UiO : Department of Informatics
University of Oslo

Comparing Implementations of Secure Messaging Protocols (long version)

Christian Johansen , Aulon Mujaj , Hamed Arshad , Josef Noll

Research report 475, November 2017

ISBN 978-82-7368-440-0
ISSN 0806-3036

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, extending and making popular the old off-the-record protocol. Several new implementations of such end-to-end encrypted messaging have appeared, as well as commonly used chat applications have been updated with implementations of such protocols. In this survey we compare existing implementations, where most of them implement one of the recent and popular protocols called Signal. We conduct a series of experiments on these implementations to find out which types of security and usability properties each application provides. The results of the experiments demonstrate that the applications have variations of usability and security properties, and none of them are infallible. Finally, the paper gives proposals for improving each application wrt. security, privacy, and usability. This technical report is based on the work done in [40].

Keywords: Secure messaging; End-to-End encryption; Signal; Security; Privacy; Usability.

Disclaimer: All the tests in the work reported here were done in the summer of 2017, and since applications in this area are very dynamic, some of the specific implementation recommendations and observations that we make may have already been treated by the developers. However, this work still should provide guidance to a user on how to check whether such desired features have been implemented in some specific application that we survey here.

[0] *Address for correspondence:*
Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway.
E-mail: `cristi@ifi.uio.no`

[0] A more succinct version of this work is **under review** for a journal.

Contents

1 Introduction

In recent years, the trend to use mobile applications for communication has grown and become a standard method of communication between people. 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 in the beginning. 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.

Over the last few years, the need for secure messaging protocols has become apparent. People are more prone to understand the privacy implications of mass surveillance. Edward Snowden has sparked a heated debate throughout the world about individual privacy which is undermined by the mass surveillance that 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.[3] The documents that leaked have information on how to get access to mobile phones or personal computers without the user's knowledge, and how the CIA did their fair share of mass surveillance.

Several 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 their adoption. After a while, companies such as Google, Facebook, and Open Whisper Systems joined forces to implement protocols into already wide adopted applications such as WhatsApp, which has over one billion monthly active users.[4]

Instant Messaging clients that did not provide asynchronous communication became uninteresting because of the rise of smart-phones 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 with asynchronous communication in mind. After the Snowden revelations, new secure messaging applications started to emerge, most notably the Signal application with their protocol also called Signal. After a while, the new protocol became quite popular among developers and researchers because of their secure messaging capabilities. Subsequently, the Signal protocol started to be implemented in other applications, which were only supporting client-to-server encryption until then.

Quite a number of new secure messaging applications exist (in 2017 we counted six, which we survey in this work) that offer end-to-end encrypted message conversations over mobile phones and computers, but these often sacrifice usability aspects for security. I the light of the above motivations one would probably prioritise privacy and security, but in order to attract most normal users 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. Moreover, the security controls should be

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

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

[3] "Wikileaks Unveils 'Vault 7': The Largest Ever Publication Of Confidential CIA Documents; Another Snowden Emerges", authored by Tyler Durden in the Zerohedge, March 2017. Available at `http://www.zerohedge.com/news/2017-03-07/wikileaks-hold-press-conference-vault-7-release-8am-eastern`

[4] The Statistics Portal: *"Number of monthly active Whatsapp users worldwide from April 2013 to December 2017 (in millions)"* `https://www.statista.com/statistics/260819/number-of-monthly-active-whatsapp-users/`

intuitive and usable enough to be handled by a majority of people, not only for the technology inclined ones.

The goals of this study

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, our first goal is:

G1: Provide a detailed, yet comprehensible and comparative study of relevant approaches to end-to-end encrypted messaging technologies?

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, i.e., usability and security, are usually conflicting, and a good balance is difficult to find. This leads us to our second goal:

G2: 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. Moreover, the Signal protocol is employing state of the art technology in encryption and key establishment. Hence, our additional goal is:

G3: What are the security mechanisms behind the Signal protocol; explained for non-experts?

There has not been much research done in the area of usability vs. security in 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. [46] 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 they found with users failing to detect and deter man-in-the-middle attacks. This paper does not use participants, but it does look at the same types of potential attack spots as [46] extending to five more applications than Signal.

Unger et al. [49] did a comprehensive study of secure messaging protocols by looking at the security properties around trust establishment, conversation security, and transport privacy. Their survey shows that protocols that specialise in encryption do not manage to provide every important security and privacy property. In this paper, our properties of interest are based on those that [49] goes through, focusing on the conversation security for the three major end-to-end encrypted messaging protocols that we found.

Main Contributions

- We make a comprehensive analysis of application that implement secure messaging, studying with six testing scenarios their essential security and usability properties.

- We provide an (updated) overview of conversation security in secure messaging protocols, following [49]. Subsequently, we describe the inner security workings of the latest versions of three major protocols (OTR, Signal, Marix), striving to make these understandable for a general audience.

The rest of the paper is organised as follows. Section 2 presents a systematisation of knowledge about three secure end-to-end encrypted messaging protocols, with a discussion of their security properties. Section 3 presents the study testing six mobile phone applications that support either the secure messaging protocols presented before or their own variants which are not open source applications. Section 4 summarises the results from the test scenarios in a unified and comparative manner. Section 5 discusses the applications as a whole also providing recommendations for improvements. Finally, Section 6 concludes the paper.

2 Background on Secure Messaging Protocols

This section provides background on the secure messaging protocols that are implemented by the applications analyzed in this paper. After presenting basic properties relevant for end-to-end encrypted messaging, and the attackers that we consider and assumptions that we make about the user applications, we look in detail at three protocols: the first is Off-the-Record (OTR), which is the baseline for the two other protocols, Signal and Matrix, which are the new protocols that are actually implemented by the current popular secure messaging applications that we consider in our study.

This section builds on the comprehensive survey [49], as well as on various other resources regarding the three protocols. Most of the resources for the two new protocols Signal and Matrix are online, since these protocols have not come out of academia. The both have built on the good foundation laid by the OTR protocol that has been well studied in academia [8, 48, 13, 1, 5, 31, 22], but who also has quite significant online resources that complement well the academic ones.

2.1 Security principles relevant for end-to-end encrypted messaging

Different secure messaging systems implement different security principles in various degrees. These are important when comparing the specifications of different secure messaging protocols or applications implementing them. Most security and privacy features that we review here can also be found in the good survey [49].

End-to-End Encryption

Communication encryption like Transport Layer Security (TLS) [14] is designed to secure communications between a client and server. Messaging applications that allow two parties, Alice and Bob, to communicate to each other through a server, can use TLS to secure their communication against network attackers. 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. Often servers cannot be trusted, as they can be hacked by an adversary, or may be contacted by law enforcement to give them the information sent by clients to the server [18].

End-to-end encryption ensures that the endpoints do the encryption while the servers only transmit the messages without being able to see the content. Neither network attackers nor a corrupted server can get access to the messages.

Confidentiality

Confidentiality ensures that the necessary level of secrecy is enforced at each junction of data processing, preventing unauthorized disclosure [24]. 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 [24]. In cryptographic protocols confidentiality is essential to ensure that keys and other data are available only as intended [9].

Attackers try to ruin confidentiality 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 [24].

Integrity

Integrity ensures that no one throughout the transmission modifies the messages. 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. 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 [24]. This can be achieved through the use of hash functions 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].

Authentication

Authentication is meant to identify the parties in a conversation. Message authentication is also called *data-origin authentication*, and protects the integrity of the sender of the message [6, 7]. Message authentication codes can provide assurance about the source and integrity of a message. A message authentication code is computed by using the message and a shared secret between the two parties [23]. If an adversary changes the message, then the computed MAC would be different as well, and moreover, an adversary cannot produce a valid MAC because only the sender and receiver have the shared secret.

Perfect Forward Secrecy

A key establishment protocol provides forward secrecy if a compromise of long-term keys of a set of principals does not compromise the session keys established in previous protocol runs involving those principals. Typical examples of protocols which provide forward secrecy are 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].

Future Secrecy

Future secrecy, as it is called by Open Whisper Systems [33], (sometimes also called backward secrecy) is the guarantee that the compromise of long-term keys does not allow subsequent ciphertexts to be decrypted by passive adversaries [49]. A protocol supports 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 future messages sent during the conversation [33].

Deniability

Deniability is a property common to new secure messaging protocols, where it is not possible for others to prove that the data was sent by some particular conversation party. 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. To provide deniability, usually secure messaging protocols have a mechanism to allow anyone to forge messages, after a conversation, to make them look like coming from someone in the conversation. Deniability also includes authenticity during the conversation so that the participants are assured that the messages they see are authentic and are not modified by anyone [20]. Deniability can be divided into three different parts:

1. **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.

2. **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.

3. **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.

Synchronicity

There are two types of communication, synchronous and asynchronous. Synchronous protocols have a requirement that all participants must be online for them to receive or send messages. Chat applications are traditionally synchronous communications. Alternatively, asynchronous messaging 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.

Modern chat protocols do not use synchronous protocols, usually because of social or technical constraints, such as device battery, limited reception or other social happenings which do not allow people to be constantly online to receive messages. That is why the majority of Instant Messaging (IM) solutions provide an asynchronous environment by having a third party server to store the messages until the other participant gets online to receive it.

Other Security Properties

Other important properties for end-to-end secure messaging protocols and applications are easily defined as following. A protocol or application may implement some but not all of these.

1. **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.

2. **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.

3. **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).

4. **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.

5. **Causality Preserving:** Implementations can avoid displaying a message before messages that causally precede it.

6. **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.

Group Chat

Group conversation are popular nowadays, e.g., using Facebook Messenger[5], Slack[6] or other popular messaging applications[7]. However, none of these messaging applications are end-to-end encrypted; this was one main goal of Open Whisper Systems with their Signal protocol for fast and reliable end-to-end encrypted group chats. Security properties in the context of group chats include.

1. **Computational Equality:** Do the participants share an equal computational load when talking to each other.

2. **Trust Equality:** There is not a single participant who has more trust or responsibility, within the group, than any other.

3. **Subgroup messaging:** Participants can send messages to only a subgroup of others without generating a new conversation.

4. **Contractible membership:** The group members do not need to restart the security protocol when a member leaves the conversation.

5. **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 to be able to change the cryptographic keys when a new user joins the secure group conversation, since then the new users will not have the ability to decrypt previously sent messages. New cryptographic keys should also be exchanged when a user leaves the conversation. Changing the keys can easily be done by restarting the protocol, but this is often computationally expensive. Protocols which offer contractible and expandable membership achieve these features without restarting the protocol.

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

2.2 Threat models considered

When evaluating the security and privacy properties of secure messaging protocols, we assume 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. These are under the usual assumptions of a Dolev-Yao model [15].
- **Passive adversaries:** These adversaries log everything that is sent to and from a user and could potentially use that information to keep track of who users 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 of the applications that the protocols are implemented into are secure and that the devices do not have malware that could exploit the messaging application.

2.3 Properties for Usability and Adoption

Various aspects need to be taken into account when looking at usability and adoption of a secure messaging application.

1. **Out-of-order resilience:** If a message is delayed in transit, but eventually arrives, its contents are accessible upon arrival.
2. **Dropped Message Resilient:** Messages can be decrypted without receipt of all previous messages. This is desirable for asynchronous and unreliable network services.
3. **Asynchronous:** Messages can be sent securely to devices which are not connected to the Internet at the time of sending.
4. **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.
5. **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.

2.4 Off-the-Record in a nut shell

Intuitively, the Off-The-Record (OTR) protocol [8, 48, 13, 1, 5, 31, 22] wants to provide for online conversations the same features that reporters want when talking with a news source. 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. A 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 is however nothing for communications over the web. OTR-like protocols aim to provide this using cryptography

techniques, since the law does not yet reach the Internet. OTR-like protocols thus need to provide perfect forward secrecy and deniability/repudiation.

Full details can be found in [40, Sec.2.4].

2.4.1 Step 1: Authenticated Key Exchange

The latest version of OTR [21] use a variation of Diffie-Hellman Key Exchanged called SIGMA. All exponentiations are done modulo a particular 1536-bit prime p, and g is a generator of the group. Alice and Bob also have long-term authentication public keys pub_A and pub_B, respectively. The point is to do an unauthenticated Diffie-Hellman key exchange to set up an encrypted channel, and inside that channel do mutual authentication.

For the Diffie-Hellman, Alice picks a random number a known only to her, computes $g^a \, mod \, p$ and sends it to Bob. Bob on the other side does the same, picks a random number, computes $g^b \, mod \, p$ and sends it to Alice. Alice calculates $ss = g^{ab} \, mod \, p$ and Bob calculates $ss = g^{ba} \, mod \, p$ which thus forms their common secret.

The plain Diffie-Hellman key exchange is vulnerable to man-in-the-middle attacks which would break the authentication that OTR needs [13]. Therefore, OTR implements a signature-based authenticated DH exchange, named SIGMA, which solves this weakness [30].

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 [30].

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.

2.4.2 Step 2: Message Transmission

The message transmission step performs encryption and authentication of messages, before sending, using AES [12] in counter mode [16] using message authentication codes (HMAC) [4] for authentication. Using AES in counter mode provides a malleable encryption scheme which allows deniability. Malleability allows to transform a ciphertext into another ciphertext which then decrypts to a related plaintext [13]. This means that a valid ciphertext cannot be connected with neither Alice nor 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 (more about this in step 4). Entire new messages, or full transcripts, can thus be forged.

For sending messages Alice 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, where usually the encryption key is a hash of the shared secret, $EK = Hash(SS)$, and then the encryption key is hashed a second time to compute the 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, $MAC(Enc_{EK}(M), MK)$. Bob computes the same EK and MK from the common secret, used to verify the MAC and decrypt the message.

2.4.3 Step 3: Re-key

Off-the-Record changes the keys every time the conversation changes directions, to make the duration of vulnerability to attacks 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 [13]. After establishing new secrets and keys, the partners erase the old secret ss and encryption key EK, so 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. The MK key is not 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 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. Publishing MK allows others to forge transcripts of conversations between Alice and Bob. This is useful since it provides extra deniability to both parties [34]. Thus, if Bob comes with a transcript of a conversation between him and Alice saying something incriminating about Alice, then Alice can in turn produce a conversation transcript incriminating the other person. 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 deleting the MK, but this is not the case since Alice's secrecy relies on Bob deleting the key information, but Alice's deniability relies only on Alice publishing her MK. 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. Therefore, the parties need to make sure they have the *same secret* which is done using the Socialist Millionaire Protocol (SMP) [26, 50]. Intuitively, SMP allows two millionaires who want to exchange information to see whether they are equally rich, without revealing anything about the fortunes themselves. Between Alice and Bob the SMP allows to know whether $ss^A = ss^B$, i.e., the respective computed secrets, without revealing these secrets to anyone [1].

2.5 Signal in a nut shell

Signal is a new end-to-end encryption protocol which has recently seen larger adoption than the Off-the-Record protocol. OTR had an original feature, i.e., refresh the message encryption keys often, which has became known as ratcheting and adopted in Signal as well [11]. The Signal Protocol is designed by Moxie Marlinspike and Trevor Perrin from Open Whisper Systems[8] to work in both synchronous and asynchronous messaging environments.[9] The goals of Signal include end-to-end encryption and advanced security properties such as forward secrecy and future secrecy [11]. Initially, Signal was divided into two different application, TextSecure[10] and RedPhone[11]. The former was for SMS and instant messaging, while the latter was an encrypted VoIP[12] application. TextSecure was based on the OTR protocol, extending the ratcheting into a Double Ratchet, combining OTR's asymmetric ratchet with a symmetric ratchet [11], and

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

[9] Advanced Ratcheting, by Moxie Marlinspike at Open Whisper Systems, November 26 2013, `https://whispersystems.org/blog/advanced-ratcheting/`

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

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

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

naming it *Axolotl Ratchet*. Open Whisper Systems later combined TextSecure and RedPhone to form the new Signal application that implements the protocol with the same name.

In recent years, the Signal Protocol has been adopted by numerous companies, such as WhatsApp[13] by Facebook, the Messenger[14] also by Facebook, and Google's new messaging app, Allo[15].

Signal uses the Double Ratchet algorithm [35] to exchange encrypted messages based on a shared secret key that the two parties have. To agree on the shared secret key Signal uses the X3DH Key Agreement [36] Protocol (standing for *Extended Triple Diffie-Hellman*[16]) which we describe later in section 2.5.3.

Full details can be found in [40, Chap.3].

2.5.1 The Double Ratchet Algorithm

The Double Ratchet Algorithm uses Key Derivation Function Chains (KDF Chains) [35] to constantly derive keys for encrypting each message, and it combines two different ratchet algorithms, a symmetric-key ratchet for deriving keys for sending and receiving messages, and a Diffie-Hellman ratchet used to provide secure inputs to the symmetric-key ratchet.

A KDF Chain is a sequencing of applications of a Key Derivation Function which returns one key used as a new KDF key for the next chain cycle as well as an output key for messages. The KDF Chain has the following important properties [35]:

- **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 role of the KDF chain throughout the Double Ratchet session is to generate the KDF keys of each party 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). While the parties exchange messages, they also exchange new Diffie-Hellman public keys. The secrets from the *Diffie-Hellman ratchet* output become the inputs to the root chain of the KDF chain, and then the output keys from the root chain become new KDF keys for the sending and receiving chains, which need to advance for each sending and receiving message. This is called the *symmetric-key ratchet*.

The output keys from the symmetric-key ratchet are unique message keys which are used to either encrypt or decrypt messages. The sending and receiving chains ensure that each message is encrypted or decrypted with a unique key which can be deleted after use. The message keys are not used to derive new message keys or chaining keys. Because of this, it is possible to store the message key without affecting the security of other keys, only the message that belongs to the particular message key may be read if this key is compromised. This is *useful for handling 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 2.5.2.

[13] `https://whatsapp.com`
[14] `https://messenger.com`
[15] `https://allo.google.com`
[16] `https://whispersystems.org/docs/specifications/x3dh/`

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 that can steal one of the chain keys can then compute all future message keys and decrypt all future messages [35].

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 existing ratchet key pair with a new one [35].

The result is a kind of "ping-pong" behaviour as the two parties take turns replacing their key pairs. An attacker that compromises one message private key has little use of it since this will soon be replaced with a new, unrelated key [35]. The Diffie-Hellman ratchet produces as output the sending chain keys and the receiving chain keys, which have to correspond, i.e., the sending chain of one party is the same as the receiving chain of the other party. Using a KDF chain here improves the resilience and break-in recovery.

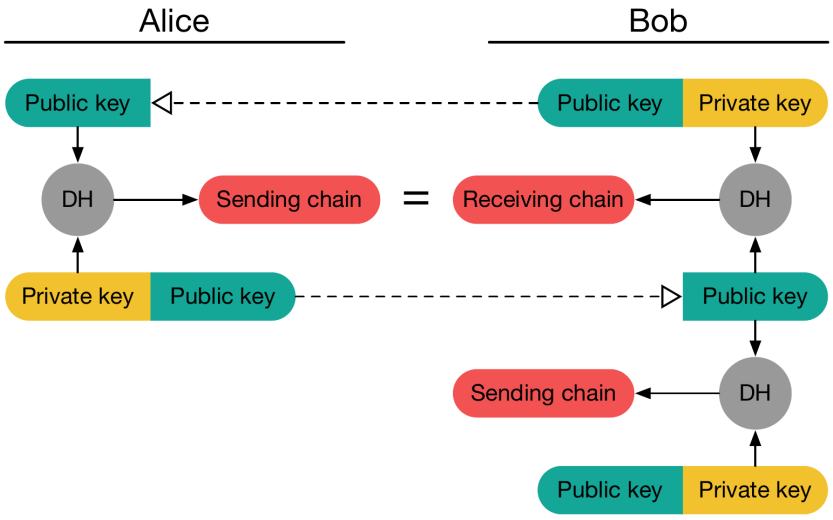


Figure 1: Sending and receiving chains (reproduced from [35]).

Combining the symmetric-key ratchet and Diffie-Hellman ratchet is as follows.

- When a message is sent or received, a symmetric-key ratchet step is applied to the sending or receiving chain to derive the message key.
- When a new ratchet public key is received, a DH ratchet step is performed prior to the symmetric-key ratchet to replace the chain keys.

Figure 2 shows the information used by Alice to send her first message to Bob. The sending chain key (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. To ensure that the secrecy throughout the Double Ratchet is upheld, the old root key (RK) is deleted after it has been used to derive a new RK.

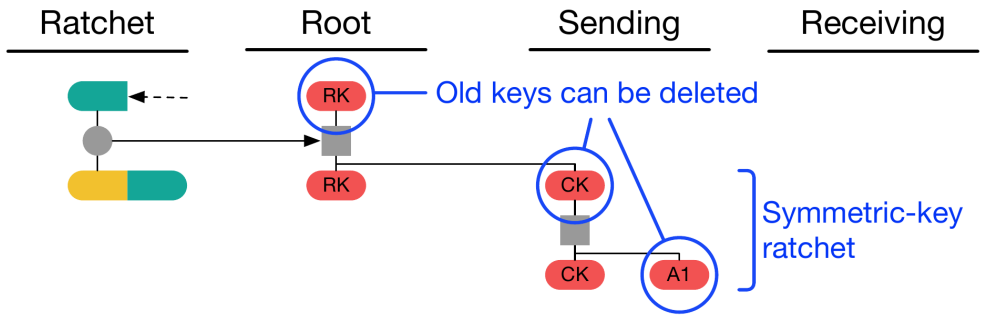


Figure 2: First double ratchet message key A1 for Alice (reproduced from [35]).

Figure 3 shows the computations that Alice does when receiving a response from Bob, which includes his new ratchet public key. Alice applies a new DH ratchet step to derive a new receiving and sending chain keys. The receiving CK is used to derive a new receiving CK and a message key, B1, 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.

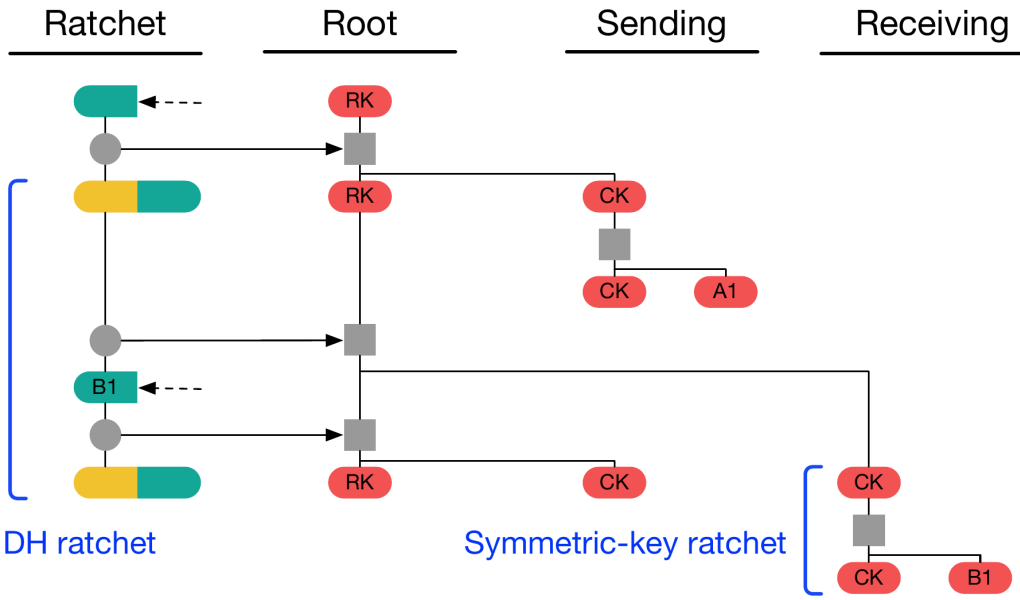


Figure 3: First receiving double ratchet message key B1 computed by Alice (reproduced from [35]).

2.5.2 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) and the length (number of message keys) in the previous sending chain (PN) [35]. This allows the receiver to advance the keys to the relevant message key, while still storing the skipped message keys in case they receive an older message at a later time.

Consider the example from Figure 4 where we assume that Alice has already received message B1, and now she receives 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 (i.e., 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 (i.e., B3). After Alice has stored B2 and B3, she can derive the last message key to decrypt message B4.

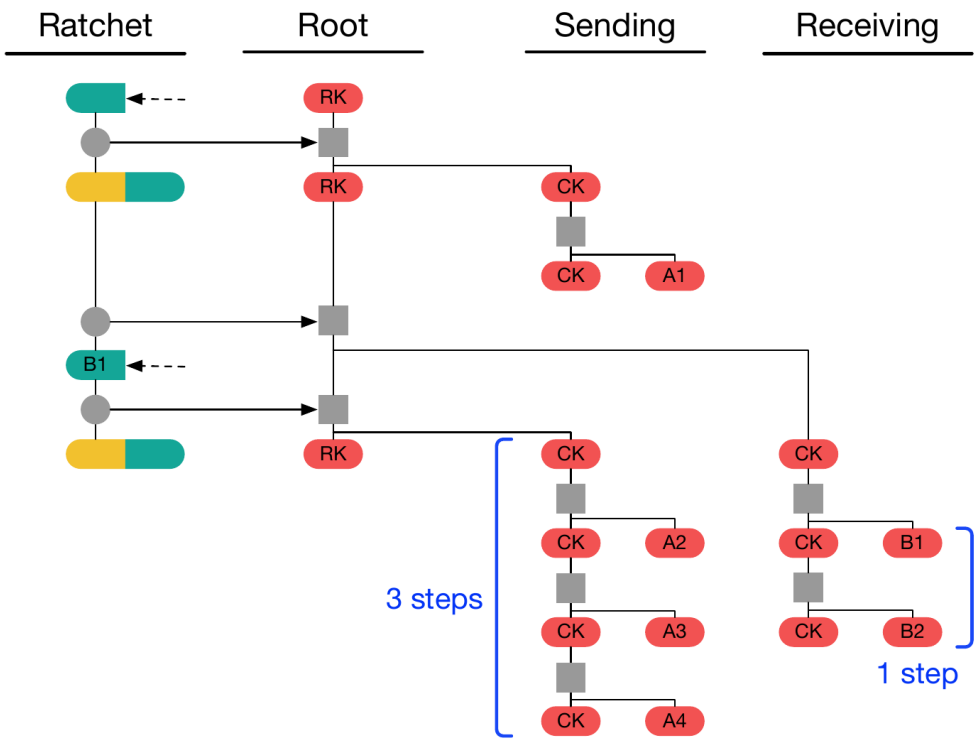


Figure 4: Handling of Out-of-Order Messages (reproduced from [35]).

2.5.3 The X3DH Key Agreement Protocol

For Signal the X3DH is designed for asynchronous settings where one user, Bob, is offline but has published information to a server, and another user, Alice, wants to use that information to send encrypted data to Bob [36]. The Extended Triple Diffie-Hellman key agreement protocol (X3DH) thus establishes a shared secret between two parties who mutually authenticate each other based on public keys, and at the same time provides both forward secrecy and cryptographic deniability.

To provide asynchrony, a server is used to store messages from Alice and Bob which later can be retrieved; and the same server keeps the sets of keys for Alice and Bob to retrieve when needed [36].

The X3DH Protocol has three different phases:

1. Bob publishes his elliptic curve public keys to the server: (i) Bob's identity key IK_B, (ii) Bob's signed prekey SPK_B, (iii) 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 the 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. The server will delete a one-time prekey each time it sends it to another user. New signed prekey signatures need to be uploaded periodically, and it is up to Bob when he wants to upload them, e.g., every day, once a week, or once a month [36].

2. Alice fetches from the server Bob's identity key, signed prekey, prekey signature, and optionally a single one-time prekey. If the verification of the prekey signature fails the

protocol is aborted. Otherwise, Alice generates an ephemeral key pair with her public key EK_A, and will use the prekey to calculate several DH keys with the purpose to provide mutual authentication and forward secrecy (see details in [36]) used to generate the secret key for encryption (SK). After calculating the SK, Alice will delete her ephemeral private key and the DH outputs to preserve secrecy.

Alice uses the key to send an 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 [45] 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 in the case of Signal.

3. Bob receives and processes Alice's initial message. Bob will load his identity private key and the private key(s) corresponding to the signed prekey and one-time prekey that Alice used [36]. 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 tries to decrypt the initial ciphertext. The decryption is the only difference between what Bob does and 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 [36]. 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, in order to uphold the forward secrecy.

2.6 Security properties of protocols for end-to-end encrypted instant messaging

Table 1 shows that none of the secure messaging protocols we have gone through in this section can give the users every security property (for more information about the definition of the properties used in Table 1 please refer to [49]). In this section we briefly comment on each of these.

While the Off-the-Record protocol does not need any additional services or servers, it cannot provide group conversation (in the current version and implementations). There have been research works investigating group conversations on top of OTR [5, 31, 22], but they have not received enough attention from the developers mainly because these do not support asynchronous chat conversations.

While Signal supports desktops through the Chrome Extensions, it does not support native desktop application. Moreover, it only allows for one device to be used, i.e., multiple mobile phones cannot be added to a user's account. This could be achieved using the same functionality for group conversations, but efficiency could be a problem.

The Matrix protocols and application (see Section 2.6.3 for details) supports multiple devices, without affecting the efficiency of the conversations. However, it does not achieve full forward and backward secrecy in the protocols, but the implementation does.

The Signal protocol has been audited by two research groups in 2016 [11, 29] and since it is open source, the community can improve it. The Matrix protocol has also been audited

Table 1: Comparison of secure messaging protocols (reproduced from [49]).

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

●: Provides the property;
◑: Partially provides the property;
-: Does not provide the property.

[3]. This indicates that researchers are taking these protocols seriously and want to strengthen their credibility.

2.6.1 Off-the-Record

OTR uses an encrypt-then-MAC approach to protect messages (see Section 2.4 for details) which provides *confidentiality, integrity, and authentication*. The SIGMA protocol (a variant of authenticated Diffie-Hellman key exchange) ensures *participation consistency* for the key exchange [30]. *Forward secrecy* is ensured by the fact that message keys are regularly replaced with new key material during the conversation. *Backward secrecy* is ensured by the fact that message keys are computed by new DH values which are advertised by the sender with each sent message. *Anonymity preservation* ensured by the fact that the long-term public keys are never observed, neither during the key exchange nor during the conversation. *Causality preservation* is only partially achieved, as messages implicitly reference their causal predecessors based on which keys they use [49]. *Speaker consistency* is only partially achieved since an adversary cannot drop messages without also dropping all future messages, for otherwise the recipients would not be able to decrypt subsequent messages [49]. The aftermath of the speaker consistency is that the recipient needs to save 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 would end up only partially providing the forward secrecy. *Out-of-order and dropped messages* are only 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 that message. The problem of storing the decryption key is that it raises the possibility of successful attacks by adversaries.

The OTR protocol signs the messages with the shared MAC keys and not the long-term keys. To strengthen the *message unlinkability* and *message repudiation* features, OTR ues malleable encryption and the MAC keys are published after each message exchange [8]. OTR only signs the ephemeral keys and not every parameter during the key exchange, which provides only *partial participation repudiation* since the conversation partners can use the signed ephemeral keys to forge transcripts.

The OTR protocol is intended for instant messaging, and thus does not provide asynchronous messaging between participants. However, the synchronous only requirement allows OTR to not rely on additional services for establishing a connection between two participants.

2.6.2 Signal

The design of the Signal protocol, presented in Section 2.5, extends OTR is made up of a Double Ratchet algorithm, 3-DH Handshake, and prekeys for the asynchronous ability it provides. The Double-Ratchet is 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 of their use of thee KDF ratchets, whereas *backward secrecy* is provided because even when KDF keys are comprised, they are soon replaced by new keys. The X3DH handshake (Section 2.5.3) provides the same level of *authentication* as the SIGMA from OTR, but X3DH achieves full *participation repudiation* since anybody can forge a transcript between two parties [49]. However, Signal *fails to provide anonymity preserving* because X3DH uses the long-term public keys during the initial key agreement. The prekeys are used to provide an *asynchronous messaging* system by sending a set of prekeys 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. A predefined number of prekeys are uploaded to the 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, by attaching message identifiers, of the messages before, to the new message [49], but it cannot guarantee participant consistency. Multi-device is partly provided, in the sense that only an extra computer can join in a conversation by using the Signal Desktop application[17], which is only a Chrome Extension[18] and not an own application.

The Signal Protocol provides computational and trust equality, subgroup messaging, contractible and expandable membership 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 marking 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 equality as computationally demanding as if there was only a one-to-one conversation.

2.6.3 Matrix

The Matrix protocol consists of two different algorithms, the Olm[19] for one-to-one conversations and Megolm[20] 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.

Besides the same security properties as Signal, Matrix provides extra features through the Megolm algorithm. Multiple devices are possible with Matrix because megolm implements a separate ratchet per sending device that is participating in a group conversation.[21] 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 membership properties.

The NCC Group has audited both of the algorithms [3] and found 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[22]; the Megolm keys get refreshed after a certain amount of messages have been sent between the participants. Forward secrecy is only partially provided since the Megolm maintains a record of the ratchet value which allows them to decrypt any messages sent in the session after the corresponding point in the conversation [41]. The Matrix developers have stated that this is intentionally designed [25], but also said that it is up to the

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

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

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

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

[21] Matrix.org Launches Cross-platform Beta of End-to-End Encryption Following Security Assessment by NCC Group `https://pr.blonde20.com/matrix-e2e/`

[22] `https://about.riot.im/`

application to offer the user the option to discard old conversations [41].

3 Analysis of Applications Implementing Secure Messaging

This section surveys applications (mostly smartphone apps) that advertise secure messaging conversation capabilities between one-to-one and/or many-to-many users. We investigate a set of usability properties relevant for secure messaging.

3.1 Testing Method

For our tests, we used two separate smartphones as described in Table 2. 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 applications used during the testing phase are locked to one version number and did not get updates, in order to keep the tests consistent. Both devices have installed the same applications under test with identical version number.

Table 2: 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

3.2 Test Scenarios

We first explain in this subsection the test scenarios that we carried and the security and usability properties we are looking for. The test scenarios are the same for each application, and screenshots were taken during the testing phases to gather enough information for later analysis. We are going to study which properties the applications support for each test scenarios. These particular properties are chosen to discover how secure and usable an application is. 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

3.2.1 Setup and Registration

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

The properties that we are testing when performing this scenario are:

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

3.2.2 Initial Contact

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

The properties that we are testing when performing this scenario are:

- **Trust-On-First-Use:** Automatically verify each other on initiation.
- **Notification About E2E Encryption:** Does the app present notifications to explain to the user that messages are end-to-end encrypted?

3.2.3 Message After a Key Change

This scenario tests 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 normally 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 properties that we are testing when performing this scenario are:

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

3.2.4 Key Change While a Message Is In Transit

Cryptographic key changes while a message is in transit is similar to the test scenario before, however we are interested in 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 is the message lost forever?

The properties that we are testing when performing this 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 again.
- **Details about transmission of message:** Users can see the difference between sent and delivered messages.

3.2.5 Verification Process Between Participants

In a conversation, Alice and Bob want to verify each other, to ensure that they are having a conversation with honest participants. This test scenario looks at how the verification process works and if it is a secure and usable method of doing the verification between participants.

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.

3.2.6 Other Security Implementations

Each application may have additional security and privacy features meant to protect from various intrusions or attacks.

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

- **Passphrase/code:** Add a passphrase/code that only the user knows and enters it to gain access to the application.
- **Two-step verification:** When registering after a reinstall or new device, then a second passphrase/code is neede which only the specific user knows.
- **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.

3.3 Running The Different Test Cases

3.3.1 Case 1: Signal

Signal is an instant messaging application as well as a voice calling application, for both Android and iOS. What sets the Signal application apart from the other applications, except for Riot, is that it is completely open source. This reassures people who use it that it does what the developers claim since anyone can audit the source code and the cryptographic protocol that is used.

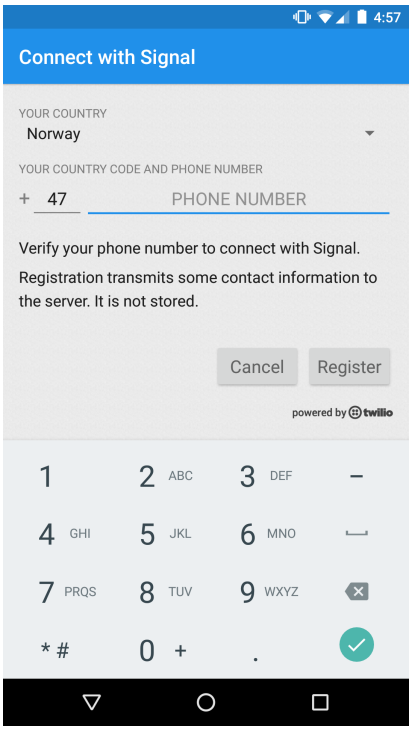
Initial Set Up:

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 will be deactivated automatically, to strengthen the security and to keep the private cryptographic keys on one device only.

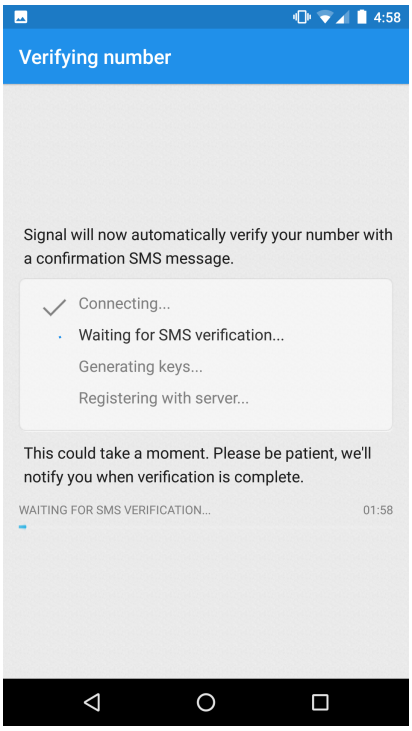
Figure 5a shows the first view a user sees when opening the app for the first time. Twilio[23] is used for the handling the SMS verification process with the Signal server when registering an account. Contact information are transmitted to the server, but are not stored.

Figure 5b 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 to be used in conversations between participants for end-to-end encryption. At the end the app registers the account within the Signal 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 Fig. 5b. When the timer has timed out, the Signal application calls the user and gives out a verification number to be typed in manually.



(a) phone number registration



(b) verifying the phone number

Figure 5: Signal: registration process

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

Message after key change:

This test scenario aims to check what happens when the cryptographic keys change, e.g., when a user in a conversation deletes and then reinstalls the Signal app. Figure 6a shows the first two messages that Alice has sent to Bob. The *double checkmark* shown on each message in Figure 6 indicates that the message has been received and read by Bob. The lock on each message indicates that the message is encrypted from one end to the other end and nobody in between can read it.

Figure 6b 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 that by pressing on the message the user can get more details about the notification.

Figure 6c is the view the user sees when pressing the message that was not delivered in Figure 6b. Alice is 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 to 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 is posted in the conversation that Bob has changed his security number, as shown in Figure 6d.

Key Change While a Message Is In Transit:

This test scenario is mostly the same as the previous one, with the difference that here we want to check what does the Signal app do when a message is sent before Bob has managed to reinstall his Signal application, i.e., handling of messages lost in transit.

Figure 7a shows the initialisation of the conversation between Alice and Bob. Figure 7b 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 icons on the third message sent by Alice indicate that Bob has finally reinstalled, but he never received the second message which was sent before he reinstalled. After Alice and Bob verify 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 7b 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.

Verification Process Between Participants:

Signal supports three different methods for the users to verify each other.

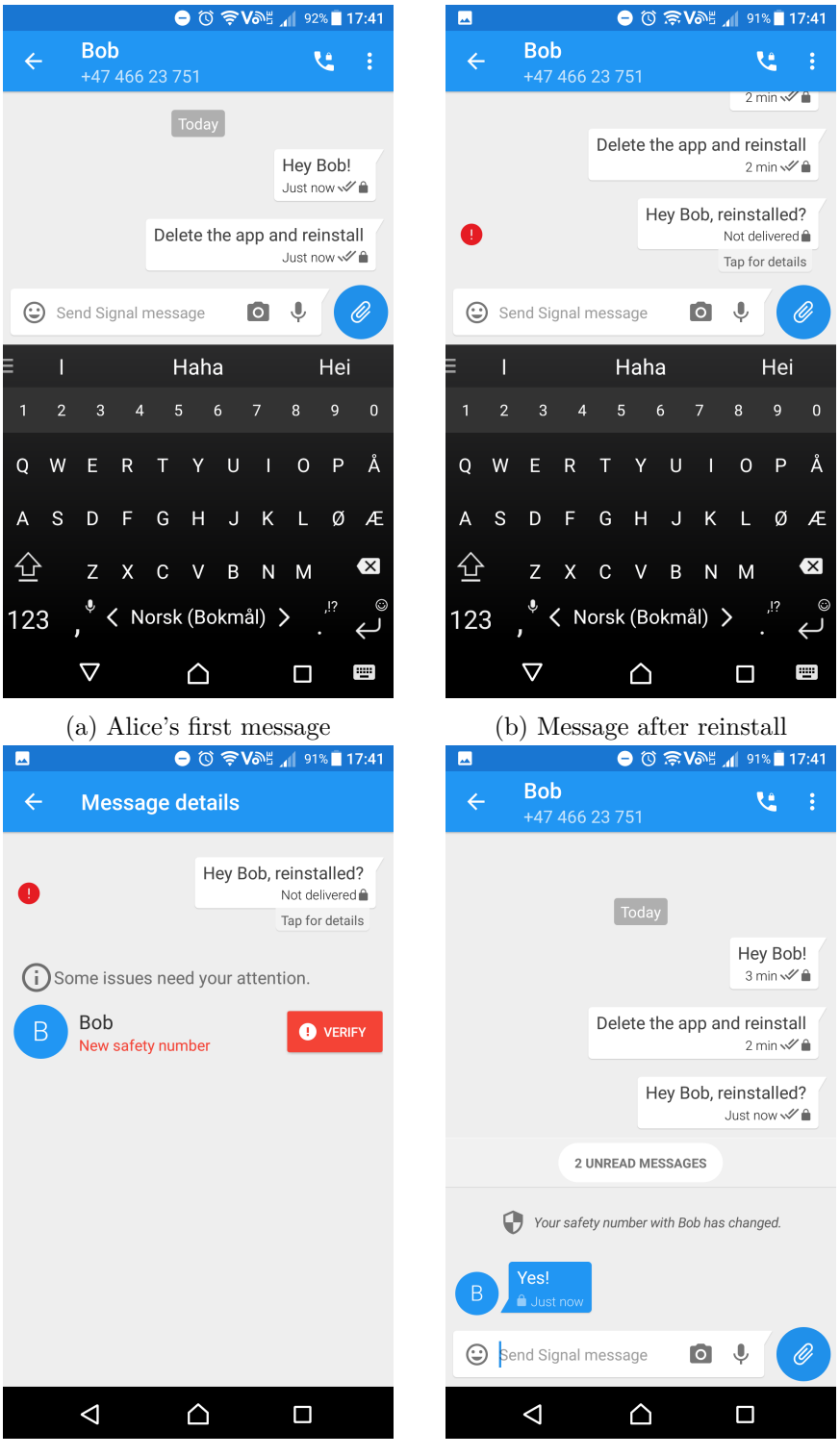
The first verification process uses the built-in calling option of Signal which is end-to-end encrypted and then read out loud to the other participant the security numbers that are shown in Figure 8a. If the Signal calling is not seen by the participants as secure enough, they can meet in person and read the numbers out loud to each other.

The second method is using QR-codes[24], shown in Figure 8a. The Signal app has a built-in QR-code scanner as well, which can be used 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 meant to be used when the users do not trust the Signal application for 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[25] [51, 19] 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.

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

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



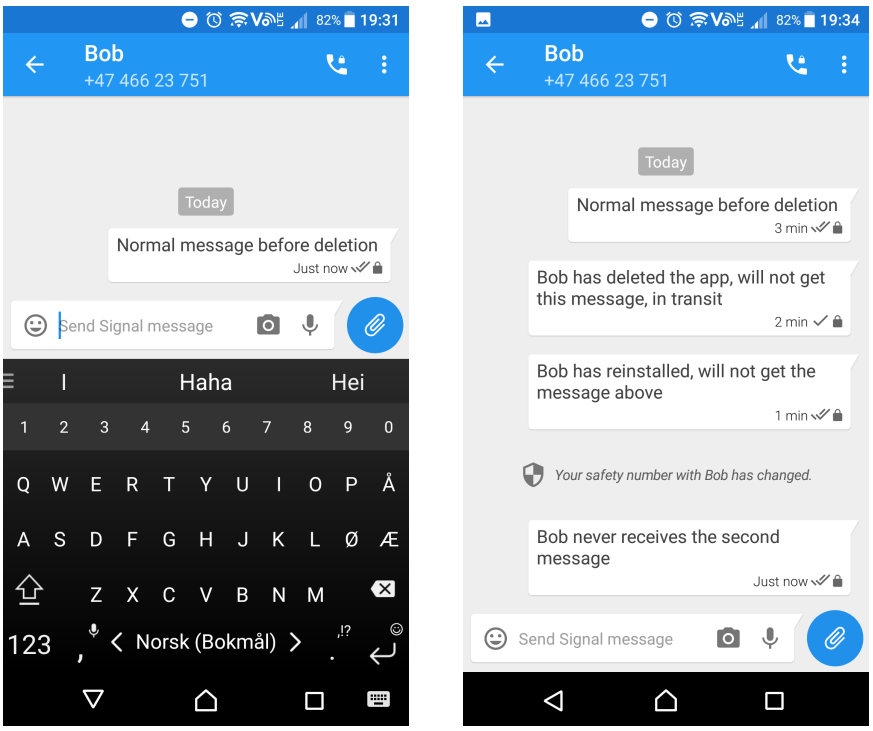
(a) Alice's first message

(b) Message after reinstall

(c) Verifying Bob again

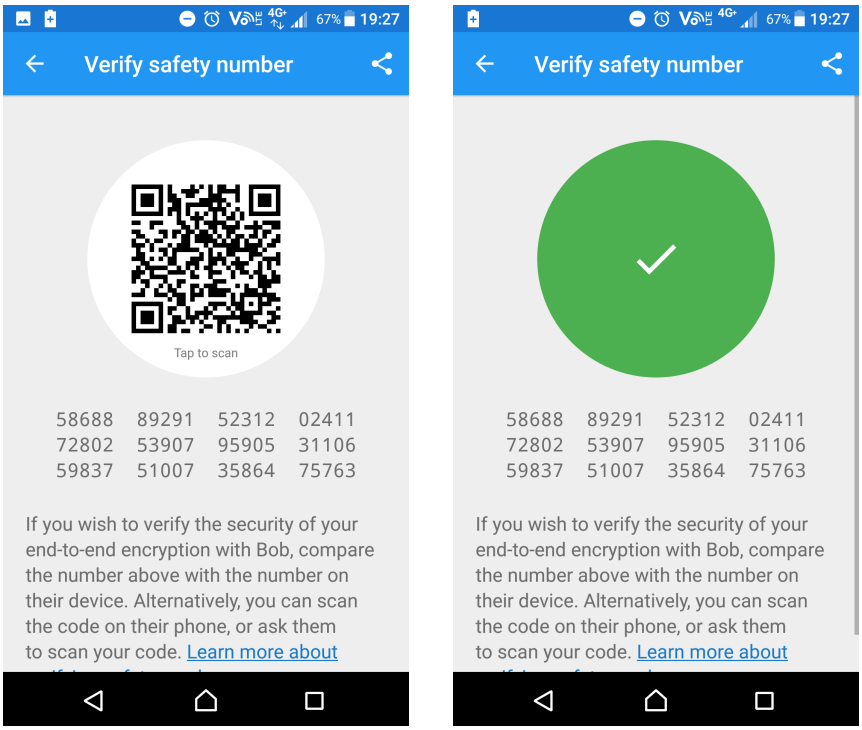
(d) Message after verification

Figure 6: Signal: Message after key change



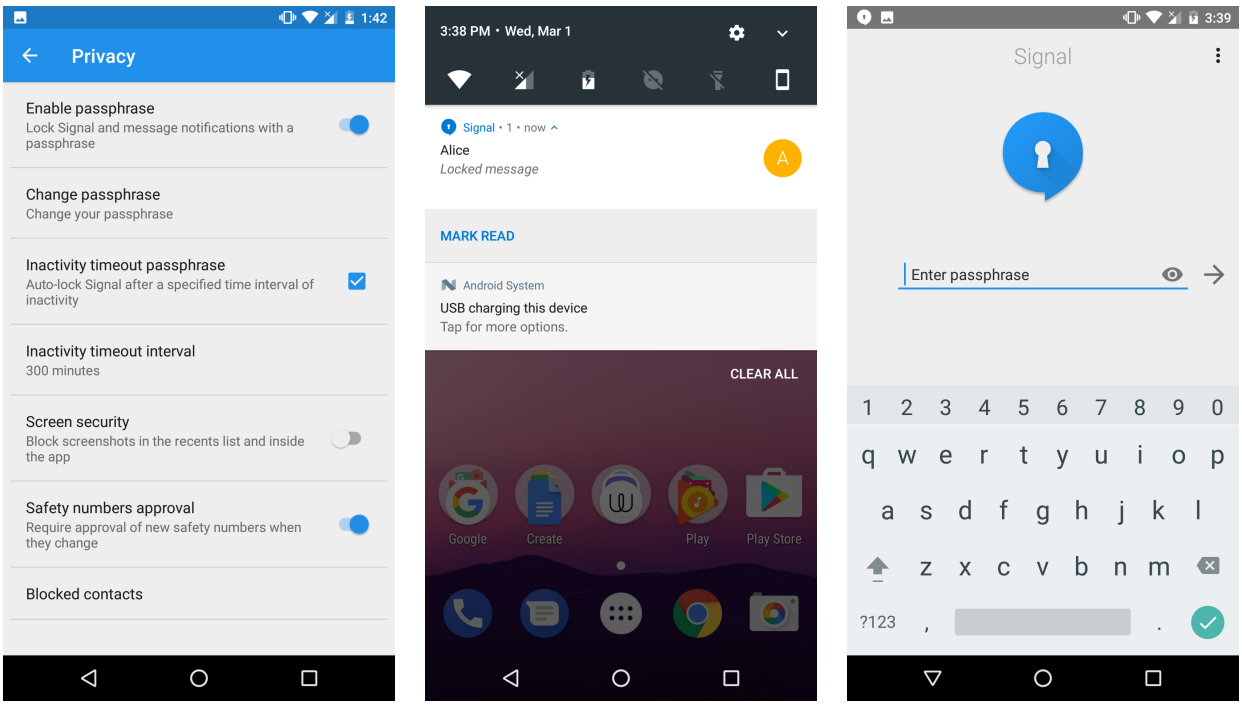
(a) Message before key change (b) Message after key change

Figure 7: Signal: Key change while message in transit



(a) Verification page (b) Verified

Figure 8: Signal: Verification process



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

Figure 9: Signal: Other security implementations

Other Security Implementations:

The Signal application has extra privacy settings. The first extra privacy setting is the "Safety numbers approval" as seen in Figure 9a. The setting is activated by default, which is important. When a user changes the safety numbers (cryptographic keys) by deleting and reinstalling the app, the device keys will also change. When the device keys change, the messages will not be shown to the user receiving the messages from a new device, until the new safety numbers are approved.

The second privacy setting is "Screen security", which does not allow the user to take screenshots as long they are inside the Signal application.

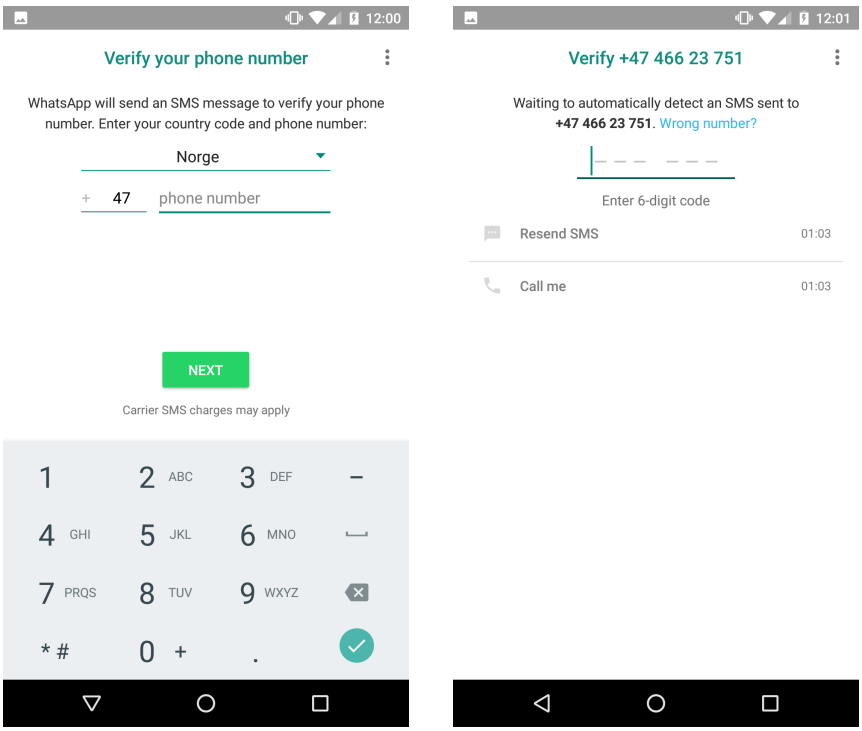
The last privacy setting is the ability to enable a passphrase. The passphrase locks the Signal application and all message notifications. It is possible to add an inactivity timeout passphrase which locks the application after some given time. Figure 9b shows the notification which is locked and when the user tries to open the application, the screenshot in Figure 9c shows that the user needs to enter their passphrase they chose when the setting was activated.

3.3.2 Case 2: WhatsApp

WhatsApp started as a small company in 2009, bought by Facebook in 2014 when it had 465 million monthly active users, and in 2017 that number has grown to 1.5 billion.[26] 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.[27] This was an important step for Open Whisper Systems and the Signal protocol since now the most popular instant messaging application would use their protocol.

[26]The Statistics Portal: *"Number of monthly active Whatsapp users worldwide from April 2013 to December 2017 (in millions)"* `https://www.statista.com/statistics/260819/number-of-monthly-active-whatsapp-users/`

[27]"Open Whisper Systems partners with WhatsApp to provide end-to-end encryption", announced by Moxie Marlinspike from Open Whisper Systems on November 18 2014, at `https://whispersystems.org/blog/whatsapp/`



(a) Phone number registration (b) Verifying the phone number

Figure 10: WhatsApp: Registration process

In April 2016 a complete transition was made from non-secure messaging to fully end-to-end encryption.[28]

Initial Set Up:

Establishing an account on WhatsApp is done in the same way as for the Signal application, where the account only works on one device at a time.

Figure 10a shows the first page a user sees when starting the application for the first time. WhatsApp uses their own infrastructure to handle the SMS verification process instead of a 3rd party such as Twilio that Signal uses.

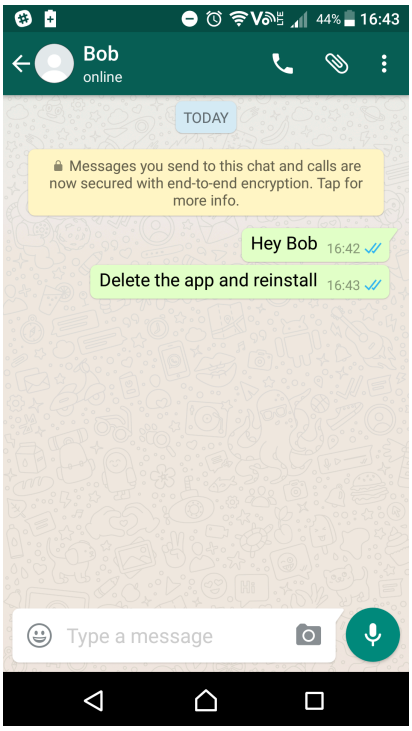
Figure 10b shows the verification page after the user has entered her phone number. WhatsApp automatically enters the verification code that is sent to the user's SMS inbox, but if the user has not given the app access to the inbox, she can enter the verification code manually. If for some reason the verification code does not arrive, the user has the options to either resend the SMS or ask WhatsApp to call the user to receive the verification code through voice.

Message after key change:

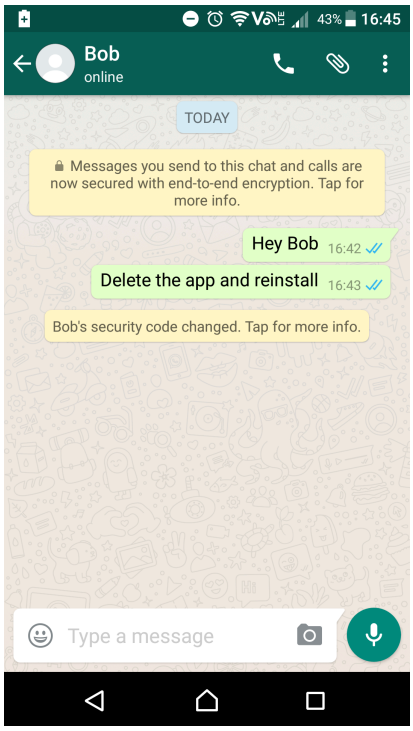
Figure 11a shows when Alice sends her first and second messages to Bob, for them to initiate a conversation together. The yellow notification box at the top of the conversation is WhatsApp notifying both participants that the conversation is end-to-end encrypted and they can read more about the encryption 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.

Figure 11b shows a new notification box appearing 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 11b, a popup informs Alice why Bob's cryptographic keys have changed and the option to verify him before she sends him new messages, as shown in Figure 11c. How the verification process works in

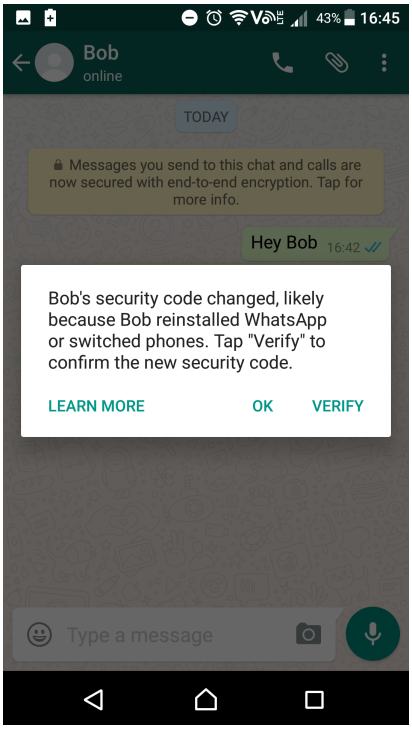
[28]"WhatsApps's Signal Protocol integration is now complete", announced by Moxie Marlinspike from Open Whisper Systems on April 05 2016, at `https://whispersystems.org/blog/whatsapp-complete/`



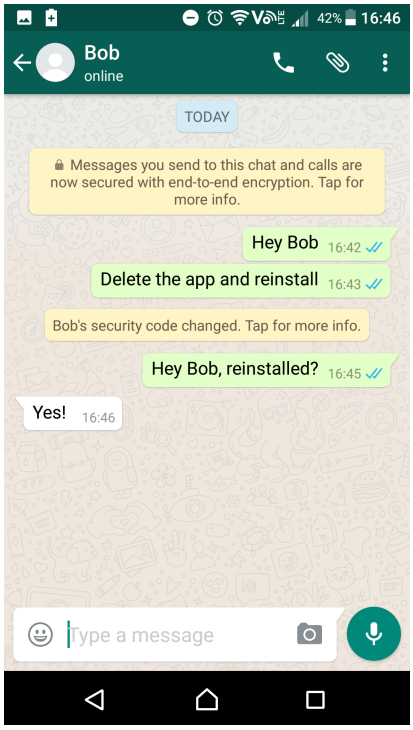
(a) Alice's first message



(b) After Bob has reinstalled

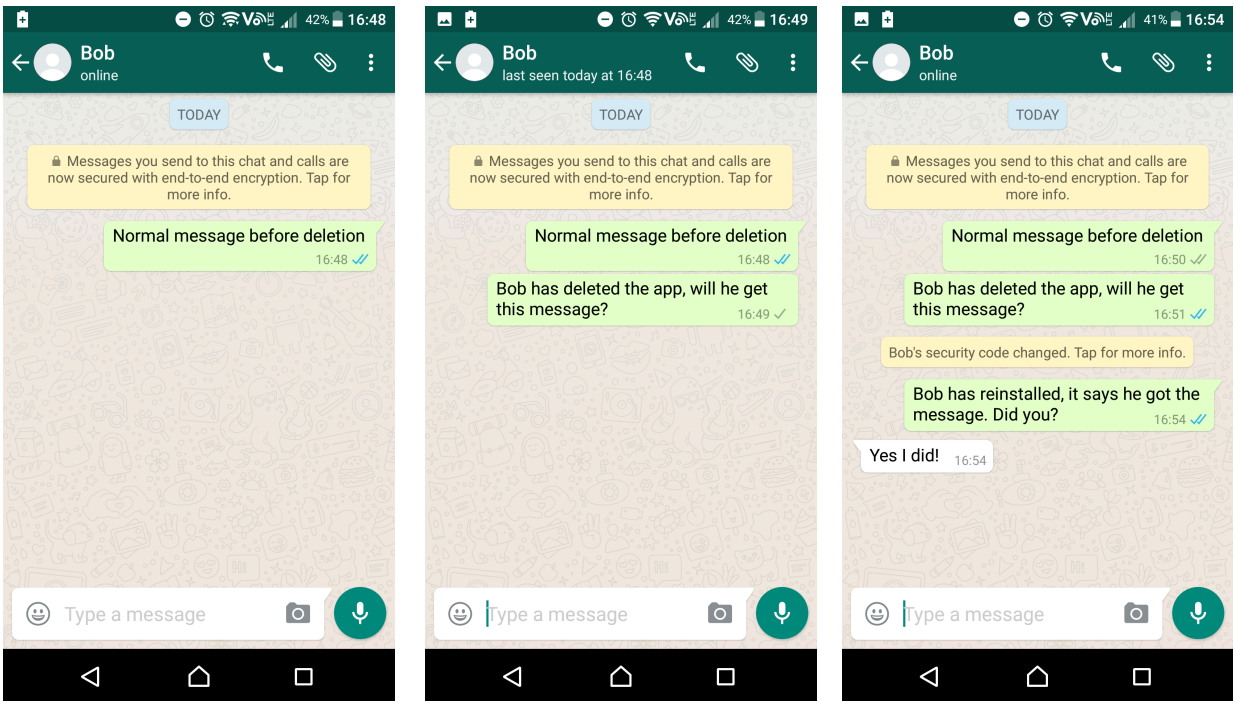


(c) Info about Bob's new keys



(d) Message after verification

Figure 11: WhatsApp: Message after key change



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

Figure 12: WhatsApp: Key change while message in transit

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. The message has again the double checkmark, informing that the message is received and is end-to-end encrypted, as shown in Figure 11d.

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 12a shows the initial message Alice sends to Bob. Figure 12b 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 finishes the reinstallation of the application, both the second message Alice sent and the same yellow notification box are added to the conversation. Figure 12b 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 created new cryptographic keys, without Alice verifying the keys first.

Verification Process Between Participants:

WhatsApp has implemented the same verification process as Signal. 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.

Other security implementations:

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

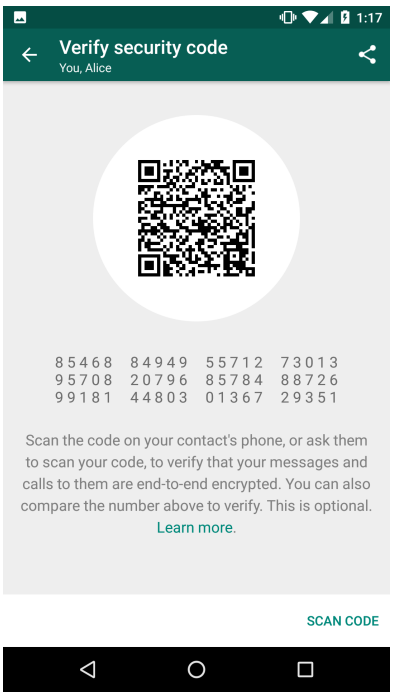
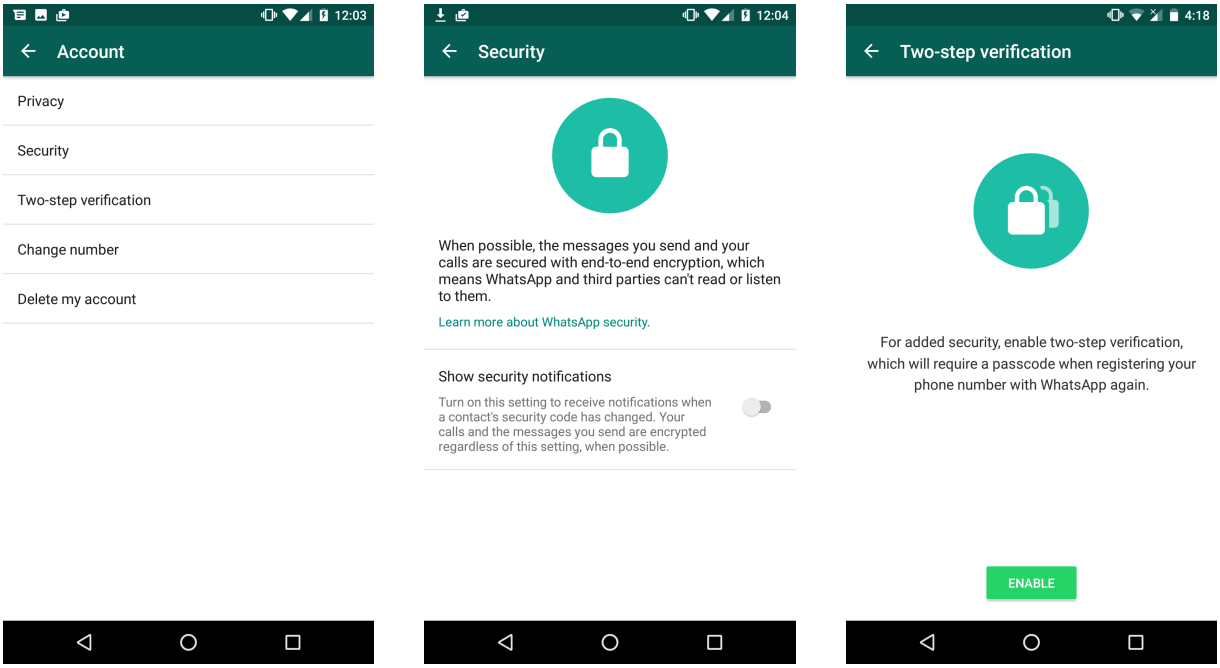


Figure 13: WhatsApp: Verification process



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

Figure 14: WhatsApp: Other security implementations

Figure 14a shows the settings page for the user's account, where we can see that there are also settings for changing the number of the account or delete the account.

When a user choose the "Security" menu item, the Figure 14b is shown. The "show security notification" option triggers a notification to Alice when Bob has either reinstalled the application or received a new device. This was the case in our earlier test scenarios about key changes. When the option is turned off Alice does not receive any notification.

Figure 14c 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.

3.3.3 Case 3: Wire

Wire is an application that implements end-to-end encryption using the protocol Proteus, which is heavily based on the Signal protocol, but re-implemented in-house.[29] Wire was started in 2012 by developers who previously worked at Microsoft and Skype, and finally released their own instant messaging application in 2014 [42]. The first version did not offer end-to-end encryption 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, but it allows also end-to-end encryption. Wire is supported on multiple platforms, from smartphones to personal computers, and is also open sourced.[30]

Initial Set Up:

The Wire app has a different registration process than the other applications. The first page, as can be seen in Figure 15a, asks to register a phone number, or specific to Wire, to log in with an email address. The latter is only supported by registering through their web application, not from the phone.

Figure 15b shows the verification process, where the user needs to enter the verification code manually, which is received in an SMS. If the user does not receive any verification code through the SMS, they can register the 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, however, the code is not entered automatically by the application, as the previous apps did.

When a user reinstalls the application or replaces the device, she does 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 have the option to register an email and password the first time they register their phone number. Figure 15c shows the option to log in with an email and password, and Figure 15d illustrates the log in function with a phone number.

Message after key change:

Figure 16a shows Alice's initial contact with Bob. Wire uses text under each message to explain if the message was delivered to Bob.

Figures 16b and 16c illustrate 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.

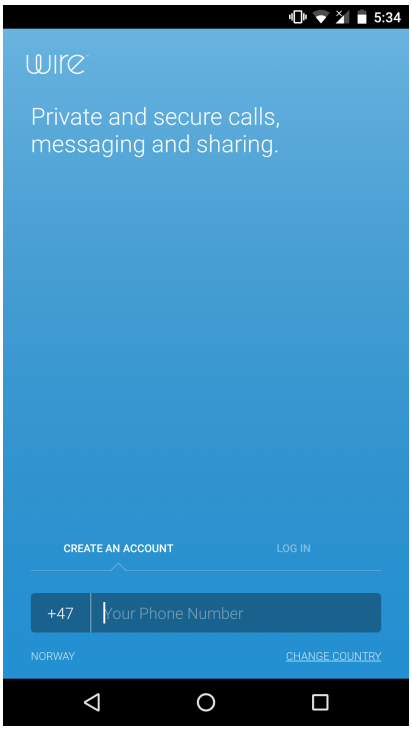
Alice can check Bob's account information to see if he has got new cryptographic keys. Figure 16d shows Bob's device keys under his account that he has new device keys after reinstalling the application, but Wire does not give any information to Alice about this. In this test scenario we can see that there are three different device keys under Bob's account because Wire allows multiple devices to be associated 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.

Key change while a message is in transit:

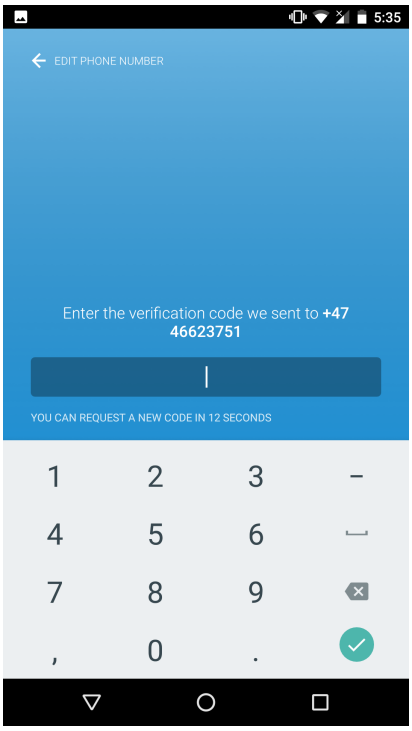
Figure 17a shows the initial message from Alice to Bob. Figure 17a is after Alice has sent a few messages to Bob: the second message is sent before Bob has reinstalled, showing that the message is only "sent" and not delivered; whereas the third message which was sent after Bob had reinstalled appears as "delivered". This shows that Wire does not notify Alice about

[29] Proteus Protocol, by Wire Swiss GmbH available at `https://github.com/wireapp/proteus`

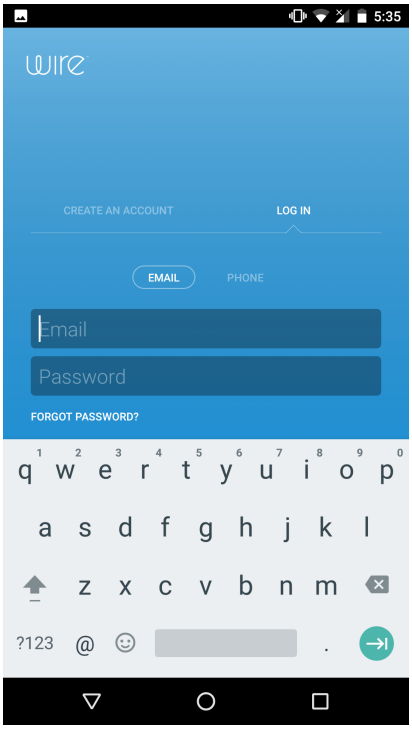
[30] `https://github.com/wireapp`



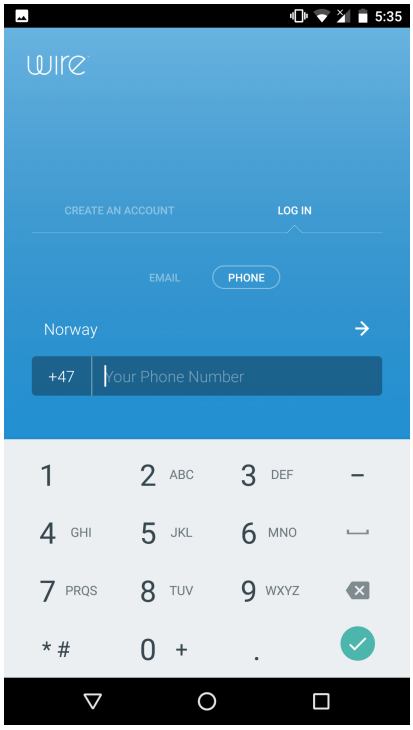
(a) Phone number registration



(b) Phone verification

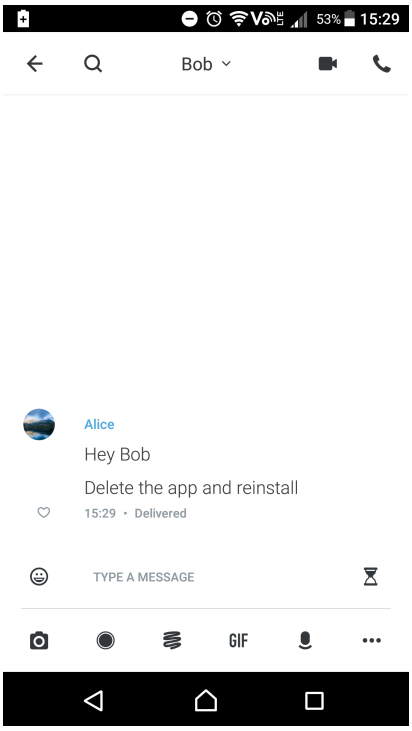


(c) User login with e-mail

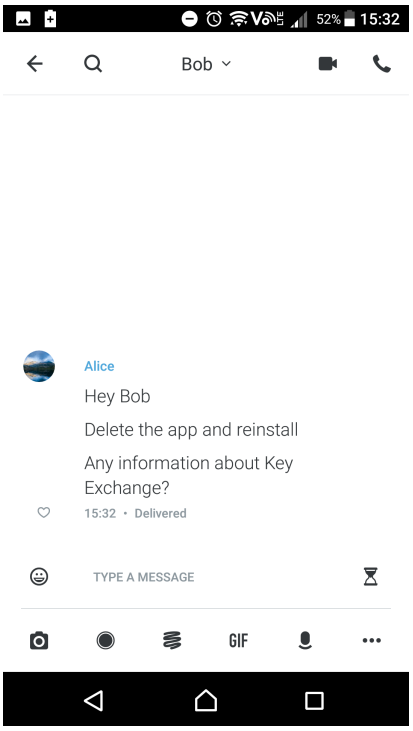


(d) Login with phone number

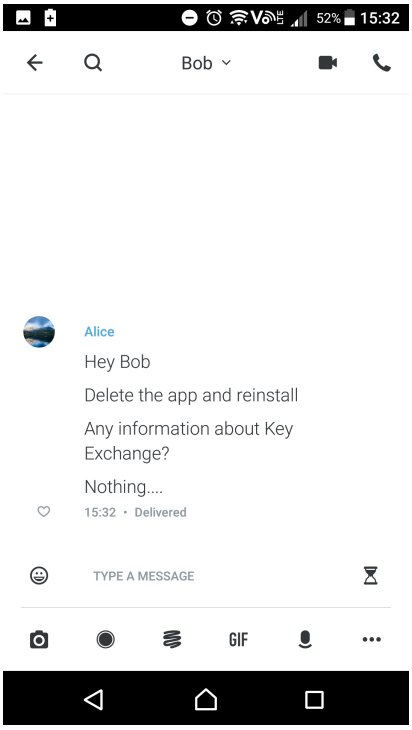
Figure 15: Wire: Registration process



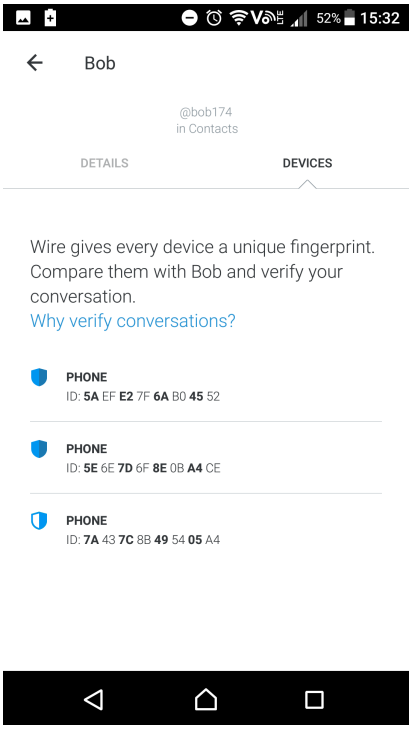
(a) Alice's first message



(b) After Bob has reinstalled

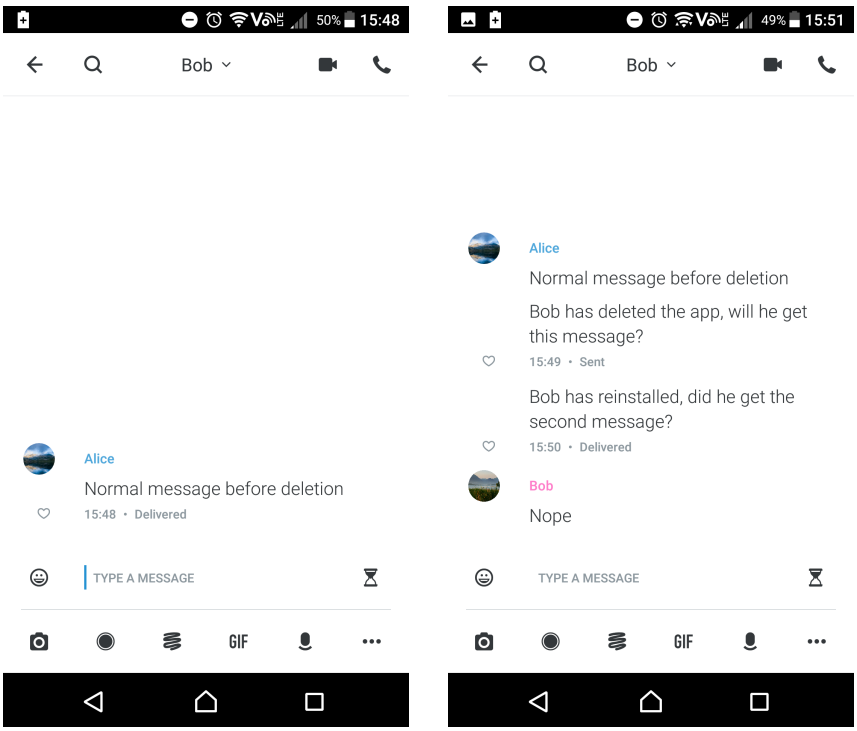


(c) No notification about key change



(d) Bob's device keys

Figure 16: Wire: Message after key change



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

(b) No notification after Bob had reinstalled

Figure 17: Wire: Key change while message in transit

Bob's new keys, and at the same time does not deliver messages with old cryptographic keys to devices with new keys.

Verification Process Between Participants:

Wire's verification process does not have the same options for verifying each participant as the other applications. However, because Wire allows several devices to be associated with one account each user has access to the whole list of devices of another conversation party.

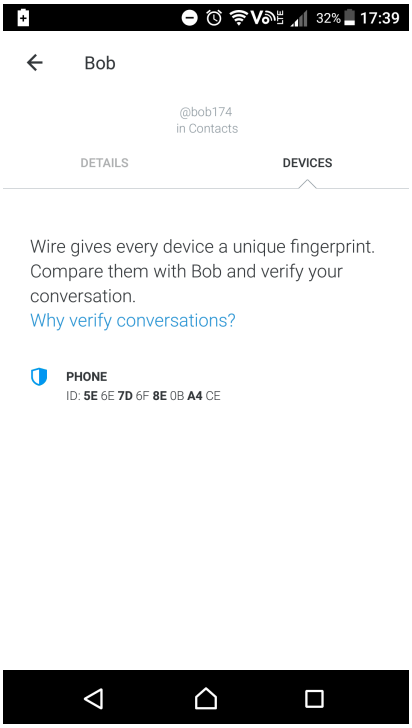
When Alice wants to verify one of Bob's devices, she can see Bob's profile, and particularly the tab displaying information about all his devices. Figure 18a shows the list of Bob's devices, where at the beginning of the testing he only has one set of device keys.

Figure 18b 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.

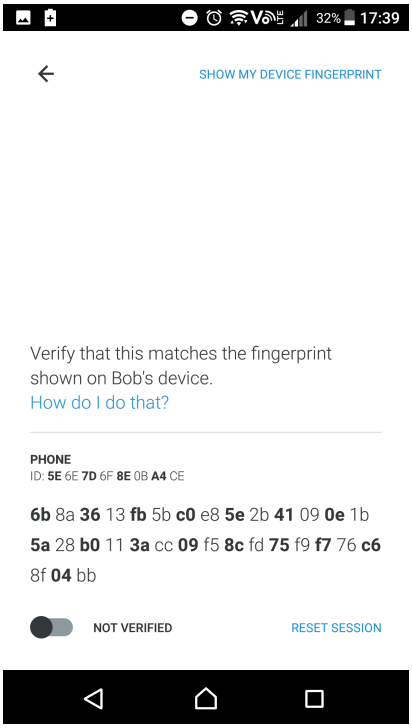
Other security implementations:

Wire does not have the extra security implementations that Signal or WhatsApp have. The few options include a way to change how the message conversation looks and the possibility to add an email to the account for easier log in.

The user can look at the devices which have been used with her account, and if there are any devices which the user does not own or recognise, she can delete that specific device. After a device is deleted, the user is prompted to change the password for that account.



(a) List of Bob's devices



(b) Bob's public keys for one device

Figure 18: Wire: Verification process

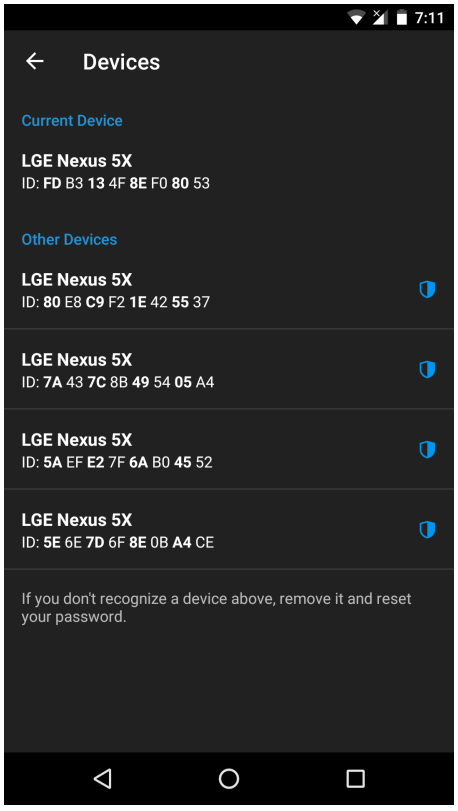
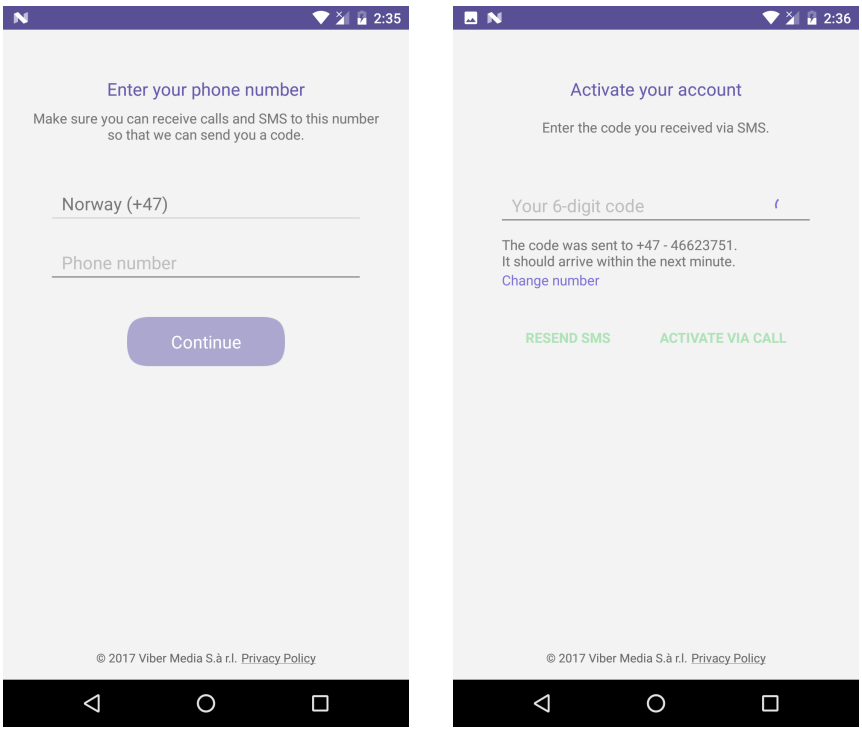


Figure 19: Wire: Other security implementations



(a) Registration with phone number (b) Verification of phone number

Figure 20: Viber: Registration process

3.3.4 Case 4: Viber

Viber is another instant messaging application that was launched in 2010, and has become quite popular, with 800 million overall users and 266 million monthly active users [32]. Viber has properties similar to the other applications, where users are capable of forming groups, send messages, call each other and send pictures, videos or voice messages to other users of Viber.[31] Viber works on smartphones and personal computers, making it cross-platform.

Viber did not have end-to-end encryption in the beginning, but introduced it in April 2016, for both one-to-one and group conversations.[32] Viber does not use the Signal protocol, but rather implement their own, which they have state that it has the same concepts as the double-ratchet protocol used by Signal.

Initial Set Up:

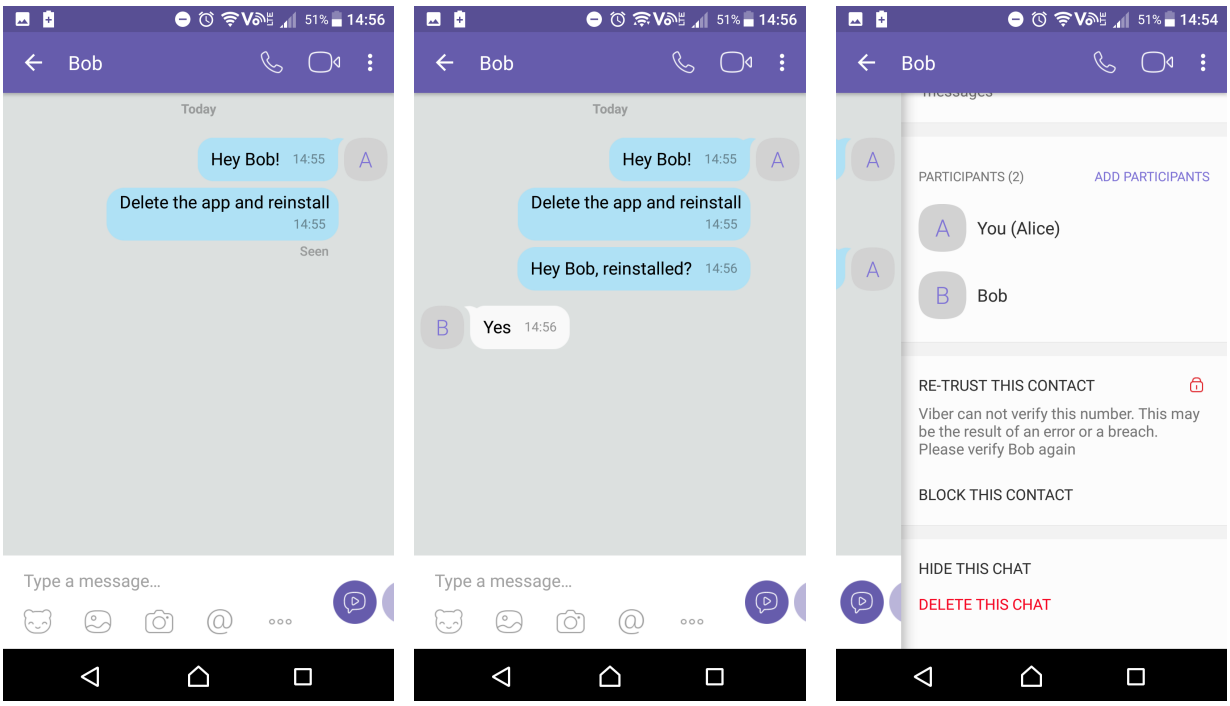
The user registration process of Viber is the same as in the previous applications. Figure 20a shows the user input for the user's phone number in the registration screen. Figure 20b shows the activation process of the user account. The user can either give Viber access to the SMS inbox to enter the verification code automatically or do it manually otherwise. If the SMS with the verification code does not arrive within one minute, the user can ask the application to either resend a new verification code or get the code through a phone call.

Message after key change:

Viber does not notify the participants when the cryptographic keys change during a conversation. The only way of knowing if the cryptographic keys have changed is for the participants in a conversation to verify each other first and then start this testing the scenario. The verification process is shown in a later test scenario.

[31] Viber, by Rakuten Inc. `https://www.viber.com/en/about`

[32] "Giving Our Users Control Over Their Private Conversations", by Michael Schmilov from Viber on April 19 2016, at `https://www.viber.com/en/blog/2016-04-19/giving-our-users-control-over-their-private-conversations`



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

Figure 21: Viber: Message after key change

Figure 21a shows Alice initiating the conversation with Bob. 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, see Figure 21b. 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, see Figure 21c, 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.

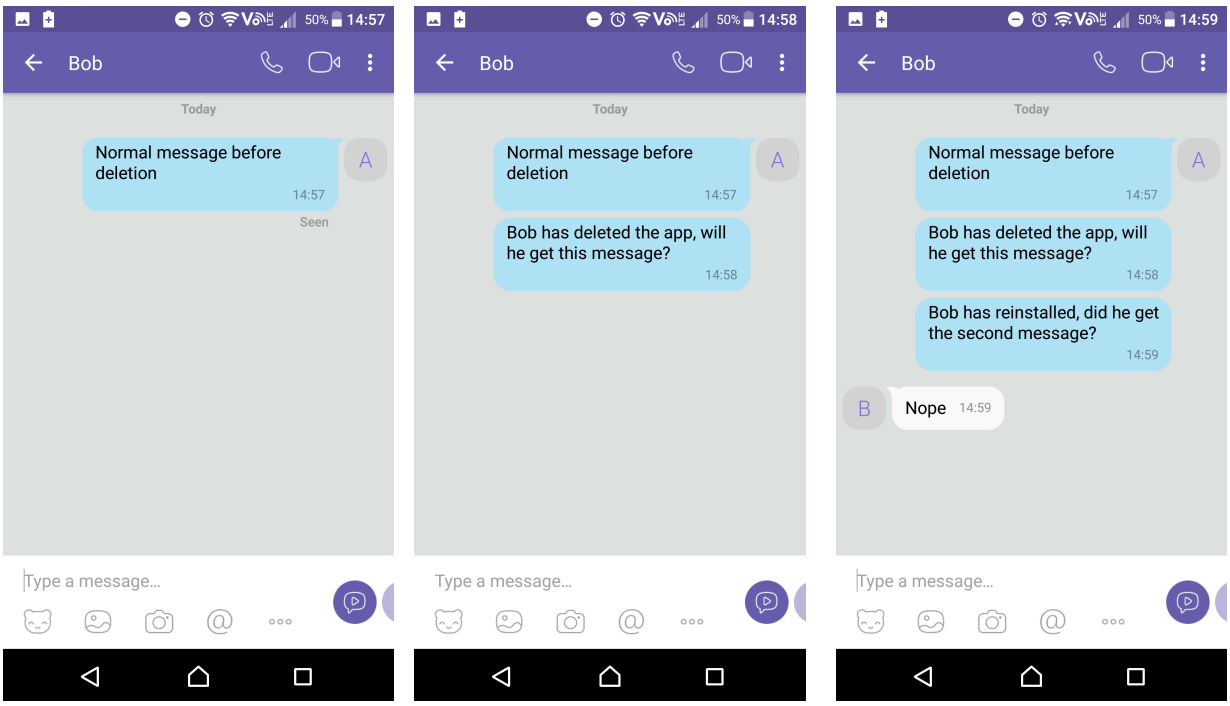
Key change while a message is in transit:

Key changes in transit are handled the same as key changes after the reinstall of the application as before. Alice initiates the conversation by sending a message to Bob right before he deletes the application (Figure 22a). Figure 22b 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 22c shows the third message from Alice to Bob after he is done reinstalling his application. Alice never receives any notification from Viber that Bob has new cryptographic keys nor that he has not received the second message. Moreover, Viber does not re-encrypt and re-send messages later on.

Verification Process Between Participants:

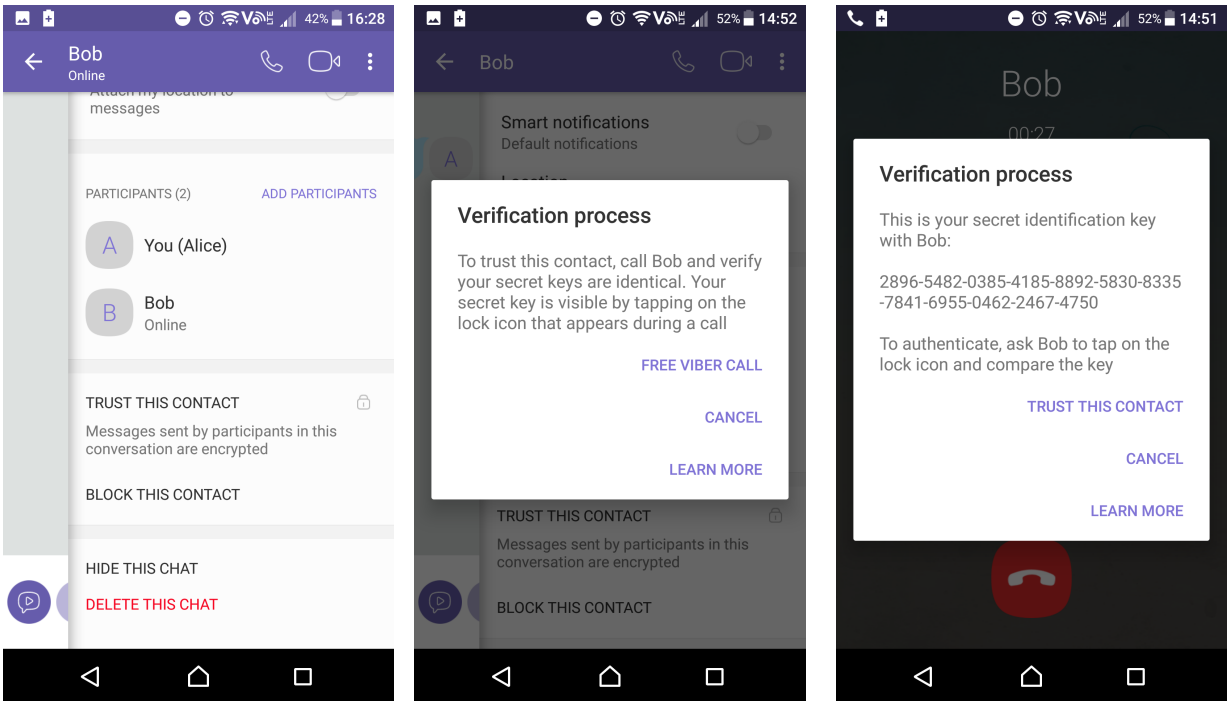
The process of verifying a contact in Viber is quite straight forward. If Alice wants to verify Bob she goes to one of their conversations, swipes from left to right to get the information tab, and then go down to the "Trust this contact" option (Figure 23a).

Figure 23b shows the popup notification box after Alice clicks 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.



(a) Alice's initial message to Bob (b) Message after Bob has deleted (c) Bob did not get the second message

Figure 22: Viber: Key change while message in transit



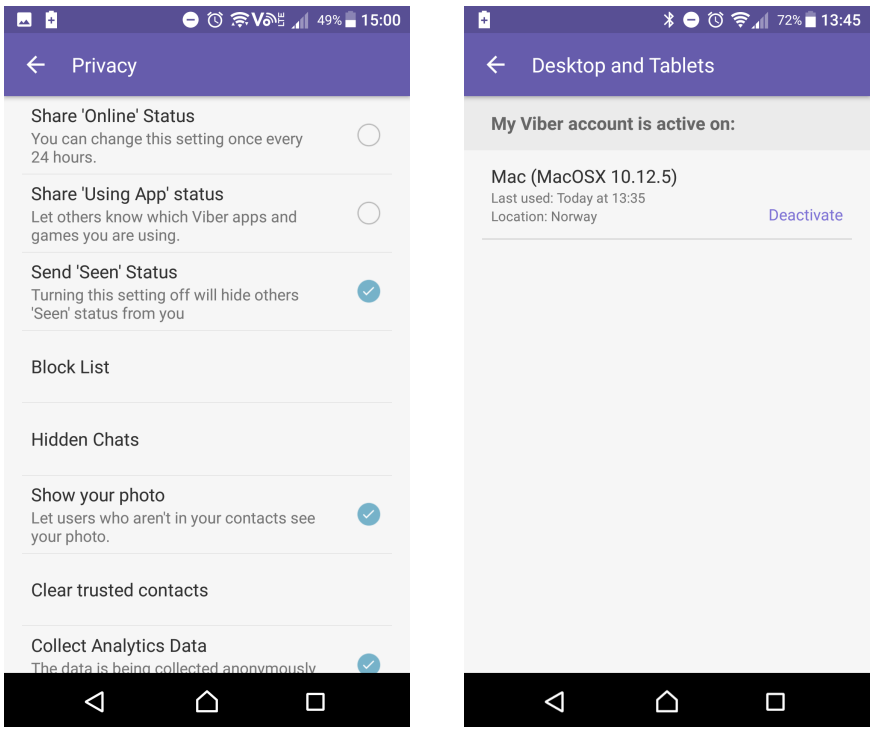
(a) Conversation info (b) Verify a contact (c) Alice verifying Bob

Figure 23: Viber: Verification process

Figure 23c shows when Alice calls Bob and wants to verify, the popup message displays the cryptographic keys that both Alice and Bob share. When they have verified each other, they press the "Trust this contact" button.

Other security implementations:

Viber does not have any extra security implementations. Figure 24 shows the privacy



(a) Viber: Privacy settings (b) Viber: Multiple Device List

Figure 24: Viber: Other security implementations

settings page where the only security implementation is the "Clear trusted contacts" which clears all the contacts that Alice has verified throughout the time they have had an account.

3.3.5 Case 5: Riot

Riot is a new chat client that is built on top of the Matrix[33] protocol for its end-to-end encrypted capabilities. Matrix is an open network for secure, decentralised communication platform which uses bridged networks and cross-platform possibilities plus full end-to-end encryption that is based on the Double Ratchet protocol from Signal.

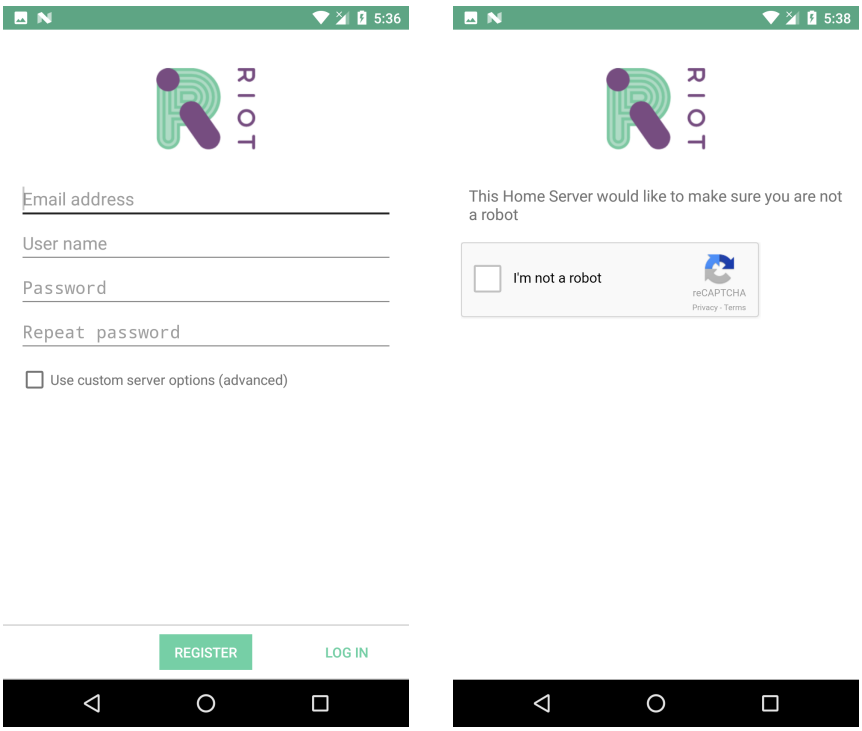
Riot uses servers, like Signal also does, but one does not need to rely on servers under the control of the Matrix team, the way Signal works. Riot and Matrix is open source, which means that anyone can set up their own servers with the Matrix implementation and use its end-to-end encryption. This is good for companies that want to have secure chat between employees but do not want to rely on anything from outside their own network. Riot does also have the same capabilities as other instant messaging applications, such as group chat, voice (VoIP) and video calling, file transfer and integration with other applications such as Slack[34] or IRC[35].

Initial Set Up:

The Riot application is the only messaging client that does not rely on a phone number, but a user registers an account with an email and username (Figure 25a). 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 25b shows the screen after the user has clicked the confirmation link and are then presented with a Captcha verification for an extra layer of security.

Message after key change:

[33] https://matrix.org/
[34] https://slack.com/
[35] https://www.wikiwand.com/en/Internet_Relay_Chat



(a) Registration by email (b) Captcha code

Figure 25: Riot: Registration process

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.

Therefore, Alice starts a chat room, invites Bob and then activates end-to-end encryption. End-to-end encryption is still in beta form, and thus is not turned on by default. Figure 26a shows the chat room, which in the beginning has open locks on each of the messages from Alice and Bob that have been sent before the encryption was toggled on. How the end-to-end encryption is toggled on is shown in the test scenario about other security implementations further down. When Alice sends her initial message to Bob, as shown in Figure 26b, the lock is changed to closed since the end-to-end encryption is on.

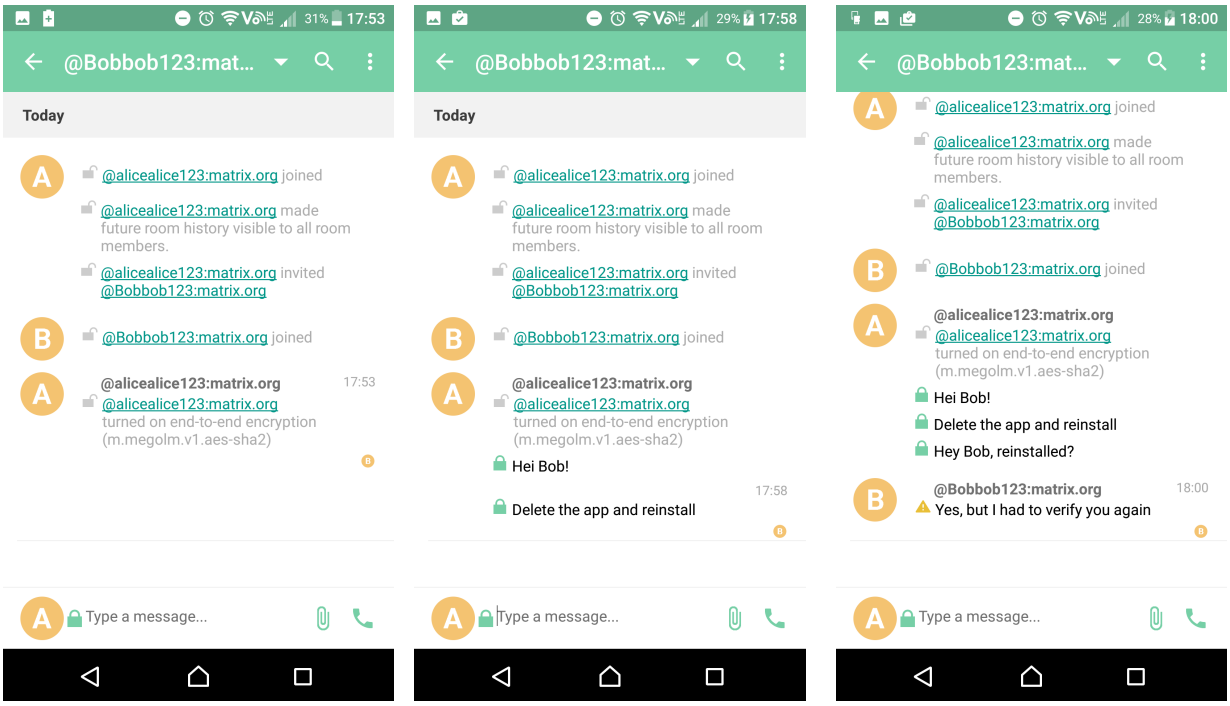
Figure 26c shows that after Bob reinstalled his application he had to re-verify Alice because his device keys were changed during the reinstall. Alice can also see that she has to re-verify Bob because his message has a yellow notification triangle showing that his new keys have not been verified and should not be trusted until that is done.

To verify Bob's new device keys, Alice can go into Bob's profile and look at the devices he has. Figure 26d shows how the devices are listed on Bob's profile account, where the yellow notification triangle indicates 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, as in Figure 26e.

When a new user (or existing user with new keys) enters the chat room, there should be no possibility of reading previously sent messages. Riot does this correctly, by not showing the previous messages when Bob enters the chat room after he reinstalls the application, but it does show him that there have been some messages exchanged between Alice and Bob earlier, as in Figure 26f.

Key change while a message is in transit:

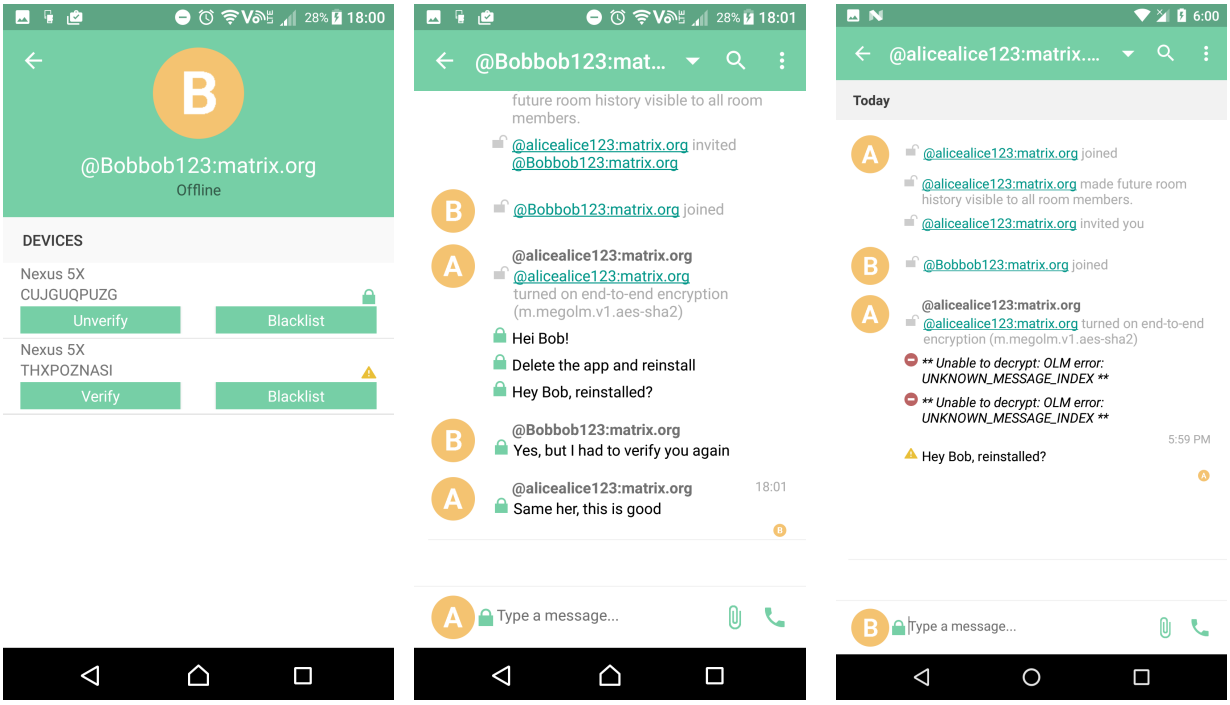
Riot handles key changes in the same way regardless of whether the message is in transit or not. Figure 27a shows Alice's initial message before Bob deletes his Riot application. After Bob



(a) Initial conversation, Alice and Bob connecting

(b) Alice's initial message to Bob

(c) Message to Bob after he reinstalled

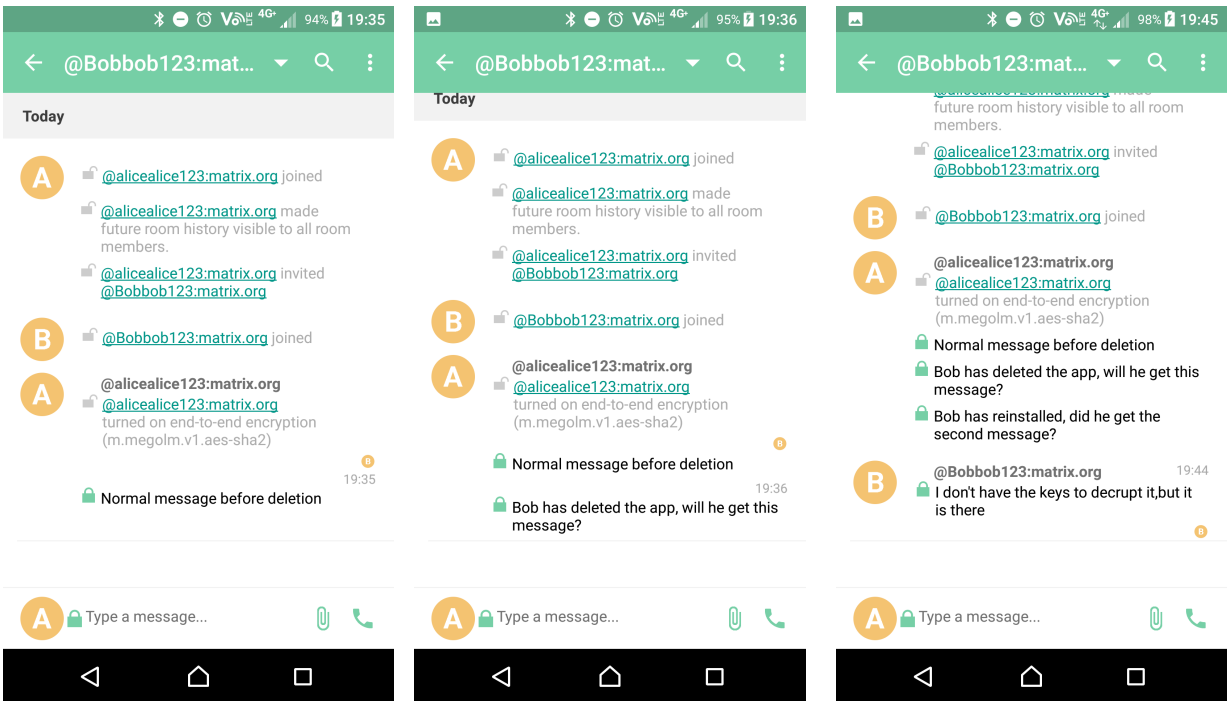


(d) Verifying Bob the 2nd time

(e) New messages are verified

(f) Bob's view of previous messages

Figure 26: Riot: Message after key change



(a) Alice's initial Message to Bob (b) Message before Bob reinstalled (c) Bob's message after reinstall

Figure 27: Riot: Key change while message in transit

has uninstalled, Alice sends her second message (Figure 27b) before he finishes reinstallation. Figure 27c shows that after Bob has reinstalled the application he can read the third message from Alice but cannot read the second message, which would appear the same as in the previous scenario where he could not decrypt. 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 got new encryption keys and lost the old ones.

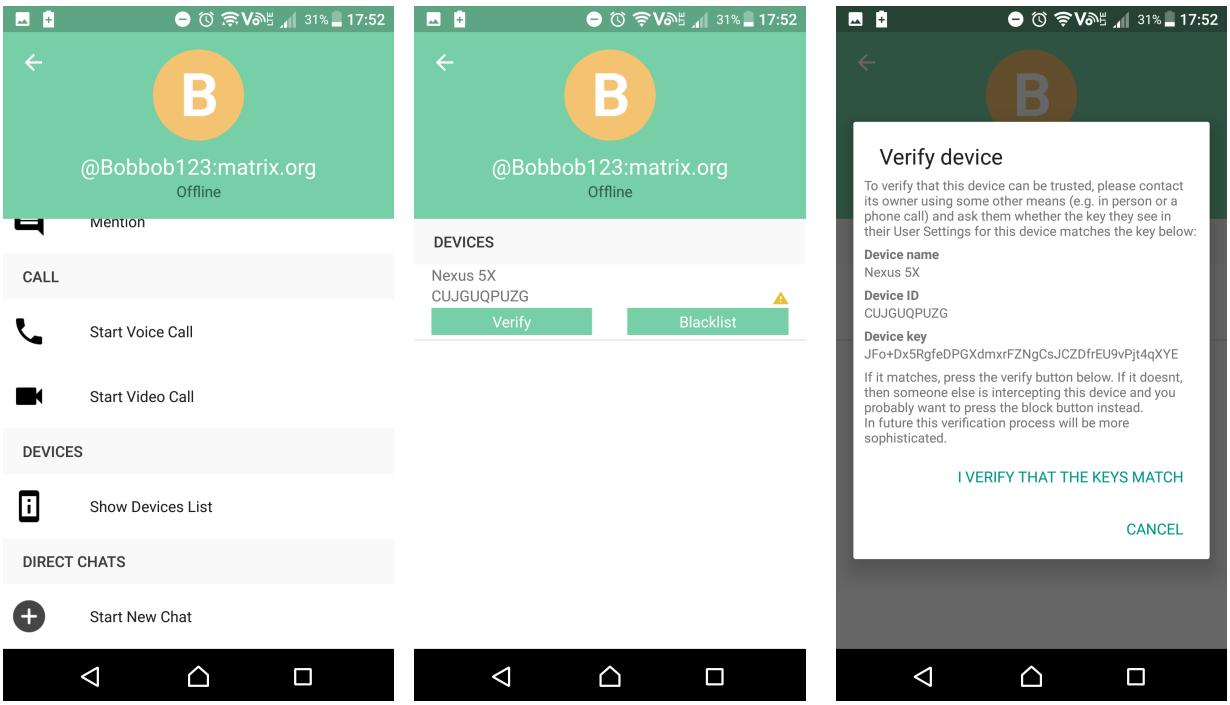
Verification Process Between Participants:

The verification process is rather easy in the Riot application. Moreover, Riot gives considerable amounts 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 that need to be verified, by clicking on the "Device" tab, as shown in Figure 28a. Alice sees a list of Bob's devices in Figure 28b, and if one of the devices has the yellow notification triangle, then that specific device has not been verified yet by Alice. She can either verify the device or blacklist it, as in Figure 28c, which means she cannot receive any messages or invites from that specific device.

If Alice decides to verify Bob's device, she clicks on the verify button and sees the verification popup from Figure 28c. The popup is informative for users, but for some end-users it may have too much information and could look cluttered. The popup states that the verification information and process will become more sophisticated in future versions 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 in person and in the end press the "I verify that the keys match" button.

Other security implementations:

Riot does not have that many extra security implementations, but since the application is only in beta, they may be implemented in future versions. Figure 29 shows the settings page providing details about a chat 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



(a) Profile details (b) List of Bob's devices (c) Verifying Bob

Figure 28: Riot: Verification process

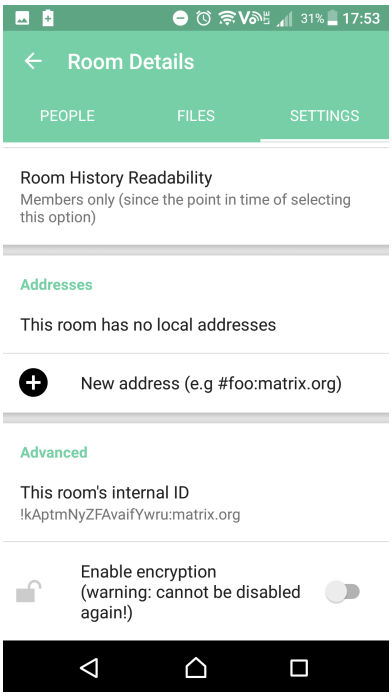
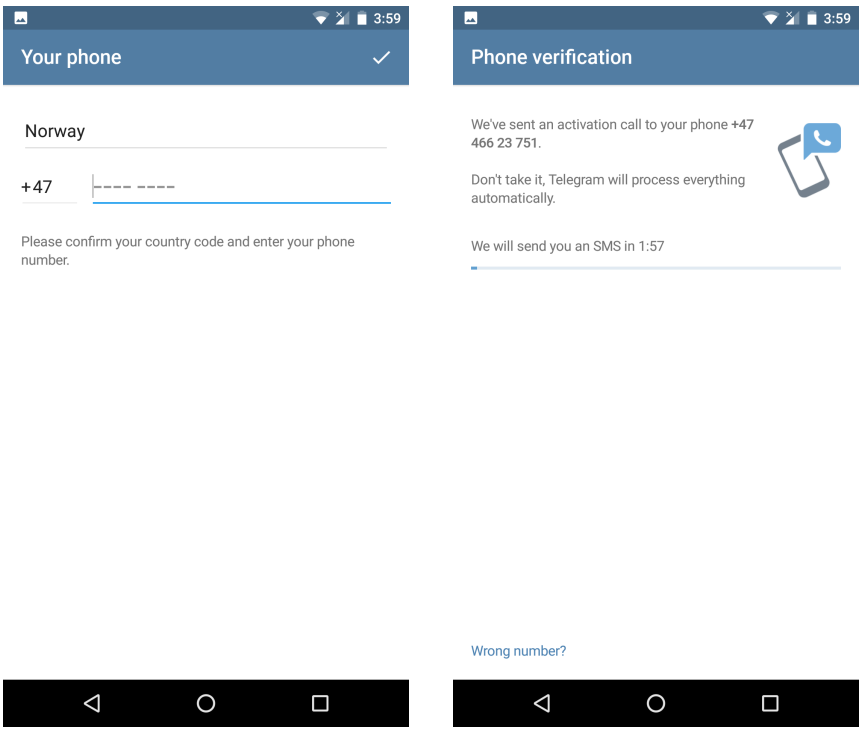


Figure 29: Riot: Other security implementations

the figure is the option to enable encryption in that specific room. Once encryption is enabled, it cannot be disabled throughout the conversation.



(a) Phone number registration (b) Verification of phone number

Figure 30: Telegram: Registration process

3.3.6 Case 6: Telegram

Telegram is an instant messaging platform which was started in 2013 after the NSA scandal. It has been developed for smartphones, tablets and even computers.[36] Telegram allows one-to-one and group communications, and the possibility to send files to people in your contact list. The difference between Telegram and the other secure instant messaging applications is that it only offers opt-in secure messaging, while normal conversations are cloud chats that are not end-to-end encrypted. Their motivation is to offer seamless cloud chat synchronization between all connected devices.[37]

For secure chat Telegram implements their own version of a cryptographic protocol that they have named the MTProto Protocol.[38] The same protocol is also used for normal cloud chats to encrypt the communication between the server and the client.

For end-to-end encrypted chats it is not allowed to screenshot inside the secret chat conversation. Therefore, the images within the conversation are shot with an external camera.

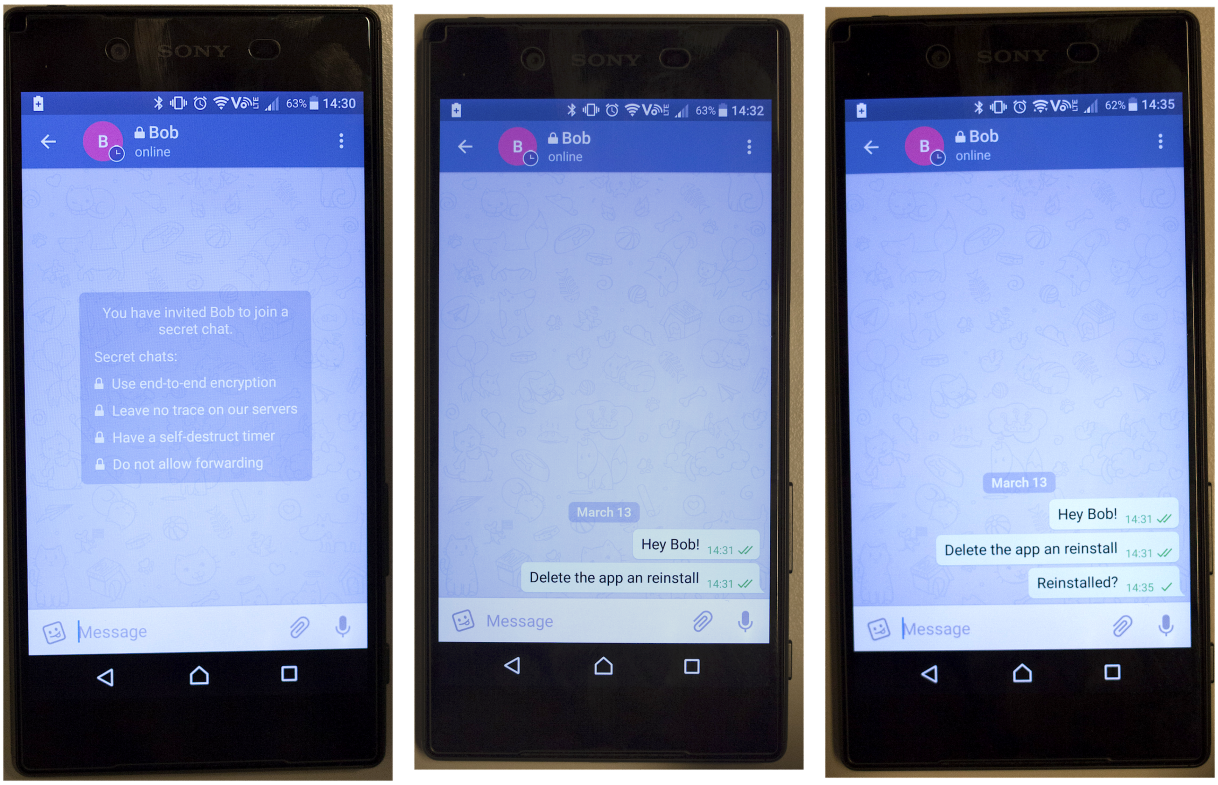
Initial Set Up:

The initial setup of the Telegram application and user registration is the same for the other applications. Figure 30a shows where the user inputs their phone number for the registration, and Figure 30b shows the activation process. Telegram sends an activation code through SMS, which can either be input manually or give Telegram access to do it automatically; and if the message is not get received in two minutes, a new SMS is sent. 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.

[36]Telegram FAQ `https://telegram.org/faq`

[37]Seamless chat cloud synd, tweet by Pavel Durov in 2015 at `https://twitter.com/durov/status/678305311921410048`

[38]MTProto Protocol, by Nikolai Durov in Telegram Documentation, available at `https://core.telegram.org/mtproto`



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

Figure 31: Telegram: Message after key change

Message after key change:

Telegram does not have end-to-end encryption enabled on by default, which means that Alice needs to start an explicit secret chat with Bob. The normal messages, which are called *cloud chats* on Telegram, do not have any encryption. Figure 31a shows the first view Alice sees 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 31b 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. Bob then reinstalls his application and Alice tries to send a new message to Bob, as shown in Figure 31c. Bob never receives the message, even after he finishes reinstalling 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, or any other information that could reveal that two users have ever had a secret chat. Thus, Telegram cannot check whether one of the users has reinstalled the application or not. It is up to the users to initiate new secret chats if the keys are lost or deleted securely by one of the participants.

Key change while a message is in transit:

As described in the last section about key changes, Telegram does not store any information about Alice or Bob having a secret chat; all is done by the client and nothing is sent to the cloud of Telegram. Therefore, there makes no sense to test this scenario, as it will have the same outcome as the one before.

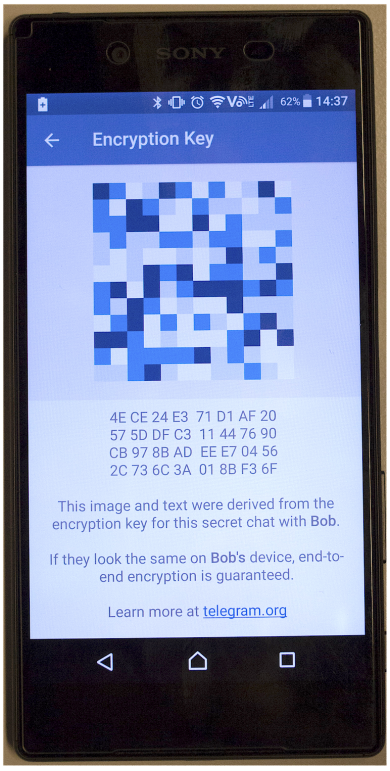


Figure 32: Telegram: Verification process

Verification Process Between Participants:

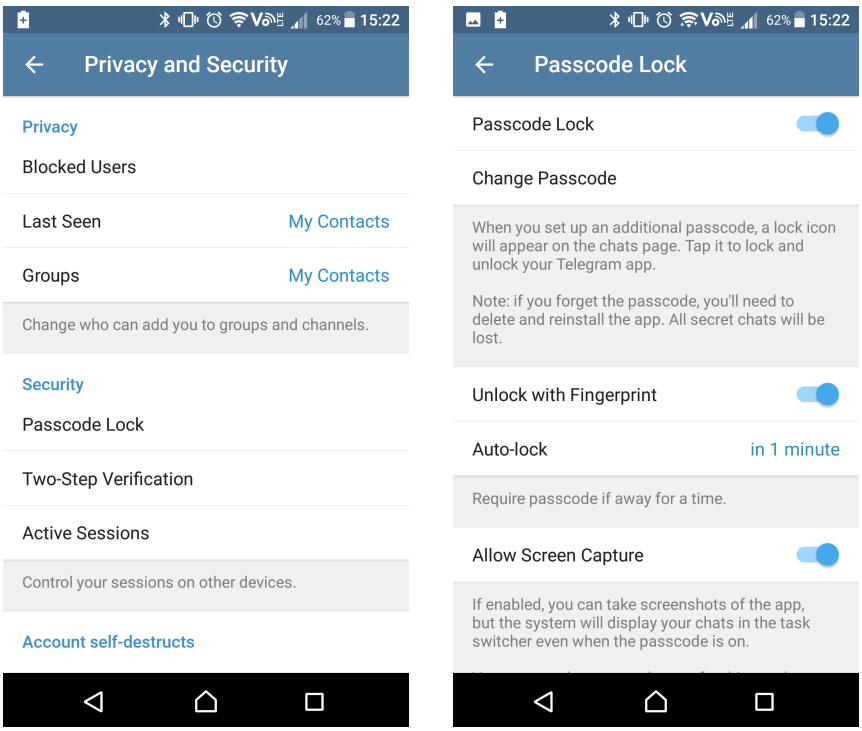
The verification process between Alice and Bob is rather difficult when using secret chats in Telegram. If Alice wants to verify Bob's encryption key she needs to open the specific secure chats settings page and click on the "Encryption Key" button, after which the verification page will be shown. Telegram does not support calling, only messaging, which makes it harder for Alice to verify Bob. Figure 32 shows the verification page, with an image derived from the encryption key, and the encryption key below. There is no way for Alice to arrive to the conclusion that it is the right image for the conversation.

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

Other security implementations:

Telegram supports a few other security features. Inside the settings page, there is one option to look at the "Privacy and Security" settings for the application. Figure 33a shows the specific settings that can be changed. Telegram supports two-step verification, i.e., 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 33b shows options under 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. 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



(a) Privacy and Security settings (b) Passcode settings

Figure 33: Telegram: Other security implementations

is not enabled, they do not have access to do so. However, for secure chats it is never allowed to screenshot.

4 Summary of Results

The results of tests are summarised in Table 3, listing what properties each application provides and which properties are missing.

From this overview one can conclude that all applications provide mostly the same properties related to the setup and registration phase. All applications except Riot support registration with the user's phone numbers, whereas Riot needs an e-mail address. Wire supports both phone number and e-mail address, which can also be used later to log in. Access to the SMS inbox is not mandatory in any of the applications, but it is set up as default to make it easier for the user, for otherwise one would need to enter the verification code manually. As Riot does not use a phone number for verification, it does not need access to SMS. Wire also does not have access to the SMS inbox because they believe that it is easy for the user to enter the verification code by hand.

Uploading the contacts list to a server is required by all applications because it enables to find if any of the contacts is already using the application. However, if any of the users do not want to upload the contacts list to the server, in order to remain anonymous, they could only give out the number to particular persons.

All the applications (except Riot that does not use a phone number) have the same properties when it comes to verification by SMS and phone call. They all first give the user the option to verify by reading the SMS and if the user never receives the SMS, then they can ask the application to call them.

Signal and WhatsApp have a Trust-On-First-Use (TOFU) method, where both trust the participants in a conversation without verifying first. The other applications need to verify each other first in order to be assured that the conversation is secure.

Table 3: Overview of the results from the analysis of secure messaging applications test scenarios.

Test Scenario and Properties	Application					
	Signal	WhatsApp	Wire	Viber	Riot	Telegram
Setup and Registration						
Phone Registration	✓	✓	✓	✓	✗	✓
E-mail Registration	✗	✗	✓	✗	✓	✗
Access SMS Inbox	✓	✓	✗	✓	✗	✓
Contact list Upload	✓	✓	✓	✓	✓	✓
Verification by SMS	✓	✓	✓	✓	✗	✓
Verification by Phone Call	✓	✓	✓	✓	✗	✓
Initial Contact						
Trust-On-First-Use	✓	✓	✗	✗	✗	✗
Notification About E2E Encryption	✗	✓	✗	✗	✓	✓
Message After a Key Change						
Notification about key changes	✓	✓	✗	✗	✓	✗
Blocking message	✓	✗	✗	✗	✗	✗
Key Change While a Message Is In Transit						
Re-encrypt and Send Message	✗	✓	✗	✗	✗	✗
Details About Transmission of Message	✓	✓	✓	✗	✓	✓
Verification Process						
QR-Code	✓	✓	✗	✗	✗	✓
Verify By Phone Call	✓	✓	✓	✓	✓	✗
Share Keys Through 3rd Party	✓	✓	✗	✗	✗	✗
Verified Check	✗	✗	✓	✓	✓	✗
Other Security Implementations						
Two-Step Verification	✗	✓	✗	✗	✗	✓
Passphrase/Code	✓	✗	✗	✗	✗	✓
Screen Security	✓	✗	✗	✗	✗	✓
Clear Trusted Contacts	✗	✗	✗	✓	✗	✗
Delete Devices From Account	✗	✗	✓	✓	✓	✓

✓: Has the property; ✗: Does not have the property.

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 having this only in the beginning would not bother the users. Only half of the applications have this notification implemented.

The differences in the way applications are handling key changes are 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. The blocking message functionality would not allow the sender to send a message before they verify the new cryptographic keys of the receiver if the receiver has generated new cryptographic keys during the conversation. Applications that do not give any notification or block sending of messages could be target for man-in-the-middle attacks, since one of the participants would never get the notification of key changes and thus could not detect any inconsistencies in the hijacked conversation. Wire and Viber are particularly vulnerable to this. The secret chats of Telegram do not work if cryptographic keys change because the application does not store any information about secret chats. The participants would need to re-start the conversation.

The only application that re-encrypts the messages and sends them again after the receiver got new cryptographic keys was WhatsApp. This is a useful usability property, which we hope it would be implemented by the rest of the applications as well. 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.

The "details about the transmission of a message" property questions whether the applications show to 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 is only tagged as "sent" for the sender, but if the message is re-encrypted and sent correctly, then the message details should also indicate "seen" by the receiver. The only application that does not show any information about whether past messages are sent or delivered is Viber.

Signal and WhatsApp have the easiest verification process by using a QR-code in conjunction with a built-in scanner. However, both have shortcomings since they do not have a check for revealing whether 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 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 that are shared between them, whereas the image encoding has no technical way of comparing it, besides by only looking at it.

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 potential intruders 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, in order 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 is not difficult to implement since they already know which contacts and their devices are verified.

The "delete devices from account" is only interesting for those applications that support 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.

5 Discussions and Recommendations

Instead of focusing on one test scenario at a time, like in the previous section, we discuss and evaluate here each application as a whole, and synthesise, based on the knowledge gained from the test scenarios, recommendations and possible improvements for each application.

5.1 Signal

The Signal application did not show major weaknesses, but several potential improvements still exist. Signal showed good understanding and care for 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 sends a message after key changes, the message gets blocked by the application until they verify each other. This is a useful property, 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.

Recommendations for Improvement. We suggest the following possible improvements which would benefit the Signal application, but it is important to note that these improvements should go through testing before deploying them in production. We suggest to use modelling and verification techniques to ensure an improvement does not break other security properties, but also, since these are usually related to the human interface to the app and the human behaviour we recommend taking a Behavioural Computer Science approach [28] to modelling both the human and app functionality as a whole, and thus consider the whole security ceremony [17, 44, 27, 43].

Re-encrypt and send lost messages: Give the user the option to re-encrypt a lost message and resend it after a re-verification is finished. The sending user always knows when a message is sent, delivered, or read, because of the checkmarks on each message within a conversation. This improvement would benefit both the sender and receiver because messages would not get lost during a conversation.

Notification about key changes: Move the notification message, i.e., change the behaviour from Figure 6c, to before someone sends a message, right after one of the users generates new keys. In this way the sender would know before sending a message that keys are changed and probably would like to know if it is still the same receiver before continuing to send 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. 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 trust 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 loosing control of their accounts since hackers would need to know the second password after a verification in order to get access to the account.

5.2 WhatsApp

We did not find major weaknesses in WhatsApp, but still identified several potential improvements. WhatsApp takes great care to strengthen the security around the user's account and messages, and we have not identified any chance 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. The reason of the developers comes from the fact that initially WhatsApp did not have E2E encryption, and if the users suddenly, after end-to-end encryption was added to the application, got notifications about changes then user confusion may occur and lead to angry users or deletion of the application. The problem of not having notifications 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.

Recommendations for Improvement.

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-encrypts and sends lost messages, then the possibility of sending private messages to the wrong person appears 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 of messages before the users within a conversation verify each other. With this, the possibility of sending private messages to an impersonator drops significantly, because they would need first to verify each other then send the message.

Passphrase/code: Add an option to enable passphrases or codes before opening the application to strengthen the security of the user's account from unauthorized access. If an adversary manages to get access to the user's phone, then the passphrase/code when accessing WhatsApp could defend from adversaries trying to read WhatsApp messages.

5.3 Wire

The Wire application has useful security and usability properties, but we still found 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.

Recommendations for Improvement.

- **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.
- **Option for blocking messages:** The same motivation as in Subsection 5.2.
- **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 and that 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 for both users since then the conversation could continue without any hiccups due to lost messages which 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 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 loosing control of their accounts since hackers would need to know the second password after a verification to get access to the account.
- **Passphrase/code:** The same motivation as in Subsection 5.2.
- **Screen Security:** Add an option in the settings page to enable screen security to 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 enable 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, thus breaking the privacy of the other conversation participants too.

5.4 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 several questionable aspects of Viber which make us reluctant in recommending this application to people that 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, the sender cannot see if the message is sent or received if they send multiple messages before looking at the status of the message.

Recommendations for Improvement. Many of the observations below are usually easy to fix or implement, in our opinion, and would drastically improve the application.

- **Details about sent messages:** Add information to each message to show whether the message has been sent, delivered, or read, because at the time of this test there was no way of knowing if old messages are read by or delivered to the receiver, only the last message has the information. This could be confusing for the sender because they never receive information about the status.
- **Tell the user about verification:** The same motivation as in Subsection 5.3.
- **Option for blocking messages:** The same motivation as in Subsection 5.2.
- **Notification about key changes:** The same motivation as in Subsection 5.3.
- **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.
- **Verification options:** The same motivation as in Subsection 5.3.
- **Two-step verification:** The same motivation as in Subsection 5.1.
- **Passphrase/code:** The same motivation as in Subsection 5.2.
- **Screen Security:** The same motivation as in Subsection 5.3.

5.5 Riot

The Riot application is still in beta stage, which is probably the reason behind some of their functionality choices. There are some good usability and security properties, but there are also a few aspects that can be improved.

When cryptographic keys change during a conversation, 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. Moreover, end-to-end encryption is not on by default, but this will probably change when the application is out of beta stage. There is no easy way of verifying users, but Riot has stated they are working on changing the method.

Recommendations for Improvement.

- **Verification options:** The same motivation as in Subsection 5.3.
- **Two-step verification:** The same motivation as in Subsection 5.1.
- **Passphrase/code:** The same motivation as in Subsection 5.2.
- **Screen Security:** The same motivation as in Subsection 5.3.

5.6 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.

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:** The same motivation as in Subsection 5.1.
- **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.

6 Conclusion and Further work

The work reported here conducted two analyses about secure messaging protocols and the applications which implement these types of protocols. The first analysis (following the recent article by Unger et al. [49]) described old and new secure messaging protocols that offer end-to-end encryption and identified types of security and privacy properties they provide, in Section 2. This review of protocols is an important prerequisite for understanding our analysis of the applications implementing them, which is our main contribution presented in Section 3. 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. 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 conducted analysis was the research experiment of applications which support these secure messaging protocols for their conversations, i.e., in Section 3. The applications offer useful usability together with security, and we would strongly recommend general public to adopt one of them (i.e., the one most suited for their needs, after reading through our analysis). We believe that even for common (non-technology people) would be very easy to adopt one of these applications; i.e., it would have the same difficulties (if one could call the such) as any other chat application, whereas the ones related to encryption would not add many difficulties. However, there are multiple applications which still could benefit from improvements. We

have provided recommendations, in Section 5, 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 prioritise if they think it would benefit their application and users.

The Signal protocol is designed to use a server for achieving the asynchronous messaging property. Even if messages are encrypted end-to-end there is still important metadata that is being manipulated and stored on the server. Therefore one would wish to secure better the server side of signal, especially when deployed in a cloud infrastructure or in a country with legislation that disregards privacy. Initial results in this direction have been presented in [47] using the recent technology of Intel SGX; however further work is needed in order to implement the ideas presented in [47]. Even more, such a secured implementation of the Signal server should also be verified to guarantee that the security properties claimed are uphold, and works that extend the TPM model from [10] should be extended towards the Intel SGX setting of [47].

Implementations of protocols from end-to-end secure instant messaging as the ones surveyed here could benefit from making use of a secure device attached to the smart phone like the OffPAD [37, 38] and possibly use a proxy that knows how to make use of such a device as in the OTDP architecture of [39].

References

[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. *Reuters*, (March 11), 2016. (Available at urlhttp://www.reuters.com/article/us-dataprotection-messaging-wire-idUSKCN0WC2GM).

[3] Alex Balducci and Jake Meredith. Olm cryptogrpahic review. Technical report, NCC Group PLC, 2016. (Available at `https://www.nccgroup.trust/us/our-research/matrix-olm-cryptographic-review/`).

[4] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology — CRYPTO '96: 16th Annual International Cryptology Conference*, pages 1–15, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[5] 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, 8 2007.

[6] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. Umac: Fast and secure message authentication. In *Annual International Cryptology Conference*, pages 216–233. Springer, 1999.

[7] John Black and Phillip Rogaway. Cbc macs for arbitrary-length messages: The three-key constructions. In *Annual International Cryptology Conference*, pages 197–215. Springer, 2000.

[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] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer, 2003.

[10] Sergiu Bursuc, Christian Johansen, and Shiwei Xu. Automated Verification of Dynamic Root of Trust Protocols. In Matteo Maffei and Mark Ryan, editors, *6th Int. Conf. on Principles of Security and Trust (POST 2017)*, volume 10204 of *LNCS*, pages 95–116. Springer, 2017.

[11] 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, 4 2017.

[12] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[13] 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.

[14] Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. Number 5246 in Request for Comments. RFC Editor, August 2008. (Available at `https://rfc-editor.org/rfc/rfc5246.txt`).

[15] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[16] Morris J. Dworkin. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. Technical Report SP 800-38A 2001 Edition, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2001.

[17] Carl M Ellison. Ceremony design and analysis. *IACR Cryptology ePrint Archive*, 2007:399, 2007.

[18] Justin Engler and Cara Marie. Secure messaging for normal people. Technical report, NCC Group, 2015. (Available at `https://www.nccgroup.trust/uk/our-research/secure-messaging-for-normal-people/`).

[19] Simson Garfinkel. *PGP: pretty good privacy*. O'Reilly Media, Inc., 1995.

[20] Ian Goldberg. Off-the-Record Messaging. Official documentation from Cypherpunks. (Available at `https://otr.cypherpunks.ca/`).

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

[22] 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.

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

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

[25] Matthew Hodgson. Encrypting Matrix: Building a universal end-to-end encrypted communication ecosystem with Matrix and Olm. In *Proceedings of the Free and Open Source Software Developers' European Meeting*, FOSDEM, 2017. (Available at `https://fosdem.org/2017/schedule/event/encrypting_matrix/`).

[26] Markus Jakobsson and Moti Yung. Proving without knowing: On oblivious, agnostic and blindfolded provers. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 186–200, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[27] Christian Johansen and Audun Jøsang. Probabilistic modelling of humans in security ceremonies. In Joaquin Garcia-Alfaro, Jordi Herrera-Joancomartí, Emil Lupu, Joachim Posegga, Alessandro Aldini, Fabio Martinelli, and Neeraj Suri, editors, *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, pages 277–292, Cham, 2015. Springer International Publishing.

[28] Christian Johansen, Tore Pedersen, and Audun Jøsang. Towards behavioural computer science. In Sheikh Mahbub Habib, Julita Vassileva, Sjouke Mauw, and Max Mühlhäuser, editors, *Trust Management X*, pages 154–163, Cham, 2016. Springer International Publishing.

[29] 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.

[30] 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.

[31] 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.

[32] Ingrid Lunden. Viber follows Messenger, launches Public Accounts for businesses and brands. *Tech Crunch*, (November 09), 2016. (Available at `https://techcrunch.com/2016/11/09/viber-follows-messenger-with-the-launch-of-public-accounts-for-businesses-and-brands/`).

[33] Moxie Marlinspike. Advanced Ratcheting. *Open Whisper Systems*, (November 26), 2013. (Available at `https://whispersystems.org/blog/advanced-ratcheting/`).

[34] Moxie Marlinspike. Simplifying OTR Deniability. *Open Whisper Systems*, (November 26), 2013. (Available at `https://whispersystems.org/blog/simplifying-otr-deniability/`).

[35] Moxie Marlinspike and Trevor Perrin. The Double Ratchet Algorithm. *Open Whisper Systems*, (November 20), 2016. (Available at `https://whispersystems.org/docs/specifications/doubleratchet/`).

[36] Moxie Marlinspike and Trevor Perrin. The X3DH Key Agreement Protocol. *Open Whisper Systems*, (November 04), 2016. (Available at `https://whispersystems.org/docs/specifications/x3dh/`).

[37] Denis Migdal, Christian Johansen, and Audun Jøsang. DEMO: OffPAD – Offline Personal Authenticating Device with Applications in Hospitals and e-Banking. In *23rd ACM Conference on Computer and Communication Security*, pages 1847–1849. ACM, 2016.

[38] Denis Migdal, Christian Johansen, and Audun Jøsang. Offpad: Offline personal authenticating device: implementations and applications. Technical report, University of Oslo, 2016.

[39] Denis Migdal, Christian Johansen, and Audun Jøsang. Offline trusted device and proxy architecture based on a new tls switching technique. In *International Workshop on Secure Internet of Things (SIOT)*. IEEE, 2017.

[40] Aulon Mujaj. A comparison of secure messaging protocols and implementations. Master's thesis, Department of Informatics at the Faculty of Mathematics and Natural Sciences of the University of Oslo, 2017.

[41] Matrix org. Megolm Group Ratchet. Technical report, Matrix, 2016. (Available at `https://matrix.org/docs/spec/megolm.html`).

[42] Sarah Perez. Skype Co-Founder Backs Wire, A New Communications App Launching Today On iOS, Android And Mac. *Tech Crunch*, (December 02), 2014. (Available at `https://techcrunch.com/2014/12/02/skype-co-founder-backs-wire-a-new-communications-app-launching-today-on-ios-android-and-mac/`).

[43] Cristian Prisacariu. Actor Network Procedures as Psi-calculi for Security Ceremonies. In Sjouke Mauw, Barbara Kordy, and Wolter Pieters, editors, *GraMSec 2014 – First International Workshop on Graphical Models for Security*, 2014. (Available on the arXiv: `https://arxiv.org/abs/1404.1988`).

[44] Kenneth Radke, Colin Boyd, Juan Gonzalez Nieto, and Margot Brereton. Ceremony analysis: Strengths and weaknesses. In Jan Camenisch, Simone Fischer-Hübner, Yuko Murayama, Armand Portmann, and Carlos Rieder, editors, *Future Challenges in Security and Privacy for Academia and Industry*, pages 104–115, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[45] 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.

[46] 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, 7 2016.

[47] Kristoffer Severinsen, Christian Johansen, and Sergiu Bursuc. Securing the End-points of the Signal Protocol using Intel SGX based Containers. In Ralf Küsters, editor, *5^{th} Workshop on Hot Issues in Security Principles and Trust (HotSpot 2017)*, pages 40–47. Stuttgart Univ. Technical Report, 2017. Available at `https://sec.uni-stuttgart.de/_media/events/hotspot2017/proceedings.pdf#page=40`.

[48] Ryan Stedman, Kayo Yoshida, and Ian Goldberg. A user study of off-the-record messaging. In *Proceedings of the 4th symposium on Usable privacy and security*, pages 95–104. ACM, 2008.

[49] 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, 5 2015.

[50] Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[51] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.