✓	✓	✗	✗
WhatsApp	✓	✗	✗	✗	✗
Wire	✗	✗	✗	✗	✓
Viber	✗	✗	✗	✓	✓
Riot	✗	✗	✗	✗	✓
Telegram	✓	✓	✓	✗	✓

✓= Provides property, ✗= Does not provide property, * = supports multi-device

When looking at other security implementations that the applications have implemented outside of the conversations; we found out that the apps have implemented some valuable settings for the user to strengthen the security around their account.

Signal and Telegram both had a passphrase or code when the application was not used after some specific time. The user had to enter their chosen phrase/code to gain access to the application after the timeout. Both of the applications have also implemented screen security to not give the users the ability to screenshot conversations. There is a settings to toggle the security off, but it is on by default on both Signal and Telegram.

WhatsApp and Telegram have two-step verification capabilities which means that whenever a user reinstalls the application or changes devices, they need to enter a second password after the normal verification code from the provider, to gain access to their account on the new device.

The only application which had a list of verified contacts, and the option to delete them, was Viber. Clients such as Wire and Riot, which have a verified check on each contact within a conversation, do not offer this option to have a list and delete the trusted contacts from there, which should not be that big of a problem to implement since they already know which contacts and their devices are verified.

The "delete devices from account" is only looked at if the application tested supports multiple devices. All the applications which supported multiple devices also had a list of devices such that the user could delete a device which is not in use anymore.

6.7 Summary

This chapter went through the six test that was done on six secure messaging applications and looked at what kind of properties are and are not provided by the apps. It showed that there is a big difference gap between the apps, and the apps could learn something from their competitors who have solved some of the problems a secure messaging application faces.

The next chapter is about the discussion of these results, where applications are given some general criticism and how the developers could implement features which would benefit both the users and the security of the application.

Chapter 7

Discussion

Chapter 6 presented the results from the different test scenarios done for each application. The results from each of the test scenarios vary, but there was room for improvement for each of the test cases.

This chapter discusses the results from the previous chapter. Instead of talking about each test scenario the last chapter showed results for; each application is discussed as a whole, and some recommendations for improvement is given to the developers. The limitations of the research which were found during the testing are discussed and what kind of improvement could be done in future experiments.

7.1 Signal

The Signal application did not show many weaknesses throughout the test, but we did find some potential improvements. Signal showed that they had a good grasp on the user experience with an easy verification process, where they used QR-Codes to verify and at the same time give users options to call each other to verify through end-to-end encrypted phone calls.

The notification about key changes pops up on the sender's side of the conversation, after they have sent a message, and not before or immediately when the receiver has generated new cryptographic keys. When a sender does send a message after key changes, the message gets blocked by the application until they verify each other. This is a useful property to have, but the notification that the keys have changed gets revealed after the verification instead of immediately when one of the users has new keys.

Overall the application has both good security when it comes to end-to-end encryption and useful user experience properties which would not cause problems for new users to jump right in and use.

7.1.1 Recommendations for Improvement

We came up with a few improvements which would benefit the Signal application, but it is important to note that these improvements should go through testing before deploying them to production.

- **Re-encrypt and send lost messages:** Give the user some form of option to re-encrypt a lost message to resend after the users are done with the verification. The sending user always knows when a message is sent, delivered or read because of the checkmarks on each message within a conversation. This benefits both the sender and receiver because messages would not get lost during a conversation.
- **Notification about key changes:** Move the notification message, which pops up on both users within a conversation after the verification process, to before someone sends a message, right after one of the users generates new keys. If the notification is moved before sending a message, then the sender knows before sending a message that keys are changed and would like to know if is the same receiver before sending messages that may be confidential.
- **Notification on E2E encryption:** On each new message conversation that is initiated, give a notification at the beginning of the conversation that it is now end-to-end encrypted and the possibility to read more about it if the users want to. This information could help educate the end-user about what E2E-encryption is and why they should care about it.
- **Verified check:** Add a way of knowing if a user is already verified or not because the Signal application does not have any way of knowing at this time. If the application keeps the information about which contact is verified by the user, then it can quickly show when one of the contacts changes cryptographic keys and could ask the user to verify each other again to regain the security properties.
- **Two-step verification:** Add a new security option to enable two-step verification for when a user changes devices or reinstalls the application. This could prevent users losing control of their accounts since hackers would need to know the second password after a verification to get access to the account.

7.2 WhatsApp

We did not find many weaknesses with the WhatsApp application, but as with the Signal application, we did come up with a few improvements that could benefit the user.

We found out that WhatsApp does enough to strengthen the security around the user's account and messages and there is a small chance of any kinds of man-in-the-middle attacks, but there could be some impersonation attacks. These types of attacks could happen because, by default, the WhatsApp application does not give any notification about key changes to the user. If the users suddenly got notifications about changes after end-to-end encryption was added to the application, then user confusion may occur and lead to angry users or deletion of the application. The problem of not having it on by default is that it gives the users a false sense of security because they never know if something has changed throughout the conversation.

7.2.1 Recommendations for Improvement

There are some improvements which we recommend WhatsApp implement to secure the application even more than it is today:

- **Re-encrypt and resend after verification:** The WhatsApp application immediately re-encrypts a lost message when it finds out that the cryptographic keys have changed and the receiver never received the message. The application should wait until the user has verified new keys before re-encryption because of the possibility of an impersonator. If an adversary has managed to impersonate a contact and WhatsApp re-encrypt and sends lost messages, then the possibility of sending private messages to the wrong person rises drastically because the sender cannot stop the message being sent after keys change.
- **Option for blocking messages:** Add an option for the user to enable blocking messages before the users within a conversation verify each other. If WhatsApp adds an option to block messages before two participants verify each other, then the possibility of sending private messages to an impersonator drops significantly, because they would need first to verify each other then send the message.
- **Verified check:** Add a way of knowing if a user is already verified or not because the Signal application does not have any way of knowing at this time. If the application keeps the information about which contact is verified by the user, then it can quickly show when one of the contacts changes cryptographic keys and could ask the user to verify each other again to regain the security properties.
- **Passphrase/code:** Add an option to enable passphrases or codes before opening the application to strengthen the user's account from unauthorized people. If an adversary manages to get access to the user's phone, then the passphrase/code when accessing WhatApp could defend from any adversary trying to get access to messages.

7.3 Wire

The Wire application had useful security and usability properties, but we did find properties which were not implemented and could cause serious security problems.

The users never get any information that participants change devices or add new devices to the conversation. When a user can add new devices to the account, and the conversations do not notify the other participants, could become a serious problem if someone has managed to gain access to a users account. The impersonator joins the conversation and receives every single message that is sent throughout the conversation.

7.3.1 Recommendations for Improvement

We found a handful of improvements for the Wire application which could benefit the app as a whole.

- **Tell the user about verification:** Explain to the user that the application does not automatically verify when initiating a conversation. Tell them that the users should verify each other first before sending messages, to ensure that nobody is impersonating them. If users within a conversation do not verify each other, then the possibility of having an impersonator is quite high because none of them have checked if they are talking to the right person.

- **Option for blocking messages:** Add an option for the user to enable blocking messages before the users within a conversation verify each other. If Wire adds an option to block messages before two participants verify each other, then the possibility of sending private messages to an impersonator drops significantly, because they would need first to verify each other then send the message.

- **Notification about key changes:** Add an option to get notifications when a user adds new devices because Wire allows multiple devices. When a user adds a new device to the conversation by installing the application on the second device, then the other participant should get a warning that another device has been added to the conversation they should verify that particular device before sending any more messages.

- **Re-encrypt and send lost messages:** If a message is lost, give the user an option to re-encrypt and resend the lost message to the receiver. This usability property would be useful to have for both of the users because then the conversation could continue without any hiccups because lost messages may confuse the users.

- **Verification options:** Add different ways of verifying each other within a conversation, because calling a person every time may become cumbersome for users. A QR-code or sharing keys with a 3rd party application could be some examples of improvements

- **Two-step verification:** Add a new security option to enable two-step verification for when a user changes devices or reinstalls the application. This could prevent users losing control of their accounts since hackers would need to know the second password after a verification to get access to the account.

- **Passphrase/code:** Add an option to enable passphrases or codes before opening the application to strengthen the user's account from unauthorized people. If an adversary manages to get access to the user's phone, then the passphrase/code when accessing Wire could defend from any adversary trying to get access to messages.
- **Screen Security:** Add an option in the settings page to enable screen security which does not allow screenshots within the application or any conversation. The reason to have screen security is that the conversations become more secure and private between then participants, but a screen security is as good as both the participants enabling it. If the screen security is not enabled, then it has no effect because one of the users has total access to screenshot any conversation.

7.4 Viber

The Viber application has some good choices in usability and security properties but falls short in some areas which should be a priority when adding end-to-end encryption.

There are enough bad implementations in Viber that it should be sufficient to warrant people not to use the application if they care about privacy and security. When cryptographic keys change, none of the users get any information about it, and it is not possible to get information if the users have not verified each other first, before changing keys. If messages are lost in between reinstalls, we found out that the sender cannot see if the message is sent or received if they send multiple messages before looking at the status of the message

7.4.1 Recommendations for Improvement

Viber has many flaws which we mean are trivial to fix and drastically improve the application.

- **Details about sent messages:** Add information to each message if the message has been sent, delivered or read because at the time of this test there is no way of knowing if old messages are read by or delivered to the receiver, only the last message has the information. If one user sends two messages, then the first message does not have any information if it has been read by the receiver, because Viber does only show information about the last message. This could be confusing for the sender because they never receive information about the status.
- **Tell the user about verification:** Explain to the user that the application does not automatically verify when initiating a conversation. Tell them that the users should verify each other first before sending messages, to ensure that nobody is impersonating them. If users within a

conversation do not verify each other, then the possibility of having an impersonator is quite high because none of them have checked if they are talking to the right person.

- **Option for blocking messages:** Add an option for the user to enable blocking messages before the users within a conversation verify each other. If Viber adds an option to block messages before two participants verify each other, then the possibility of sending private messages to an impersonator drops significantly, because they would need first to verify each other then send the message.

- **Notification about key changes:** Add an option to get notifications when a user gets new cryptographic keys because then they would need to re-trust each other.

- **Re-encrypt and send lost messages:** If a message is lost, give the user an option to re-encrypt and resend the lost message to the receiver. This usability property would be useful to have for both of the users because then the conversation could continue without any hiccups because lost messages may confuse the users.

- **Verification options:** Add different ways of verifying each other within a conversation, because calling a person every time may become cumbersome for users. A QR-code or sharing keys with a 3rd party application could be some examples of improvements

- **Two-step verification:** Add a new security option to enable two-step verification for when a user changes devices or reinstalls the application. This could prevent users losing control of their accounts since hackers would need to know the second password after a verification to get access to the account.

- **Passphrase/code:** Add an option to enable passphrases or codes before opening the application to strengthen the user's account from unauthorized people. If an adversary manages to get access to the user's phone, then the passphrase/code when accessing Wire could defend from any adversary trying to get access to messages.

- **Screen Security:** Add an option in the settings page to enable screen security which does not allow screenshots within the application or any conversation. The reason to have screen security is that the conversations become more secure and private between then participants, but a screen security is as good as both the participants enabling it. If the screen security is not enabled, then it has no effect because one of the users has total access to screenshot any conversation.

7.5 Riot

The Riot application is still in beta, which is the reason for some of the choices they made when developing the application. There are some good usability and security properties, but there are also a few bad properties as well.

When cryptographic keys change during a conversation, then the previous messages are locked for the user when reinstalling the application, but the sender has an easy option to resend messages if necessary. After a reinstall, the users need to verify each other again because the keys have changed and the application does notify the users about changes.

There are some bad choices here, and these are that end-to-end encrypted is not on by default but this changes when the application is out of beta. The same with the verification process, there is not an easy way of verifying users, but Riot has stated they are working on changing the method.

7.5.1 Recommendations for Improvement

We came up with a few improvements for the Riot application which could benefit the app as a whole.

- **Verification options:** Add different ways of verifying each other within a conversation, because calling a person every time may become cumbersome for users. A QR-code or sharing keys with a 3rd party application could be some examples of improvements

- **Two-step verification:** Add a new security option to enable two-step verification for when a user changes devices or reinstalls the application. This could prevent users losing control of their accounts since hackers would need to know the second password after a verification to get access to the account.

- **Passphrase/code:** Add an option to enable passphrases or codes before opening the application to strengthen the user's account from unauthorized people. If an adversary manages to get access to the user's phone, then the passphrase/code when accessing Wire could defend from any adversary trying to get access to messages.

- **Screen Security:** Add an option in the settings page to enable screen security which does not allow screenshots within the application or any conversation. The reason to have screen security is that the conversations become more secure and private between then participants, but a screen security is as good as both the participants enabling it. If the screen security is not enabled, then it has no effect because one of the users has total access to screenshot any conversation.

7.6 Telegram

Telegram has some useful security properties, but the usability features are a bit lacking and confusing for people who are not tech savvy.

The biggest flaw for a secure messaging application is that the end-to-end encryption is not on by default. We think that this should become a norm these days for an application that advertises encrypted conversations.

If a message is sent to the receiver after cryptographic keys change, then no message will arrive because secret chats are locked to one set of keys, and when a user generates new keys, they would need to start an entirely new secret chat. This is good for security, but the problem arises when a user does not get the information about key changes because the application never tells them.

7.6.1 Recommendations for Improvement

We do not have that many recommendations for improvement because Telegram does not have end-to-end encryption on by default which means an explicit secret chat needs to be initiated every time users want to communicate.

- **Verification options:** Add different ways of verifying each other within a conversation because only showing the QR-code in person is not a good way of doing it. There should be more options such as calling and sharing keys through 3rd party applications.

- **Verified check:** Add a way of knowing if a user is already verified or not because Telegram does not have any way of knowing at this time. If the application keeps the information about which contact is verified by the user, then it can quickly show when one of the contacts changes cryptographic keys and could ask the user to verify each other again to regain the security properties.

- **Notify about key changes:** Telegram should notify the user when the other participant has deleted the application, and they would need to initiate a new conversation. Telegram does not give any information as it is implemented today, which would lead to lost messages because the sender never gets the information that the receiver has deleted or changed devices.

7.7 Research Improvements

We found throughout the testing of the applications that this does not give us real life results because they are tested by only one person instead of in a group by multiple participants. The best way of going through this test case would be with a group of participants where the knowledge about secure messaging applications ranges from none at all to participants with knowledge about these kinds of applications. We only had one person testing the applications which make the generalization of the problems difficult, because they become biased instead of coming to a conclusion from many different perspectives of other users.

To receive better results from this type of research, the use of participants can gather different results from each person, which means that the results could be different than what we found during our single research.

The applications were chosen because we had used them before personally. What should have been done to get a better grasp of what types of applications people use, was to set up a questionnaire for different applications to choose from and then we could find out which app to research.

7.8 Summary

This chapter discussed the results of each application as a whole and found that there are multiple improvements for each application. The applications were not perfect, but with the recommended improvements, the applications could ensure a more secure application and at the same time make a good user experience.

The research was also discussed in how it could be improved by using multiple participants instead of doing it at a test lab because there is a risk of biased studies in the results.

Chapter 8

Conclusion

This thesis has shown how secure messaging applications have an impact on us when usability overshadow the security properties. The prioritization of usability could have a negative effect on the security of the applications instead of prioritizing them equally.

The results show that the main caveat of the secure messaging applications is the way changes in cryptographic keys are handled between parties within a conversation. In multiple applications, the participants are never given any information about changes, which weakens the security of the conversation.

This thesis conducted two analyses about secure messaging protocols and applications which implement these types of protocols. The first analysis (following the recent article by Unger et al. [53]) described old and new secure messaging protocols that offer end-to-end encryption and identified types of security and privacy properties they provide. The Signal and Matrix protocols are both secure messaging protocols that manage end-to-end encryption well, but none of them could offer every security property. The analysis concluded that none of the secure messaging protocols could provide every security property and that it was room for improvement if developers work together and discuss how they can help each other in implementing the rest of the properties that are missing. The Signal protocol does not fully support multiple devices, while the Matrix protocol does provide it. On the other hand, the Matrix protocol does not fully provide forward and future secrecy in the protocol, because it is up to the implementation to support it.

The second analysis conducted in this thesis was the research experiment of applications which support these secure messaging protocols within their conversations. The applications offer useful usability together with security, but there are multiple applications which still benefit from improvements. The types of improvements recommended could harden the security around the user's account, and at the same time keep the useful usability properties. Chapter 5 went through what types of usability properties we want the applications to offer on the test scenarios, and then showed steps to reproduce the scenarios on each application. Chapter 6 concluded that the applications offered useful usability and security properties. However, multiple applications were missing important usability properties which could jeopardize the security of the users because they would never know if there was an attack on the conversation. Recommendations have been given in Chapter 7 for each application to harden their security around the user's account, but at the same time keep the useful usability properties. The given recommendations are not tested to confirm that it is the best choice for the applications, and it is up to the developers to prioritize if they benefit their applications and users.

When discussing the shortcomings of our research in section 7.7, we conclude that the research could be redone with multiple participations to ensure a more generalized result of the test cases. Usability and security of secure messaging applications should be done in a case study where the range of secure messaging applications between multiple participants ranges from no knowledge at all to participants with security knowledge. The results that are given in this thesis may be seen as biased because of one opinion on secure messaging applications, instead of multiple opinions.

On April 20th, a related blog article about usability and security of applications came to our attention [40]. The article examined the same types of scenarios of apps as this thesis did, but not with the same depth and research behind the protocols and applications, and no recommendations were given to the developers to improve their applications.

It is solely up to the user which secure messaging application they want to use, but hopefully, by reading this master thesis, they can come to a conclusion they find appealing for them and start using a messenger which gives the user enough security and privacy to stand against the government surveillance.

8.1 Future Work

The evaluation tests that we did could be redone with multiple participants. By using multiple participants, it could be easier to identify how a particular device/app is used and if it is easier for impersonators and attackers to lure users into giving private information. As we mentioned in section 1.5 about related work, Schroder et al. [51] did a user study with participants, but only on the Signal application, while we could do it with each application that we described in this thesis.

Another interesting research avenue is the automated formal verification that Blanchet et al. [31] has done for Signal protocol, and Cohn-Gordon et al. [10] with their formal manual verification of the same protocol. One could look at a similar verification with ProVerif[1] and the Matrix protocol to verify that Matrix respects authenticity, confidentiality, forward secrecy and other important security properties.

Formal verification of the code of the different mobile applications studied in this thesis is also useful. Baier et al. [3] presents principles of model checking, which is a prominent formal verification technique

[1]`http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`

for assessing functional properties of information and communication systems. The book also talks about a model checker called SPIN which is made by Gerard Holzmann [28] and is a tool which is used for formal verification of systems and applications. By combining the different techniques and principles from the *Principles of Model Checking* and the model checker SPIN, we could do a verification of the different applications that we went through in this thesis.

Bibliography

[1] Chris Alexander and Ian Goldberg. Improved user authentication in off-the-record messaging. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, WPES '07, pages 41–47, New York, NY, USA, 2007. ACM.

[2] Eric Auchard. Go ahead, make some free, end-to-end encrypted video calls on wire. `http://www.reuters.com/article/us-dataprotection-messaging-wire-idUSKCN0WC2GM`, 2016. Last accessed 2017-04-23.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[4] Alex Balducci and Jake Meredith. Olm cryptogrpahic review. Technical report, NCC Group PLC, 2016. Last accessed 2017-03-15.

[5] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. *Keying Hash Functions for Message Authentication*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[6] Mihir Bellare and Phillip Rogaway. Message authentication. `http://cseweb.ucsd.edu/~mihir/cse207/w-mac.pdf`, 2005. Last Accessed 2017-04-21.

[7] J. Bian, R. Seker, and U. Topaloglu. Off-the-record instant messaging for group conversation. In *2007 IEEE International Conference on Information Reuse and Integration*, pages 79–84, Aug 2007.

[8] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, pages 77–84, New York, NY, USA, 2004. ACM.

[9] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer, 2003.

[10] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the Signal messaging protocol. In *Proc. IEEE European Symposium on Security and Privacy (EuroS&P) 2017*. IEEE, April 2017.

[11] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Secure off-the-record messaging. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, pages 81–89, New York, NY, USA, 2005. ACM.

[12] Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008. `https://rfc-editor.org/rfc/rfc5246.txt`.

[13] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, sep 2006.

[14] Gordana Dodig-Crnkovic. Scientific methods in computer science. `https://users.dcc.uchile.cl/~cgutierr/cursos/INV/crnkovic.pdf`, 2002.

[15] Tyler Durden. Wikileaks unveils 'vault 7': "the largest ever publication of confidential cia documents"; another snowden emerges. `http://www.zerohedge.com/news/2017-03-07/wikileaks-hold-press-conference-vault-7-release-8am-eastern`, march 2017. Last accessed 2017-04-12.

[16] Nikolai Durov. MTProto Protocol. `https://core.telegram.org/mtproto`. Last accessed 2017-02-03.

[17] Pavel Durov. Seamless chat cloud synd. `https://twitter.com/durov/status/678305311921410048?ref_src=twsrc%5Etfw`. Last accessed 2017-02-03.

[18] Morris J. Dworkin. Sp 800-38a 2001 edition. recommendation for block cipher modes of operation: Methods and techniques. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2001.

[19] Justin Engler and Cara Marie. Secure messaging for normal people. Technical report, NCC Group, 2015.

[20] Ricardo Freitas. Scientific research methods and computerscience. `http://www.map.edu.pt/mapi/2008/map-i-research-methods-workshop-2009/RicardoFreitasFinal.pdf`, 2009.

[21] Wire Swiss GmbH. Proteus protocol. `https://github.com/wireapp/proteus`. Last accessed 2017-02-04.

[22] Ian Goldberg. Off-the-record messaging cypherpunks. `https://otr.cypherpunks.ca/`.

[23] Ian Goldberg, David Goulet, Jacob Appelbaum, and Jurre van Bergen. Off-the-record messaging protocol version 3. Technical report, University of Waterloo, 2012.

[24] Ian Goldberg, Berkant Ustaoğlu, Matthew D. Van Gundy, and Hao Chen. Multi-party off-the-record messaging. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 358–368, New York, NY, USA, 2009. ACM.

[25] D. Gollmann. *Computer Security*. Wiley, 2011.

[26] S. Harris. *CISSP All-in-One Exam Guide, 6th Edition*. All-in-One. McGraw-Hill Education, 2012.

[27] Matthew Hodgson. Encrypting matrix: Building a universal end-to-end encrypted communication ecosystem with matrix and olm. `https://fosdem.org/2017/schedule/event/encrypting_matrix/`, Last accessed 2017. 2017-03-29.

[28] Gerard J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, May 1997.

[29] Rakuten Inc. Viber. `https://www.viber.com/en/about`. Last accessed 2017-02-02.

[30] TechCrunch Ingrid Lunden. Viber follows messenger, launches public accounts for businesses and brands. `https://techcrunch.com/2016/11/09/viber-follows-messenger-with-the-launch-of-public-accounts-for-businesses-and-br` 2016. Last accessed 2017-02-04.

[31] N. Kobeissi, K. Bhargavan, and B. Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017. to appear.

[32] Hugo Krawczyk. *SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols*, pages 400–425. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[33] Hong Liu, Eugene Y. Vasserman, and Nicholas Hopper. Improved group off-the-record messaging. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES '13, pages 249–254, New York, NY, USA, 2013. ACM.

[34] Moxie Marlinspike. Advanced ratcheting. `https://whispersystems.org/blog/advanced-ratcheting/`, 2013.

[35] Moxie Marlinspike. Simplifying otr deniability. `https://whispersystems.org/blog/simplifying-otr-deniability/`, 2013. Last accessed 2017-03-25.

[36] Moxie Marlinspike. Open whisper systems partners with whatsapp to provide end-to-end encryption. `https://whispersystems.org/blog/whatsapp/`, 2014. Last accessed 2017-02-04.

[37] Moxie Marlinspike. Whatsapps's signal protocol integration is now complete. `https://whispersystems.org/blog/whatsapp-complete/`, 2016. Last accessed 2017-02-04.

[38] Moxie Marlinspike and Trevor Perrin. The double ratchet algorithm. `https://whispersystems.org/docs/specifications/doubleratchet/`, 2016.

[39] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol. `https://whispersystems.org/docs/specifications/x3dh/`, 2016.

[40] Tina Membe. A look at how private messengers handle key changes. `https://medium.com/@pepelephew/a-look-at-how-private-messengers-handle-key-changes-5fd4334b809a`, 2017. Last Accessed 2017-04-20.

[41] Viber Michael Schmilov. Giving our users control over their private conversations. `https://www.viber.com/en/blog/2016-04-19/giving-our-users-control-over-their-private-conversations`, 2016. Last accessed 2017-02-04.

[42] Information Technology Laboratory (National Institute of Standards and Technology). *Announcing the Advanced Encryption Standard (AES).* Federal information processing standards publication, 2001.

[43] Matrix org. Matrix.org launches cross-platform beta of end-to-end encryption following security assessment by ncc group. `https://pr.blonde20.com/matrix-e2e/`. Last accessed 2017-03-22.

[44] Matrix org. Megolm group ratchet. Technical report, Matrix, 2016. Last accessed 2017-03-22.

[45] C. Paar and J. Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners.* Springer Berlin Heidelberg, 2009.

[46] Christof Paar and Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners.* Springer Publishing Company, Incorporated, 1st edition, 2009.

[47] Sarah Perez. Skype co-founder backs wire, a new communications app launching today on ios, android and mac. `https://techcrunch.com/2014/12/02/skype-co-founder-backs-wire-a-new-communications-app-launching-today-on-ios-a`
2014. Last accessed 2017-03-14.

[48] Trevor Perrin. The xeddsa and vxeddsa signature schemes. `https://whispersystems.org/docs/specifications/xeddsa/`, 2016.

[49] The Statistics Portal. Number of monthly active whatsapp users worldwide from april 2013 to january 2017 (in millions). `https://www.statista.com/statistics/260819/`

number-of-monthly-active-whatsapp-users/. Last accessed 2017-02-04.

[50] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 98–107, New York, NY, USA, 2002. ACM.

[51] Svenja Schröder, Markus Huber, David Wind, and Christoph Rottermanner. When signal hits the fan: On the usability and security of state-of-the-art secure mobile messaging. In *1st European Workshop on Usable Security*, Proceedings of 1st European Workshop on Usable Security, July 2016.

[52] Telegram. Telegram faq. https://telegram.org/faq. Last accessed 2017-02-03.

[53] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. Sok: Secure messaging. In *2015 IEEE Symposium on Security and Privacy*, pages 232–249, May 2015.