Hindawi
Security and Communication Networks
Volume 2021, Article ID 9965573, 30 pages
https://doi.org/10.1155/2021/9965573

Review Article

The Snowden Phone: A Comparative Survey of Secure Instant Messaging Mobile Applications

Christian Johansen,[1] Aulon Mujaj,[1] Hamed Arshad,[1] and Josef Noll[2]

[1]*Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway*
[2]*Department of Technology Systems, University of Oslo, Postboks 70, 2027 Kjeller, Norway*

Correspondence should be addressed to Hamed Arshad; hamedar@ifi.uio.no

Received 12 March 2021; Accepted 24 June 2021; Published 10 July 2021

Academic Editor: Shehzad Ashraf Chaudhry

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. New implementations of such end-to-end encrypted messaging protocols have appeared, and several popular chat applications have been updated to use such protocols. In this survey, we compare six existing applications for end-to-end encrypted instant messaging, namely, Signal, WhatsApp, Wire, Viber, Riot, and Telegram, most of them implementing one of the recent and popular protocols called Signal. We conduct five types of experiments on each of the six applications using the same hardware setup. During these experiments, we test 21 security and usability properties specially relevant for applications (not protocols). The results of our experiments demonstrate that the applications vary in terms of the usability and security properties they provide, and none of them are perfect. In consequence, we make 12 recommendations for improvement of either security, privacy, or usability, suitable for one or more of the tested applications.

1. Introduction

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 did 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 (https://en.wikipedia.org/wiki/Edward_Snowden#Global_surveillance_disclosures and two feature films on this topic are as follows: Oliver Stone's https://en.wikipedia.org/wiki/Snowden_(film) and Laura Poitras' https://en.wikipedia.org/wiki/Citizenfour), people finally understood that mass surveillance was an issue, and secure mobile messengers became more critical and popular. (Disclaimer: all the tests reported here were performed 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 for a new user on how to check which desired features are implemented by a specific application, even more so if the application is among the six surveyed here. More details for this paper can be found in the technical report [1] and the thesis of the second author [2].)

People are more prone to understand the privacy implications of mass surveillance [3]. 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 (https://en.wikipedia.org/wiki/List_of_government_mass_surveillance_projects). No

need to look further than the first quarter of 2017, when WikiLeaks (https://wikileaks.org/) 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 from the agency ("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 https://www.zerohedge.com/news/2017-03-07/wikileaks-hold-press-conference-vault-7-release-8am-eastern). 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 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 that does not leak any information about the user's message content. However, the problem with new applications is their adoption. After a while, companies such as Google, Facebook, and Open Whisper Systems joined forces to implement protocols into already widely adopted applications such as WhatsApp, which has over one billion monthly active users (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/).

Instant messaging clients that did not provide asynchronous communication became uninteresting because of the rise of smartphones and applications that were not always online. The most mature secure messaging protocol, Off-the-Record, did not support asynchronous messaging, which motivated the development of new protocols with asynchronous communication built-in. The most notable is the Signal application with their protocol also called Signal. After a while, the new protocol became quite popular among developers and researchers [4–8]. Subsequently, the Signal protocol started to be implemented in other applications, which were supporting only 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. In 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 enough information for the users 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 intuitive and usable enough to be handled by a majority of people, not only for the technology inclined ones.

1.1. The Goals of This Study. The area of end-to-end encryption in instant messaging applications has become rather broad recently. It is difficult for a user to find digestible information sources, and even less when it comes to comparative integrated studies. Therefore, our first goal is as follows:

G1: provide comprehensible and comparative study of relevant approaches to end-to-end encrypted messaging applications.

A detailed 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 technologies should be both usable so to allow a wide adoption but also have rather strong security requirements. These two, i.e., usability and security, are usually conflicting, and a good balance is difficult to find. This leads to our second goal.

G2: overview the security and privacy properties provided by current end-to-end messaging technologies and to what extent existing applications achieve these properties.

The Signal application (and protocol) is one of the most used end-to-end messaging technologies currently available for smartphones and desktop PCs. Moreover, the Signal protocol is employing state-of-the-art encryption and key establishment techniques.

G3: describe for nonexperts the security mechanisms behind the Signal protocol.

There is little research in the area of usability vs. security in secure messaging applications. Schroder et al. [9] were the first to look at the usability issues for end-to-end encrypted messengers, doing a user study of the usability of Signal's security features and proposing fixes to the issues they found with users failing to detect and deter man-in-the-middle attacks. In this paper, we look at the same types of potential attack as in [9], but we also look at the application interface and various interactions a user has with these applications. Moreover, we extend to five more applications than Signal (namely, we look also at WhatsApp, Wire, Viber, Riot, and Telegram) and check several new application usability properties (summarised in Table 1 from Section 4) for all the six applications under the test. We supplement the fixes proposed in [9] with 12 more recommendations for improvement and in Section 5, applicable to one or more of the tested applications.

Unger et al. [4] did a comprehensive study of secure messaging protocols, looking at security properties related to trust establishment, conversation security, and transport privacy. Since the audience of this paper may include also people without much technical or security skills, we make here an accessible summary of the findings from [4] about the protocols that are implemented by the applications that we study. In order to make this paper self-contained and easier to understand, we also provide an in-a-nut-shell description for each of the three major end-to-end encrypted messaging protocols, i.e., OTR, Signal, and Matrix.

1.2. Main Contributions.

(i) We make a comprehensive analysis of applications that implement secure messaging, by performing five testing scenarios to study their essential security and

Table 1: 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.

usability properties. We also provide suggestions for improvements.

(ii) We provide an (updated) overview of conversation security in secure messaging protocols, following [4]. Subsequently, we describe the inner security workings of the latest versions of two major protocols (OTR and Signal), 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 secure messaging protocols that are implemented by the applications analysed in this paper. First, we present the attacker models that we consider and assumptions that we make about the user applications, and then we review basic properties relevant for end-to-end encrypted messaging.

This section builds on the comprehensive survey [4], as well as on various other resources regarding these protocols. Most of the resources for the two new protocols Signal and Matrix are online, since these protocols have not come out of academia. However, both are built on the good foundation laid by the OTR protocol, which has been well studied in academia [10–16], and also complemented by significant online resources.

We relegate the more detailed information on the Off-the-Record and Signal protocols to Appendix. Appendix Section A surveys Off-the-Record (OTR) that is the baseline for the two other protocols, Signal (surveyed in Appendix Section B) and Matrix, which are new protocols that are actually implemented (or copied) by the current popular secure messaging applications.

2.1. Relevant Threat Models. We assume the following adversaries:

(i) Active adversaries: man-in-the-middle attacks are possible on both local 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 [17].

(ii) 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.

(iii) 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 messaging applications, e.g., an app on a smartphone, are secure and that the devices do not have malware that could exploit the messaging application.

2.2. Security Principles Relevant for End-to-End Encrypted Messaging. Different secure messaging protocols capture different security principles in various degrees and are important when comparing specifications of applications implementing them. Most security and privacy features that we review here are also found in [4].

End-to-end encryption: communication encryption protocols such as Transport Layer Security (TLS) [18] are designed to secure communications between a client and server. Messaging applications that allow two parties 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 read, store, or edit the message before encrypting again and sending it to the other user. Often servers cannot be trusted, as they can be hacked by an adversary, or may be contacted by law enforcement to give information sent by clients through the server [19]. End-to-end encryption ensures that the endpoints do the encryption while the servers only transmit the messages without network attackers nor a corrupted server being able to see the content.

Confidentiality: confidentiality ensures that the necessary level of secrecy is enforced at each junction of data processing, preventing unauthorized disclosure [20]. Confidentiality can be provided by encrypting data while it is stored and transmitted. In cryptographic protocols, confidentiality is essential to ensure that keys and other data are available only as intended [21]. Attackers try to break confidentiality by stealing password files, breaking encryption schemes, etc. 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 [20].

Integrity: integrity ensures that no one throughout the transmission modifies the messages. Hardware, software, and communication mechanisms must work in concert to maintain, process, and move data to intended destinations, without unexpected alterations. Systems that enforce and provide this security property ensure that attackers, or mistakes by users, do not compromise the integrity of systems or data [20]. 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 fields or nonces [21].

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 [22–27]. 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 [28]. 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 [29–31]. 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 [21, 32].

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 [4]. A protocol supports future secrecy when it can provide the "self-healing" aspect of the Diffie–Hellman ratchet, which is described in Appendix Section A, 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 were 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. 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 [34]. 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 other 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 require all participants to 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 e-mails, since there is a third party, like a server, 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.

Group chat properties: group conversations are popular nowadays, e.g., using Facebook Messenger (https://www.messenger.com), Slack (https://slack.com), or other popular messaging applications (https://www.engadget.com/2016/09/30/12-most-used-messaging-apps/). Security properties in the context of group chats include the following:

(1) Computational equality: whether the participants share an equal computational load when talking to each other.
(2) Trust equality: no single participant has more trust or responsibility, within the group, than any other.
(3) Subgroup messaging: participants can send messages to only a subgroup without generating a new conversation.
(4) Contractible membership: no 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 exchanged 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 memberships usually achieve these features without restarting the protocol.

Other security properties: a protocol or application for end-to-end secure IM may implement any (if not all) of the following:

(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 (such as the Tor (https://www.torproject.org/) network [35,36]) 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, it implies both speaker consistency and causality preserving are assured.

2.3. Usability and Adoption Principles for End-to-End Encrypted Instant Messaging. Various aspects need to be taken into account when looking at usability and adoption of a secure IM 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) Multidevice support: A user can connect to the conversation from multiple devices at the same time and has 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. Overview of Protocols for End-to-End Encrypted Instant Messaging. According to [4] and as shown in Table 2, none of the secure messaging protocols such as the Off-the-Record, Signal, and Matrix protocols can give the users every security property (for more information about the definition of the properties used in Table 2, please refer to [4]). In this section, we briefly comment on each of these, including for completeness also the Matrix (Matrix protocol having several IM implementations, at https://matrix.org) as a major protocol.

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 [14–16], 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; that is, 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 2.4.3 for details) support 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 2017 [6, 7] and since it is open source, the community can improve it. The Matrix protocol has also been audited [37]. This indicates that researchers are taking these protocols seriously and want to strengthen their credibility.

2.4.1. Off-the-Record. OTR uses an encrypt-then-MAC approach to protect messages (see Appendix Section A 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 [38]. 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 is 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 [4]. 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 [4]. 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 on the client side.

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 uses malleable encryption and the MAC keys are published after each message exchange [10]. OTR only signs the ephemeral keys and not every parameter during the key

Table 2: Comparison of secure messaging protocols (reproduced from [4]).

Properties	Protocol/client		
	OTR Pidgin	Signal Signal	Matrix 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	◷	●	●
Multidevice support	◷	⋈	●
No additional service	●	◷	◷
Group chat			
Computational equality	◷	●	●
Trust equality	◷	●	●
Subgroup messaging	◷	●	●
Contractible membership	◷	●	●
Expandable membership	◷	●	●

●: provides the property; ⋈: partially provides the property; ◷: does not provide the property.

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. However, the synchronous only requirement allows OTR to not rely on additional services for establishing a connection between two participants.

2.4.2. Signal. The design of the Signal protocol extends OTR, thus maintaining the same security features, while in some cases adding stronger or new features as well. Forward secrecy is provided because of the 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 (Appendix Section B.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 [4]. 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.

Group conversation is achieved by using multicast encryption, in 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 [4], but it cannot guarantee participant consistency. Multidevice is partly provided, in the sense that only an extra computer can join in a conversation by using the Signal Desktop application (https://whispersystems.org/blog/signal-desktop/), which is only a Chrome Extension (https://en.wikipedia.org/wiki/Browser_extension) and not an own application.

The Signal protocol provides computational and trust equality, subgroup messaging, and contractible and expandable membership properties. By using pairwise group messaging and multicast encryption, Signal has the ability to push group management into the client apps, 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.

2.4.3. Matrix. The Matrix protocol consists of two different algorithms, the Olm (https://matrix.org/docs/spec/olm.html) for one-to-one conversations and Megolm (https://matrix.org/docs/spec/megolm.html) 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. Multiple devices are possible with Matrix because Megolm implements a separate ratchet per sending device that is participating in a group conversation (Matrix.org Launches Cross-platform Beta of End-to-End Encryption Following Security Assessment by NCC Group https://pr.blonde20.com/matrix-e2e/). The protocol does not restart when the ratchet is replaced with a new one, which provides computational and trust equality, subgroup messaging, and contractible and expandable membership properties.

The NCC Group has audited both algorithms [37] 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 (https://about.riot.im/); ensures that the Megolm keys get refreshed after a certain amount of messages. 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. The Matrix developers have stated that this is intentionally designed [39] but also said that it is up to the application to offer the user the option to discard old conversations.

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. This is in contrast to the previous works [4] which looked at the protocols that are underlying some of the applications that we are evaluating. Here, we introduce and test more properties than in [9] that are specific for applications.

3.1. Test Scenarios. We first explain in this section 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. In each test scenario, we are going to study a set of properties regarding the security and usability of each of the

six applications. The results from testing the 21 properties described here will be summarised in Section 4 and Table 1.

3.1.1. Initial Setup. This test scenario includes two stages: "setup and registration" and "initial contact." Stage one is the first process a user needs to go through after installing an application. Here, we test 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. The properties of interest are as follows:

(i) Phone registration: user can register an account with a phone number.

(ii) E-mail registration: user can register an account with an e-mail address.

(iii) Verification by SMS: receive verification code through SMS.

(iv) Verification by phone call: receive verification code through a phone call.

(v) Access SMS inbox: the app requires access to the SMS inbox in order to read the verification code automatically.

(vi) Contact list upload: the app requires to upload contacts to a server in order to see if others are using the same application.

Stage two examines how applications handle the first message sent from one participant to another, whether the participants are informed of the secure messaging capabilities or whether the app shows how the cryptographic keys are used. Properties are as follows:

(i) Trust-on-first-use: automatically verify each other on initiation.

(ii) Notification about E2E encryption: the app presents notifications to explain to the user that messages are encrypted.

3.1.2. 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. Properties of interest are as follows:

(i) Notification about key changes: notifying Alice that Bob has changed cryptographic keys.

(ii) Blocking message: blocking new messages from being sent until Alice and Bob verify each other.

3.1.3. Key Change While a Message Is in Transit. Cryptographic key changes while a message is in transit are 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. Properties of interest are as follows:

(i) Re-encrypt and send message: does the application re-encrypt the message and send it again after the receiver has generated new cryptographic keys or is the message lost forever?

(ii) Details about transmission of message: users can see the difference between sent and delivered messages.

3.1.4. 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 it:

(i) QR-code: verify each other through a QR-code.

(ii) Verify by phone call: call each other with E2E-encrypted phone call and read keys out loud.

(iii) Share keys through 3rd party: share the keys through other applications.

(iv) Verified check: users can check later if a specific user is already verified.

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

(i) Passphrase/code: possibility of a passphrase/code that only the user knows and enters to access the application.

(ii) Two-step verification: when registering after a reinstall or new device, then a second passphrase/code is needed which only the specific user knows.

(iii) Screen security: the user is not allowed to screenshot within the application.

(iv) Clear trusted contacts: can all verified contacts be cleared, which means the user needs to verify each contact again?

(v) Delete devices from account: if the application allows multiple devices, is there an option to delete devices which are not in use anymore.

3.2. Running the Different Test Cases. For our tests, we used two separate smartphones as described in Table 3. Both phones have their personal phone number; the Sony phone has the contact information of the Nexus phone named Bob, while the Nexus phone has the contact details of the Sony phone named Alice. The reason behind the contact details is

Table 3: 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	3 GB	2 GB

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 an identical version number.

3.2.1. 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 the fact that it is completely open source. This reassures people that it does what the developers claim, since anyone can audit the source code as well as the employed cryptographic protocols.

(1) Initial Setup. An account can be registered to one device at a time, which means that using the same number on a second device will automatically deactivate the first device, to strengthen the security and to keep the private cryptographic keys on one device only. Figure 1(a) shows the first view a user sees when opening the app for the first time. Twilio is used for handling the SMS verification process with the Signal server when registering an account. Contact information is transmitted to the server but is not stored.

Figure 1(b) 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. 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 Figure 1(b), after which the Signal application calls the user and gives out a verification number to be typed in manually.

(2) Message after a Key Change. This test scenario analyses what happens when cryptographic keys change, e.g., when a user in a conversation deletes and then reinstalls the Signal app. Figure 2(a) shows the first two messages that Alice sends to Bob. The double checkmark shown on each message indicates that it has been received and read by Bob. The lock indicates that the message is encrypted from one end to the other, and nobody in between can read it. Figure 2(b) shows when Alice sends Bob another message after he has deleted and reinstalled his Signal app. The application notifies Alice that the message has not been delivered with a red notification icon on the left. It also gives information that by pressing on the message, the user can get more details about the notification.

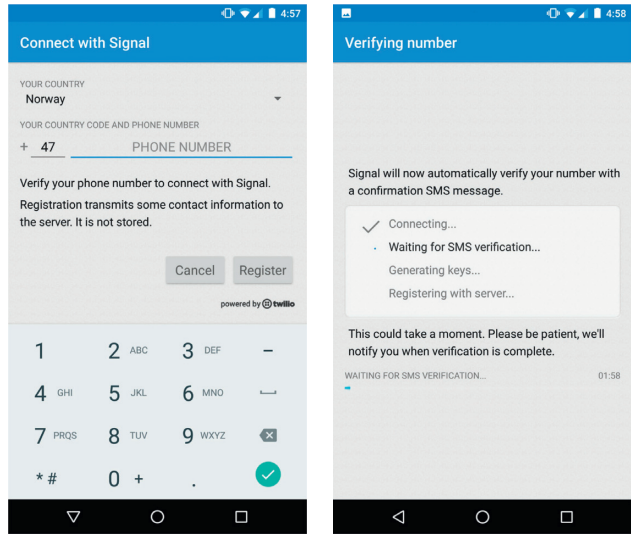
Figure 2(c) shows the view the user sees when pressing the message that was not delivered. Alice is presented with information that Bob has a new security number (cryptographic keys), and she needs to verify the new keys to get the ability to send any new message to Bob. After the verification process 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 2(d).

(3) 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 3(a) shows the initialization of the conversation. Figure 3(b) shows the conversation after a couple of messages from Alice to Bob. The second message is sent after Bob has deleted his application, and there is only a single checkmark on that message, which means the message has been sent, but not received. The icons on the third message 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, 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 3(b) is because the Signal app 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.

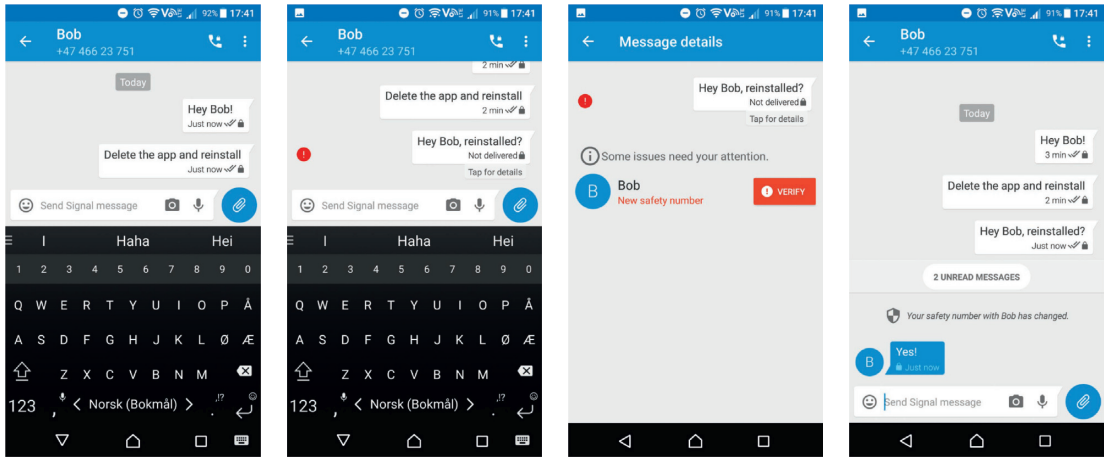
(4) Verification Process between Participants. Signal supports three different methods for the users to verify each other. The first verification process uses the built-in calling option of Signal which is end-to-end encrypted and then read out loud to the other participant the security numbers that are shown in Figure 3(c). If the Signal calling is not regarded as secure enough, users can meet in person and show the numbers to each other.

The second method, shown in Figure 3(c), uses the built-in QR-code scanner 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



(a) (b)

Figure 1: Signal: registration process. (a) Phone number registration. (b) Verifying the phone number.



(a) (b) (c) (d)

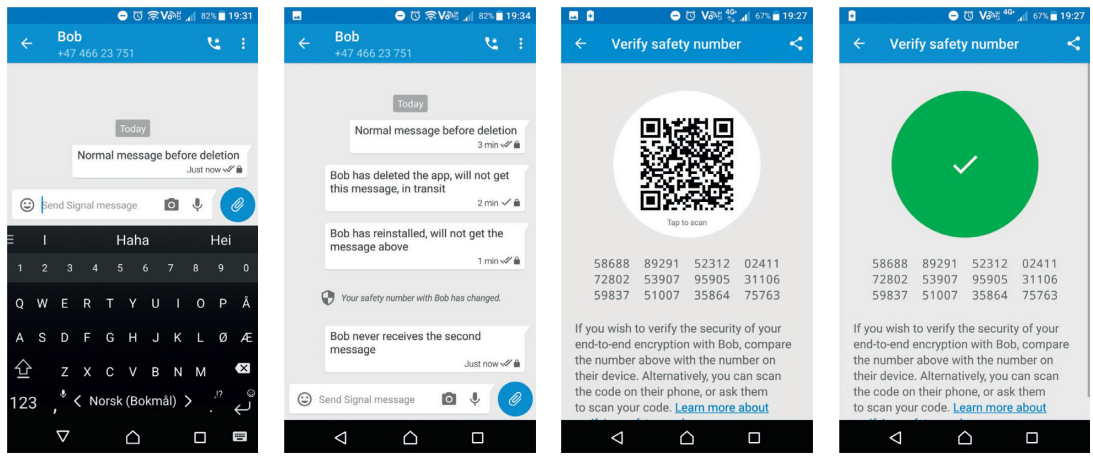
Figure 2: Signal: message after a key change. (a) Alice's first message. (b) Message after reinstall. (c) Verifying Bob again. (d) Message after verification.

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 (https://en.wikipedia.org/wiki/Pretty_Good_Privacy) [40,41] 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.

(5) Other Security Implementations. The Signal application has extra privacy settings. The first is the "Safety numbers approval" as shown in Figure 4(a). 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 receiver on 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 as they are inside the Signal application.

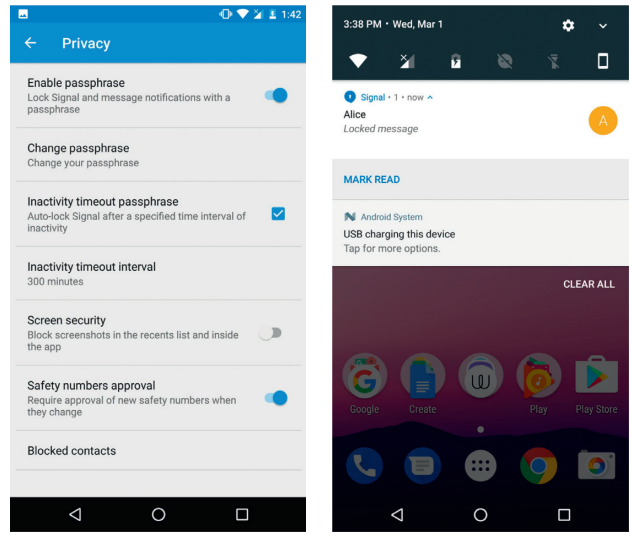
The last privacy setting is the ability to enable a passphrase. The passphrase locks the Signal application and all message notifications. It is possible to add an inactivity timeout passphrase which locks the application after some given time. Figure 4(b) shows the notification which is locked and when the user tries to open the application, she needs to enter the passphrase they chose when the setting was activated.

3.2.2. Case 2: WhatsApp. WhatsApp started as a small company in 2009, bought by Facebook in 2014 when it had 465 million monthly active users, and in 2017 that number



(a) (b) (c) (d)

Figure 3: Signal: key change while a message is in transit and verification process. (a) Message before key change. (b) Message after key change. (c) Verification page. (d) Verified.

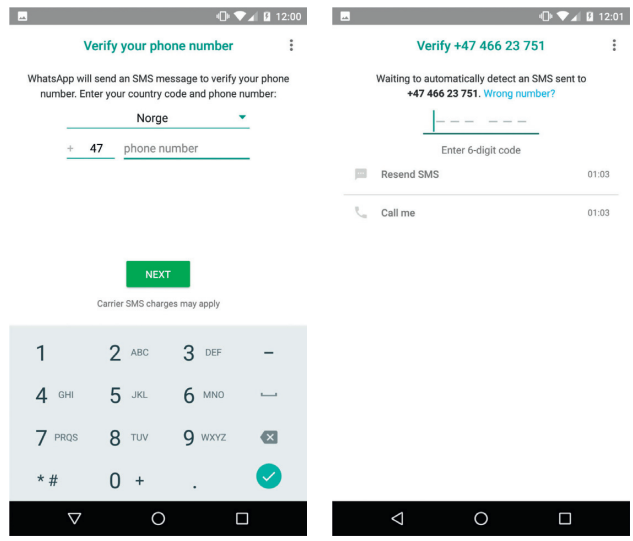


(a) (b)

Figure 4: Signal: other security implementations. (a) Privacy settings. (b) Notification locked.

has grown to 1.5 billion (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/). WhatsApp initially was only a cross-platform nonsecure 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 ("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/). This was an important step for the Signal protocol and Open Whisper Systems since now the most popular instant messaging application would use their protocol. In April 2016, a complete transition was made from nonsecure messaging to fully end-to-end encryption ("WhatsApp'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/).

(1) Initial Setup. 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 5(a) shows the first page a user sees when starting the application for the first time. WhatsApp uses its own infrastructure to handle the SMS verification process instead of a third party. Figure 5(b) 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



(a) (b)

Figure 5: WhatsApp: registration process. (a) Phone number registration. (b) Verifying the phone number.

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.

(2) Message after a Key Change. Figure 6(a) shows when Alice sends her first and second messages to Bob in order to initiate a conversation. The yellow notification box at the top of the conversation is shown to both participants stating that the conversation is end-to-end encrypted and more information can be obtained by pressing the box. Each message is shown with a double checkmark, as in Signal.

Figure 6(b) shows a new notification box appearing on Alice's conversation page after Bob has reinstalled his application. WhatsApp automatically checks if new cryptographic keys (security code) are changed even though she has not sent him any message. When Alice taps the notification box, a popup (Figure 6(c)) informs Alice why Bob's cryptographic keys have changed and the option to verify him before she sends new messages. Alice sends a new message to Bob after verifying new cryptographic keys of Bob and the message is labeled (Figure 6(d)) with a double checkmark meaning that everything went well.

(3) Key Change While a Message Is in Transit. This test scenario starts as the previous one, but here we look at how WhatsApp handles messages sent before Bob finishes to reinstall. Figure 7(a) shows Alice sending a second message to Bob after he has deleted his application. The single checkmark on the message means that it 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 7(b) displays the conversation after Alice sends a third message, showing that Bob receives the second message that was sent before he reinstalled. This means that WhatsApp re-encrypts messages when the receiver generates new cryptographic keys and the sender does not verify the new keys.

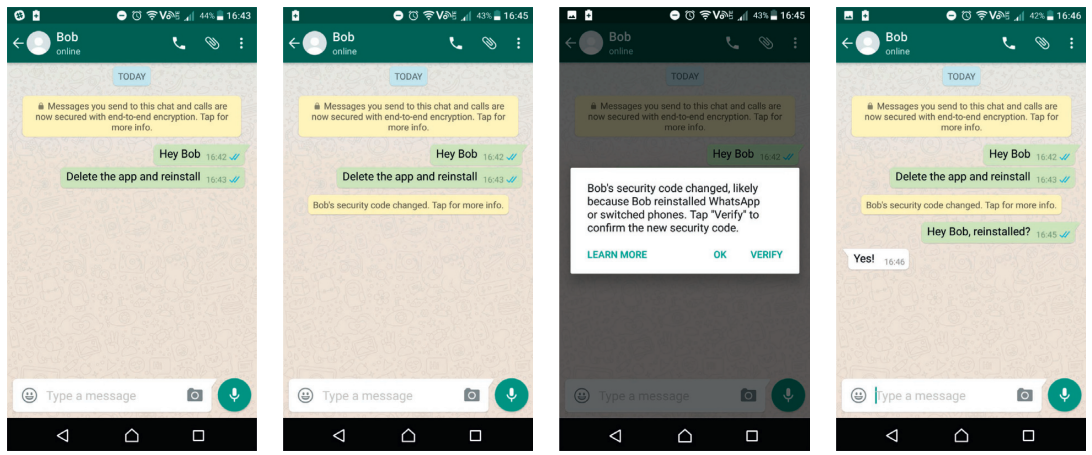
(4) Verification Process between Participants. WhatsApp has implemented the same verification process as Signal. It uses the Signal numerical format for verification, a QR-code for scanning with the built-in scanner, and the user can choose if they want to copy the security numbers outside of the WhatsApp 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 also features end-to-end encrypted calling, which enables users to call each other and verify the security code.

(5) Other Security Implementations. As shown in Figure 8(a) (setting page), there are options for changing the number of the account or even delete the account.

If Alice chooses the "Security" menu item and then activates the "Show security notification" option (shown in Figure 8(b)), then when Bob reinstalls the application or receives a new device the application shows a notification to Alice. Otherwise, if the option is turned off, then Alice does not receive any notification.

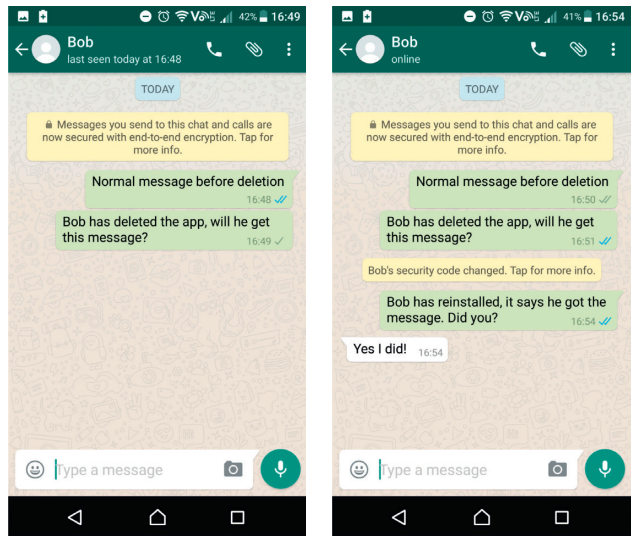
Figure 8(c) shows the two-step verification settings that WhatsApp has implemented, where the user needs to enter an additional passphrase when registering the account with the same number on a new device or after a fresh reinstall.

3.2.3. Case 3: Wire. Wire is an application that implements end-to-end encryption using the Proteus protocol, which is heavily based on the Signal protocol, but reimplemented in-house (Proteus Protocol, by Wire Swiss GmbH available at https://github.com/wireapp/proteus). Wire was started in 2012 by developers who previously worked at Microsoft and Skype, and finally, they released their own instant messaging



(a) (b) (c) (d)

FIGURE 6: WhatsApp: message after a key change. (a) Alice's first message. (b) After Bob has reinstalled. (c) Info about Bob's new keys. (d) Message after verification.



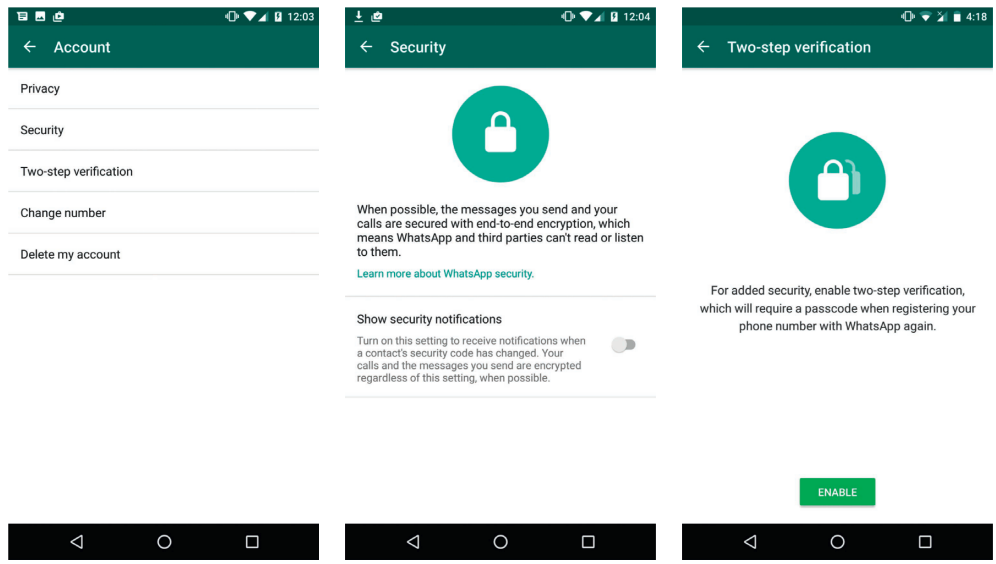
(a) (b)

FIGURE 7: WhatsApp: key change while message is in transit. (a) Bob deletes his app. (b) Bob reinstalled.

application in 2014 ("Skype Co-Founder Backs Wire, A New Communications App Launching Today On iOS, Android And Mac, by Sarah Perez in Tech Crunch on December 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/). 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 [42]. Wire offers the same features as the other applications, such as text, video, voice, photo, and music messages, is supported on multiple platforms, from smartphones to personal computers, and is also open sourced (https://github.com/wireapp).

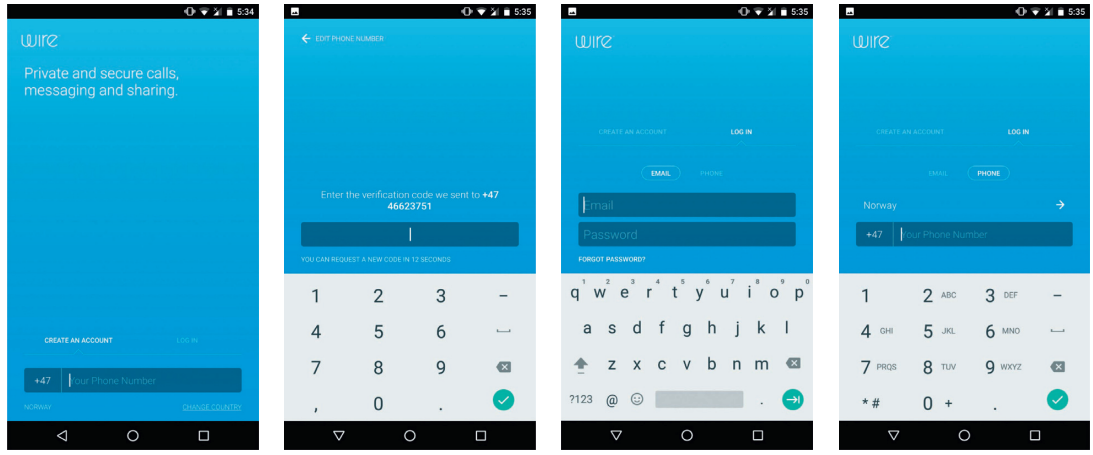
(1) Initial Setup. The Wire app has a different registration process than the other applications. The first page, as shown in Figure 9(a), asks to register a phone number. However, one can also create an account through the Wire web application by using just an e-mail address. Figure 9(b) shows the verification process, where the user needs to enter the verification code (which is received in an SMS) manually. If the user never receives the verification code, she can ask Wire to call the user to receive it. Wire application does not read the code received in the SMS automatically. Figures 9(c) and 9(d) demonstrate options to log in with an e-mail and/or a phone number, respectively. When a user reinstalls the application or changes her device, she does not need to go through the registration again.

(2) Message after a Key Change. Figure 10(a) shows Alice's initial contact with Bob. Wire uses a text under each message to explain if the message is delivered to the



(a) (b) (c)

Figure 8: WhatsApp: other security implementations. (a) Privacy settings. (b) Security notification. (c) Two-step verification.

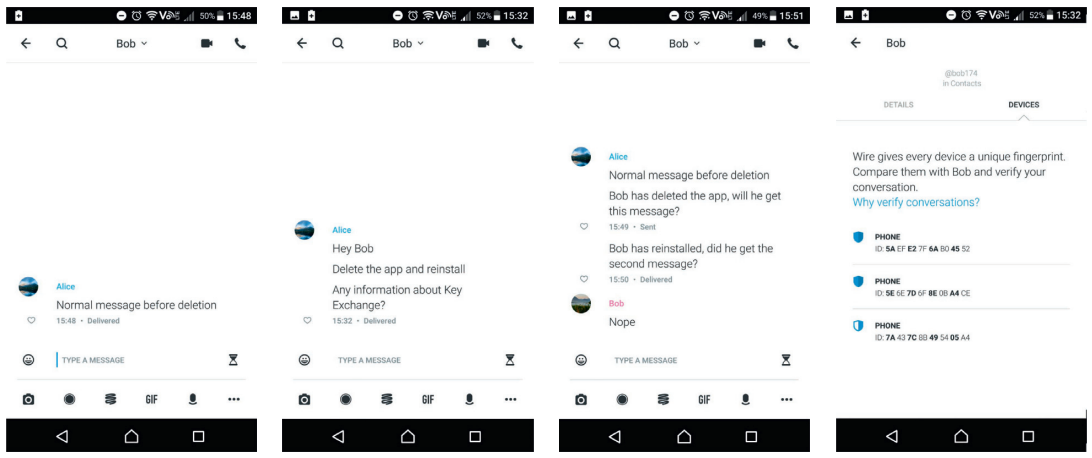


(a) (b) (c) (d)

Figure 9: Wire: registration process. (a) Phone number registration. (b). Phone verification. (c) User login with e-mail. (d) Login with phone number.

recipient or not. If the receiver reinstalls his application (which results in changing his cryptographic keys), then the sender (i.e., Alice) will not be notified by the Wire application about the new cryptographic keys of the receiver. As shown in Figure 10(b), Alice sends two messages to Bob, where Bob has reinstalled his application, but Alice does not get any notification by Wire that Bob has 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 Bob has got new cryptographic keys. Figure 10(d) shows Bob's device keys under his account, for three different devices, because Wire allows multiple devices to be associated to one account. This 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 only has a half shield because it has not been verified yet.

(3) Key Change While a Message Is in Transit. Figure 10(a) shows the initial message from Alice to Bob. If Alice sends a message when Bob has deleted the app, the message will be labeled by just "sent" and not delivered (Figure 10(c)). However, if Alice sends another message when Bob has reinstalled the app, then the message will be labeled as "delivered." This test demonstrates that Wire does not notify Alice about Bob's new keys and at the same time does not deliver messages encrypted with old cryptographic keys to devices with new keys.



(a) (b) (c) (d)

Figure 10: Wire: key change while a message is in transit and message after a key change. (a) Alice's first message before deletion. (b) After Bob has reinstalled. (c) No notification after Bob reinstalls. (d) Bob's device keys.

(4) Verification Process between Participants. Wire's verification process does not offer the same options for verifying each participant as the other applications. However, because Wire allows several devices to be associated with one account, and 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 10(d)). If Alice taps on one of the devices from the list, as shown in Figure 11(a), she can get some information about the phone's ID number and the public keys of that device. Alice can either call Bob over the phone or meet him in person and then verify the keys. When the verification is done, Alice needs to toggle the "not verified" switch to specify that this particular device is verified.

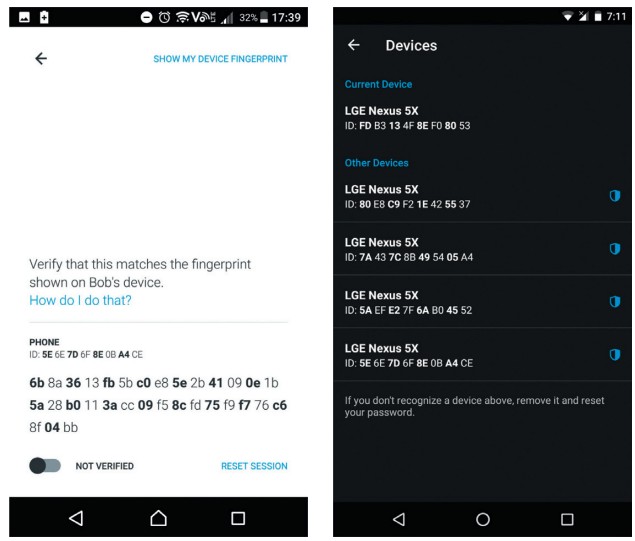
(5) Other Security Implementations. Wire does not have the extra security implementations that Signal or WhatsApp has. The few options include a way to change how the message conversation looks and the possibility to add an e-mail to the account for easier log in. The user can look at the devices which have been used with her account (as shown in Figure 11(b)), and if there are any devices which the user does not own or recognize, she can delete that specific device. After deleting a device, the user is prompted to change her password.

3.2.4. 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 [43]. 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 (Viber, by Rakuten Inc. https://www.viber.com/en/about). Viber works on smartphones and personal computers, which makes it cross-platform. Viber did not have end-to-end encryption in the beginning, but it is introduced in April 2016 for both one-to-one and group conversations ("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). Viber does not use the Signal protocol but implements its own protocol, which has the same concepts as the double ratchet protocol used by Signal (as stated by its developers).

(1) Initial Setup. The user registration process of Viber is the same as that in the previous applications. Figure 12(a) shows the user input for the user's phone number in the registration screen. Figure 12(b) 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.

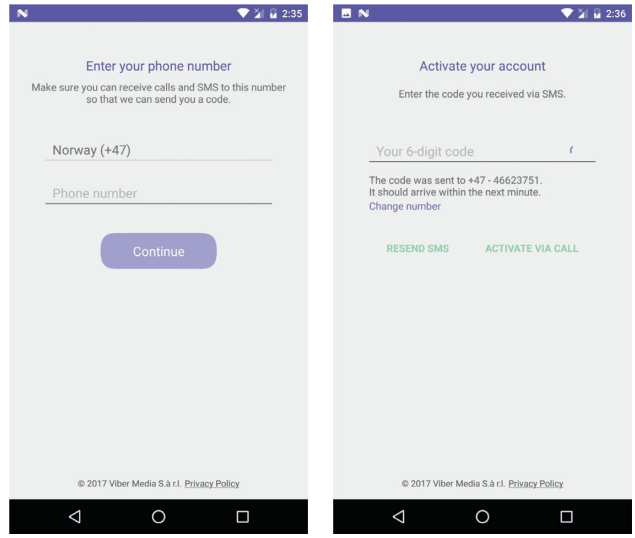
(2) Message after a Key Change. Viber does not notify the participants when the cryptographic keys change during a conversation. The first two messages in Figure 13(a) show Alice initiating the conversation with Bob. After Bob has reinstalled the application, Alice sends him another message. However, Figure 13(a) (third message) shows that Viber does not give any notification to Alice that Bob has generated new cryptographic keys. The only way for Alice to find out this is by checking the details of the conversation, by swiping from right to left. As shown in Figure 13(b), if the "Trust this contact" tab has changed to "Re-trust this contact," then Alice can infer that Bob has new cryptographic keys that should be verified.

(3) 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 explained before. Figure 13(c)shows that Alice sends a second message to Bob before he has reinstalled his application, and there is no information given to



(a) (b)

Figure 11: Wire: verification process. (a) Bob public keys for a device. (b) Other security features.



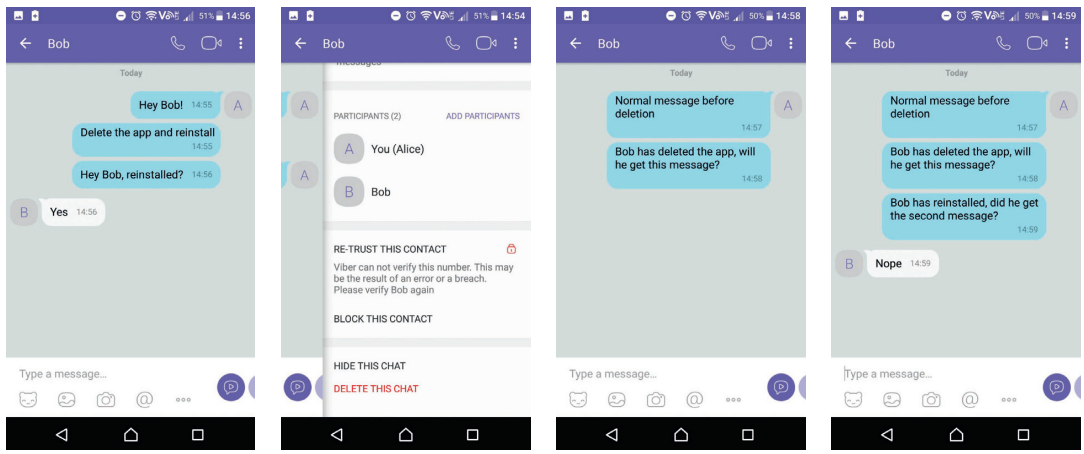
(a) (b)

Figure 12: Viber: registration process. (a) Registration with phone number. (b) Verification of phone number.

Alice if the message is sent or read by Bob. Figure 13(d) shows the third message from Alice to Bob after he has reinstalled his application. Alice never receives any notification from Viber that Bob has new cryptographic keys nor that he has not received the second message, and Viber does not re-encrypt and resend messages later on.

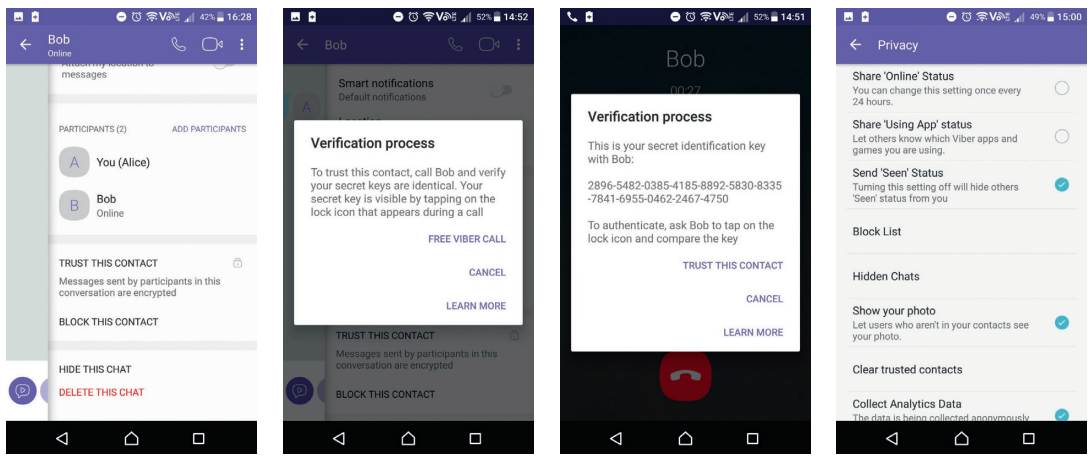
(4) Verification Process between Participants. The process of verifying a contact in Viber is quite straight forward. If Alice wants to verify Bob, she goes to one of their conversations, swipes (Figure 14(a)) to get the information tab, and then goes to the "Trust this contact" option. Figure 14(b) 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. Figure 14(c) 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.

(5) Other Security Implementations. Viber does not have extra security implementations. Figure 14(d) shows the privacy settings where the only security implementation is the "Clear trusted contacts" which clears all the contacts that Alice has verified throughout the time she had the account.



(a) (b) (c) (d)

FIGURE 13: Viber: message after a key change and key change while a message is in transit. (a) No notification about key changes. (b) Bob needs to be retrusted. (c) Messages after Bob has deleted. (d) Bob did not get the second message.



(a) (b) (c) (d)

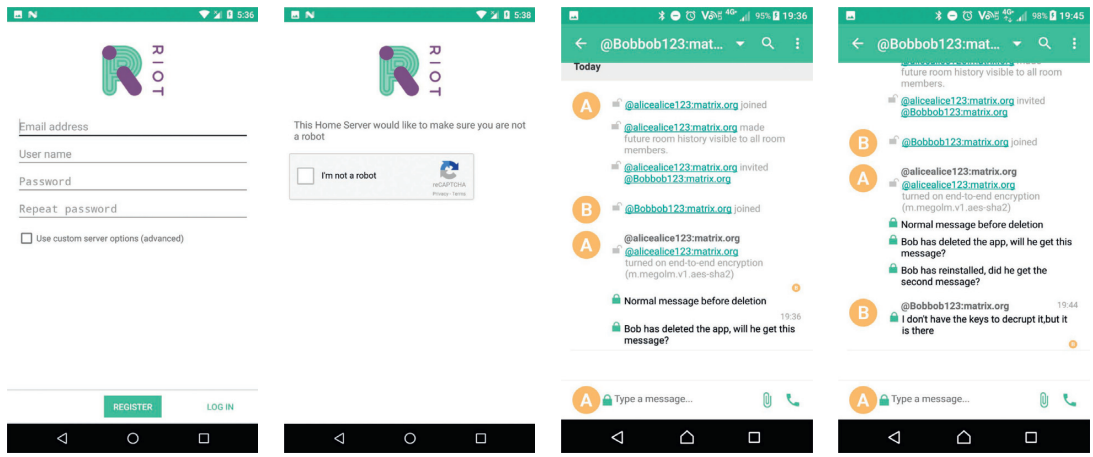
FIGURE 14: Viber: verification process and privacy settings. (a) Conversation info. (b) Verify a contact. (c) Alice verifying Bob. (d) Viber: privacy settings.

3.2.5. Case 5: Riot. Riot is a new chat client that is built on top of the Matrix (https://matrix.org/) protocol for its end-to-end encrypted capabilities. Matrix is an open standard for decentralized communications 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 (the same as Signal), but one does not need to rely on servers under the control of the Matrix team (unlike Signal). Riot and Matrix are open source, which means 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 outside their own network. Riot also provides group chat, voice (VoIP) and video calling, file transfer, and integration with other applications such as Slack (https://slack.com/) or IRC (https://www.wikiwand.com/en/Internet_Relay_Chat).

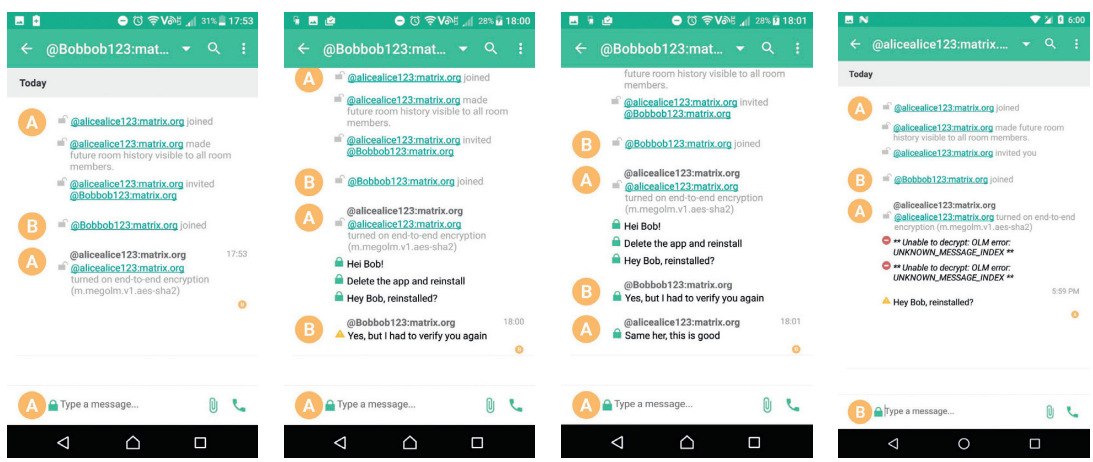
(1) Initial Setup. Riot is the only messaging client that does not rely on a phone number, but a user registers an account with an e-mail address (Figure 15(a)). When a user registers through the app, Riot sends a confirmation link to the entered e-mail address and the user has to click on that link, after which a Captcha verification will be requested for an extra layer of security as shown in Figure 15(b).

(2) Message after a Key Change. Riot is not the typical instant messaging application such as Signal or WhatsApp. Their vision is to make an application which works in 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. It should be noted that end-to-end encryption is still in beta form, and thus, it is not turned on by default. Figure 16(a) 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



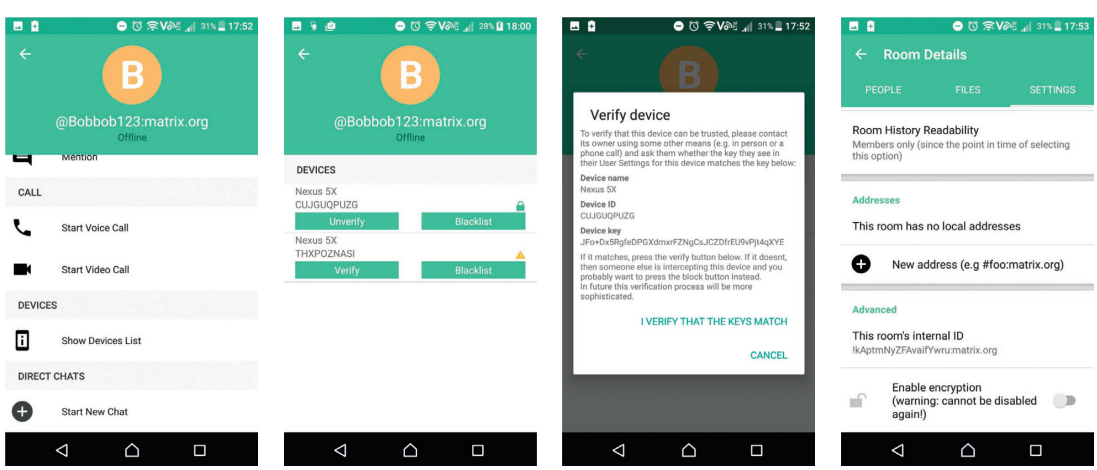
(a) (b) (c) (d)

Figure 15: Riot: registration process and key change while a message is in transit. (a) Registration by e-mail. (b) Captcha code. (c) Message before reinstall. (d) Bob's message after reinstall.



(a) (b) (c) (d)

Figure 16: Riot: message after a key change. (a) Initial conversation, Alice's view. (b) Message to Bob after he reinstalled. (c) New messages are verified. (d) Bob's view of previous message.



(a) (b) (c) (d)

Figure 17: Riot: verification process and other security features. (a) Profile details. (b) Verifying Bob the 2nd time. (c) Verifying Bob. (d) Other security features.

the encryption was toggled on. How the end-to-end encryption is toggled on is shown in the "other security implementations" part (Figure 17(d)). When Alice sends her initial message to Bob, the lock is changed to closed (Figure 15(c)) since E2E encryption is on. Figure 16(b) shows that if Bob reinstalls his application, he has to reverify Alice because his device keys are changed. Alice can also see that she has to reverify Bob as Bob's message has a yellow notification triangle. When Alice verifies Bob, the messages are then listed with a correct closed lock, which means that the messages are encrypted correctly (Figure 16(c)). In Riot, when a new user (or existing user with new keys) enters the chat room, she cannot read/access the previous messages (see Figure 16(d)).

(3) Key Change While a Message Is in Transit. Riot handles key changes in the same way as the previous section regardless of whether the message is in transit or not. When Bob deletes the application, Alice sends the second message to Bob (Figure 15(c)). Alice also sends a third message to Bob when Bob reinstalls the application. As shown in Figure 15(d), Bob can read just the third message but not the second. This shows that Riot handles the key changes in the same way as before, and Bob can see there were some messages sent but cannot decrypt since he lost the old keys.

(4) 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 by clicking on the "Device" tab, as shown in Figure 17(a). One of Bob's devices in Figure 17(b) has the yellow notification triangle, then that specific device has not been verified by Alice yet. She can either verify the device or put it into the blacklist, as in Figure 17(c), which means that specific device is no longer able to send any messages or invites to Alice.

If Alice decides to verify Bob's device, she clicks on the verify button and sees the verification popup from Figure 17(c). 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 the verification, Alice can either call Bob or meet him in person and then exchanges the device keys. Finally, Alice needs to press the "I verify that the keys match" button.

(5) 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 17(d) 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 the figure is the option to enable encryption in that specific room. Once encryption is enabled, it cannot be disabled throughout the conversation.

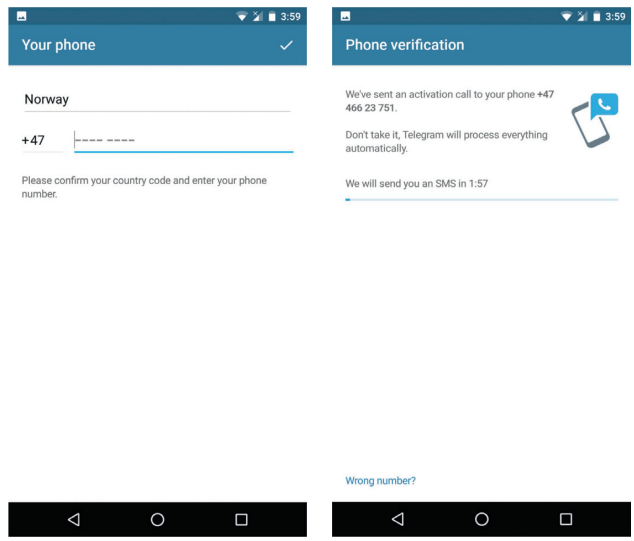
3.2.6. Case 6: Telegram. Telegram is an instant messaging platform which was started in 2013 after the NSA scandal. It has been developed for smartphones, tablets, and even computers (Telegram FAQ https://telegram.org/faq). Telegram allows one-to-one and group communications and the possibility to send files to people in the contact list. The difference between Telegram and the other secure IM 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 (Seamless chat cloud synd, tweet by Pavel Durov in 2015 at https://twitter.com/durov/status/678305311921410048).

For secure chatting, Telegram implements its own cryptographic protocol called the MTProto Protocol (MTProto Protocol, by Nikolai Durov in Telegram Documentation, available at https://core.telegram.org/mtproto). 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. Hence, in order to provide images for different test scenarios, we used an external camera.

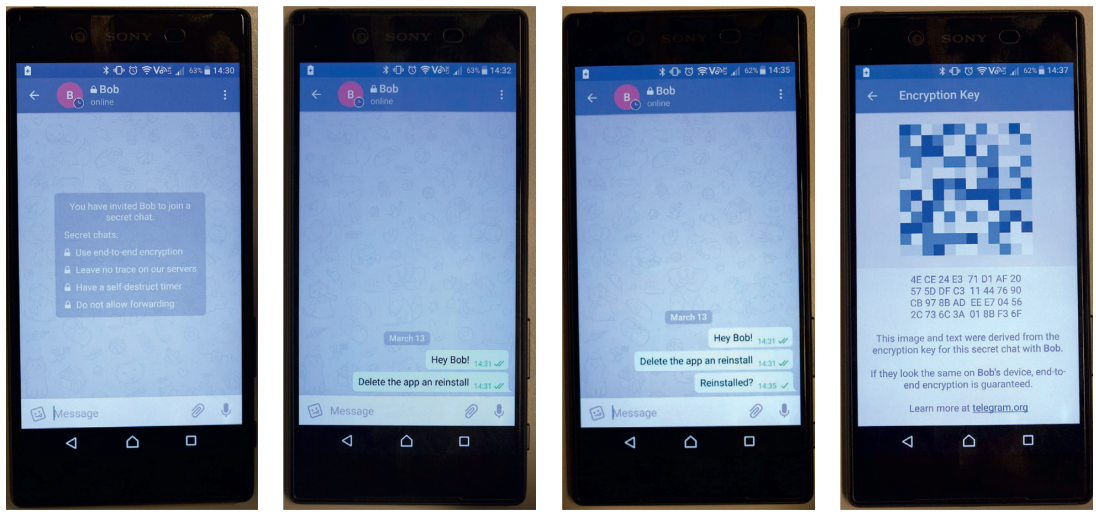
(1) Initial Setup. The initial setup of the Telegram application and user registration is the same for the other applications. Figure 18(a) shows that the user needs to enter her phone number for the registration process. As shown in Figure 18(b), Telegram sends an activation code through SMS, which can either be input manually or give Telegram access to take it automatically. If the verification message is not received in two minutes, a new SMS will be sent. The user also can ask Telegram to call her and activate it through phone call.

(2) Message after a Key Change. The end-to-end encryption is not enabled in Telegram by default. Normal messages, which are called cloud chats on Telegram, are not encrypted. Figure 19(a) shows the first view Alice sees when initiating a secret conversation with Bob, which says that the chat is end-to-end encrypted and the messages cannot be forwarded for security reasons. Figure 19(b) shows the first messages that have been sent from Alice to Bob, and the double checkmarks illustrate that Bob has received and read the message (a single checkmark means that the message has been sent). If Bob reinstalls his application and meanwhile Alice sends a message to Bob, then Bob will never receive that message (even after finishing the reinstallation of the application) as shown in Figure 19(c). Thus, Telegram does not use the previous device keys after reinstalling the application by one of the participants. Hence, Alice needs to start a new secret chat with Bob and send the previous undelivered messages again. Telegram does not store keys, or any other information that could reveal that two users have ever had a secret chat. Therefore, Telegram cannot check whether one of the users has reinstalled the application or not.



(a) (b)

Figure 18: Telegram: registration process. (a) Phone number registration. (b) Verification of phone number.



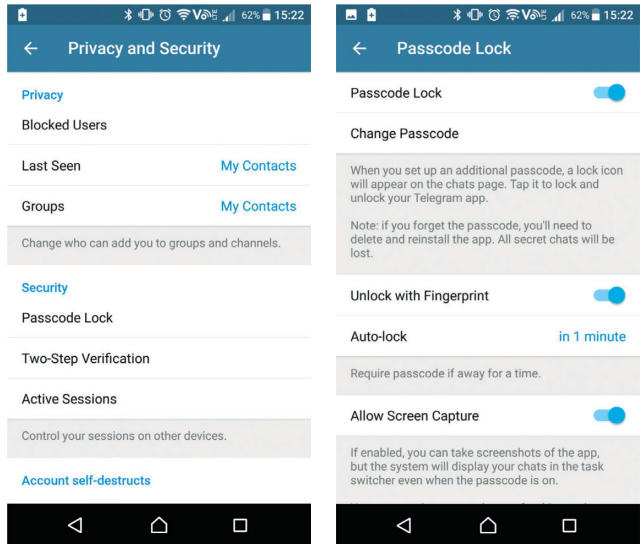
(a) (b) (c) (d)

Figure 19: Telegram: message after a key change. (a) Initial contact. (b) Alice: initial message. (c) Message after reinstall. (d) Telegram: verification process.

(3) Key Change While a Message Is in Transit. As described in the last scenario, 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.

(4) 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 keys, she needs to open the specific secure chats settings page and then click on the "Encryption Key" button. Telegram just supports messaging and does not support calling for the verification. Figure 19(d) shows the verification page, with an image, which is 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.

(5) 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 20(a)) 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) (b)

Figure 20: Telegram: other security implementations. (a) Privacy and Security settings. (b) Passcode settings.

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 20(b) shows options under the "Passcode Lock." This function locks the whole application with a passcode that 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 autolock, 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. However, for secure chats, it is never allowed to screenshot.

4. Summary of Results

The results of the tests are summarised in Table 1, listing what properties each application provides and which 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 for 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 a user, in order to remain anonymous, does not want to upload her contacts list, she needs instead to only give out her phone number to particular persons in order to communicate.

All the applications (except for 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 method, where users trust the other participants in a conversation without verifying first. The other applications ask users to verify each other first in order to be assured that the conversation is secure.

A notification at the start of the conversation would be useful to a new user who does not know what end-to-end encryption is, and 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 restart the conversation.

The only application that re-encrypts the messages and sends them again after the receiver gets 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, 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. 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.

Signal and Telegram both had a passphrase or code that the user had to enter in order to gain access to the application after some specific timeout expires. Both applications have also implemented screen security to not give potential intruders the ability to screenshot conversations. There is a setting to toggle the security off, but it is on by default in both Signal and Telegram.

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 even if it 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. Discussion and Recommendations

Instead of focusing on one test scenario at a time, like in the previous section, this section discusses and evaluates each application as a whole. Moreover, based on the knowledge gained from the test scenarios, some possible improvements for each application are provided, which have to be verified critically using modelling and verification techniques (besides standard software testing) in order to ensure that an improvement does not break other security properties.

5.1. Signal. The experiments conducted in Section 3.2.1 demonstrate that the Signal app does not have major weaknesses. However, there are several potential improvements that we discuss in the following. Signal showed good understanding and care for the user experience, with an easy verification process. The users can employ QR-codes for the verification purpose and/or can call each other and they can do the verification process through end-to-end encrypted phone calls.

When a party sends a message after changing the keys, the application blocks the message until the sender and receiver verify each other again. This is a useful property, but the application does not reveal the notification immediately when one of the participants in the conversation changes her keys (for example, due to the reinstallation of the application).

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.

The following provides recommendations for improvement that are applicable to the Signal app. We first state the feature in general terms and then explicit it for the specific app if needed:

(A) Re-encrypt and send lost messages: give the user an option to re-encrypt a lost message and resend it after finishing the reverification process, so that messages would not get lost during a conversation. In the Signal application, the sender of a message can know the status of her messages (i.e., sent or delivered/read) due to the existence of checkmarks on each message, making the implementation of this feature easy.

(B) Notification about key changes: it is recommended for an application to show a notification message immediately after each change of cryptographic keys. If the keys of a party in a conversation change, then the Signal application does not show any notification message to the participants immediately. It only gives the notification (that the keys are changed) when one party wants to send a message to another one (after changing the keys).

(C) Notification on end-to-end encryption: giving a notification at the beginning of each conversation stating that it is now end-to-end encrypted and the possibility to read more about it could help educate the end-users about what E2E encryption is and why they should care about it.

(D) Verified check: offer visual cues so the user can easily know that a party should be verified again to regain the trust properties. As demonstrated by our experiments, in the Signal application, there is no way to know if a user is already verified or not. However, if the application already keeps the list of verified contacts, then when one of the participants changes her cryptographic keys, it should be easy to implement this feature.

(E) Two-step verification: the security of an application would be improved by adding a "two-step verification" option, which, e.g., when a user wants to change her device or reinstall the application, she has to enter a previously chosen password, besides the code received in the SMS.

5.2. WhatsApp. The results of the experiments do not show major weaknesses in the WhatsApp application. However, the WhatsApp application can be improved in several ways as discussed below. WhatsApp takes great care to strengthen the security around the user's account and messages; however, it may suffer from impersonation attacks. Recommendations applicable to WhatsApp are as follows.

(F) Block messaging until verification: in order to prevent sending private messages to an impersonator, the participants should not be able to send any message before verifying each other. 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. Hence, an adversary may impersonate a legitimate contact and consequently WhatsApp re-encrypts and sends lost messages to the adversary who would thus receive private messages in an unauthorized way. Since this process is automatically controlled by the app, even the sender cannot stop re-encrypting and resending of lost messages after a key change. In order to overcome this weakness, the application must suspend the process of re-encrypting and resending of lost messages (after a key change) until the new cryptographic keys are verified.

(G) Locking the application using a passphrase/code: adding an option that requires a passphrase or code before opening the application would enhance the security of the user's account from unauthorized access. If an adversary somehow gets access to a user's phone, she would be unable to access the application messages as she does not know the passphrase/code.

WhatsApp recently has updated its privacy policy, which has caused concern among the users of this messaging application. According to WhatsApp's statements (https://faq.whatsapp.com/general/security-and-privacy/answering-your-questions-about-whatsapps-privacy-policy), their new Terms of Service and Privacy Policy is related to the managing of businesses on WhatsApp, which is an optional feature. It is claimed that all personal communications with friends, family, and so on are still protected by end-to-end encryption and neither WhatsApp nor Facebook can access them. It is also claimed that WhatsApp not only does not maintain logs of calls and messages but also it does not have access to the shared locations. Besides, groups will remain private and neither group data nor contact lists will be shared with Facebook or other apps offered by Facebook.

However, all communications with a WhatsApp business account may be used by Facebook to improve the marketing by means of displaying personalized advertisements on apps offered by Facebook, e.g., WhatsApp, Instagram, and Facebook. For example, Facebook may show an advertisement with a button to communicate with the related business through WhatsApp.

5.3. Wire. The Wire application features several useful security and usability properties. However, there are some properties which are not provided and might cause serious security problems. In this application, if a user uses a new device, Wire does not notify the other participants in a conversation. Thus, an impersonator may join the conversation and receive all the exchanged messages. Recommendations applicable to Wire are (A), (note that the Wire application uses a text under each message to show the status of the message (e.g., delivered or sent), which makes easy to implement this feature) (E), (F), and (G) from above, as well as the following new ones:

(H) Notify users regarding verification: when the application does not verify participants, it should notify users that they have to verify each other manually when initiating a new conversation, to prevent impersonation attacks.

(I) Notification about the verification of new devices: notify the other participants that a user added a new device and they have to verify the new device before sending any more messages (as the new device will also receive these messages). The Wire application allows a user to use the same account on multiple devices. However, if a user (in a conversation) adds a new device, the other participants (in that conversation) will not be notified about this change.

(J) More verification options: provide several different ways of performing the verification. For example, a QR-code or sharing keys with a third-party application are some possible options. Managing keys and authentication credential could greatly benefit from a secure device attached to the smart phone such as the OffPAD [44], in conjunction with a proxy that knows how to use such a device as in the OTDP architecture of [45]. Wire provides only calling a person every time, which may become cumbersome for users.

(K) Screen security: the privacy can be improved by providing a "screen security" option, which does not allow screenshots to be taken within the

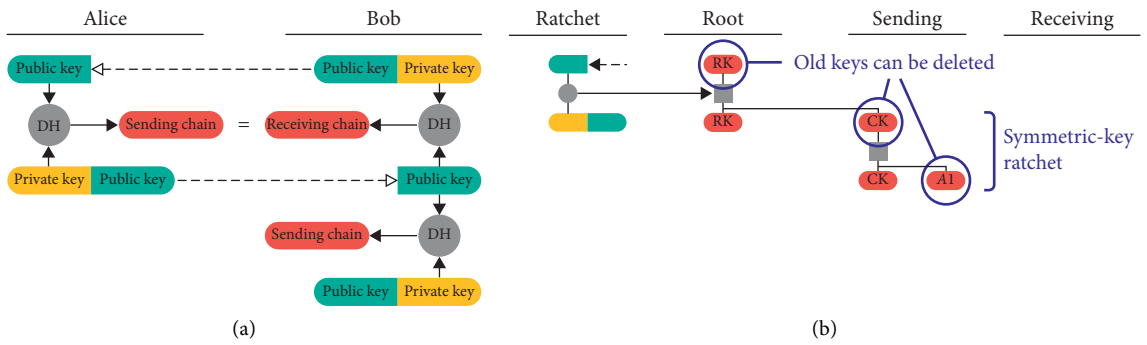


FIGURE 21: Signal Double Ratchet key derivation function (KDF) chains (reproduced from [53]). (a) Sending and receiving chains. (b) First double ratchet message key $A1$ for Alice.

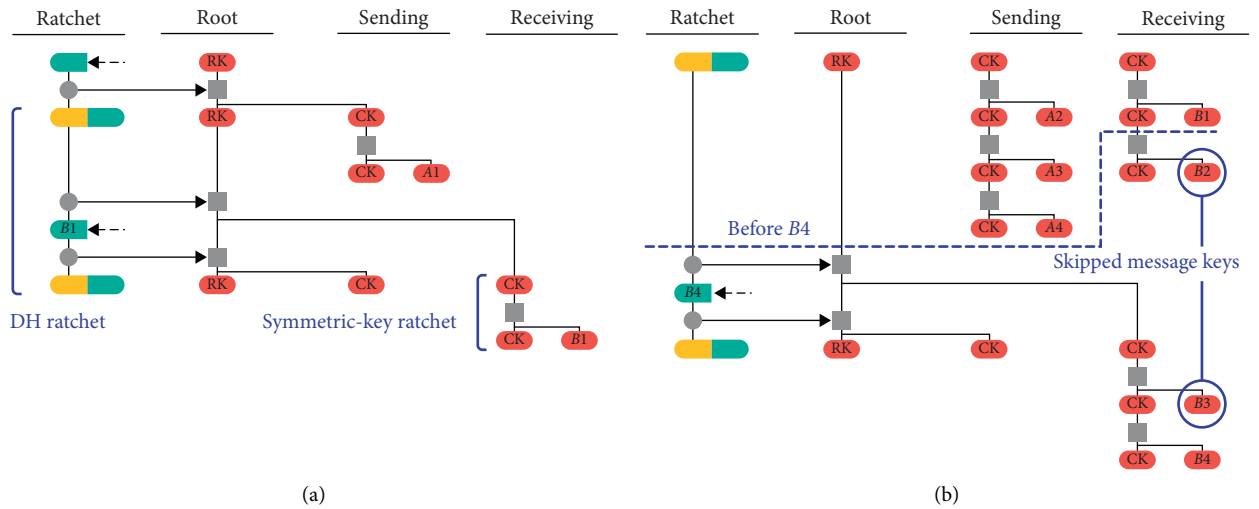


FIGURE 22: Signal more advanced aspects of the double ratchet protocol (reproduced from [53]). (a) First receiving double ratchet message key $B1$ computed by Alice. (b) Handling of out-of-order messages.

conversations. It is worth to mention that the screen security is useful if all participants in a conversation enable it because if a user does not enable this option, then she can take a snapshot and thus breaches the privacy of the other participants in the conversation.

5.4. Viber. The Viber application provides some good usability and security properties. However, there are several questionable aspects which make us reluctant in recommending this application. If the cryptographic keys of a user in a conversation change (e.g., because of reinstallation), Viber does not notify the other participants in the conversation about such changes. In addition, if a user sends several messages while the receiver is reinstalling Viber, then the sender cannot know whether the previous messages have been received or not. Many of the observations for Viber are easy to fix or implement, in our opinion, and would drastically improve the application. These include (A), (B), (E), (F), (G), (H), (J), and (K) from above, as well as the following new one:

(L) Labeling the status of messages: it is important for all messages in a conversation to have a label stating their status (e.g., sent, delivered, and read). In Viber, only the last message in a conversation is labeled with status information.

5.5. Riot. The Riot application is still in beta stage, and despite its usability and security properties, it can still be improved. When cryptographic keys change during a conversation, the previous messages are locked for the user (who is reinstalling the application). However, there is an option for the sender to send the locked messages (encrypted with new keys) again. In Riot, the users do not receive any notification message regarding the key changes (and the need to reverify each other). Moreover, Riot does not use end-to-end encryption by default. In addition, Riot does not provide an easy way for the verification of the users. Recommendations applicable to Riot are (E), (G), (J), and (K) from above.

5.6. Telegram. The telegram application has some useful security properties, but the usability features are rather

lacking and confusing for people who are not tech savvy. The biggest flaw in a secure messaging application is that the end-to-end encryption is not on by default. Hence, an explicit secret chat needs to be initiated every time users want to communicate. We think that this should become a norm these days for an application that advertises encrypted conversations.

In Telegram, if a user sends a message while the cryptographic keys have changed, the intended receiver does not get that message because secret chats are locked to one set of keys. Hence, if a user generates new keys, the participants need to start a new secret chat and cannot continue the previous secret chat. This is good for security, but the problem arises since Telegram does not notify users about key changes.

Telegram provides just one way for the verification of users, showing QR-codes in person, which is not good enough for a secure messaging application. We believe that Telegram can be improved by providing more options, such as calling or sharing keys through third-party applications. Recommendations applicable to Telegram are (B), (D), (J), and (K) from above.

6. Conclusion and Further Work

We have presented the testing and analyses that we have conducted on six secure messaging applications. As a prerequisite, in Section 2, we have started with giving a gentle introduction (following the recent article by Unger et al. [4]) to three secure messaging protocols that offer end-to-end encryption and the types of security and privacy properties they provide. This review of protocols is important 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 Signal protocol does not fully support multiple devices, while the Matrix protocol does this well. On the other hand, the Matrix protocol does not fully provide forward and future secrecy in the protocol and leaves it up to the implementation to support it. 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, 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 [8] using the recent technology of Intel SGX. This is also applicable to Matrix.

The contribution of this paper is the research experiment described in long Section 3 where we have performed five sets of tests on each of the six instant messaging applications that implement one of the two secure messaging protocols Signal or Matrix. We have thus tested 21 properties related to the usability and security of the applications Signal, WhatsApp, Wire, Viber, Riot, and Telegram. The results for the 21 properties on each of the six applications are summarized in Section 4 and listed in Table 1. In conclusion, these applications offer useful usability together with security, and we would strongly recommend to the 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 lay people without special technical skills, it would be rather easy to adopt one of these applications; that is, one would not encounter more difficulties (though this word is rather strong) than with any other chat application, whereas the features related to encryption would not add significant inconvenience. However, several of the tested applications could still benefit from improvements. We have provided 12 recommendations, in Section 5, each one is applicable to one or more of the six applications in order to harden their security while maintaining their useful usability properties.

Appendix

A. Off-the-Record in a Nut Shell

Intuitively, the Off-the-Record (OTR) protocol [10–16] 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 are however no similar laws for communications over the web. OTR-like protocols aim to provide this using cryptography techniques and thus need to provide at least perfect forward secrecy and deniability/repudiation. Full details can be found in [4], (Section 2.4).

Step 1: authenticated key exchange

The latest version of OTR [46] uses a variation of Diffie–Hellman key exchange called SIGMA [38]. 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, the channel does mutual authentication. The plain Diffie–Hellman key exchange is vulnerable to man-in-the-middle attacks which would break the authentication that OTR needs [12]. Therefore, OTR implements a signature-based authenticated DH exchange, named SIGMA, which solves this weakness [38].

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 the common DH secret and applied to the sender's identity [38]. OTR uses a four

message variant known as SIGMA-R, since it provides defence both against active attacks on the responder's identity and passive attacks on the initiator's identity.

Step 2: message transmission

The message transmission step, before sending, performs encryption and authentication of messages using AES [47] in counter mode [48] using message authentication codes (HMAC) [49] 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 [12]. 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, $\text{Enc}_{\text{EK}}(M)$and then MACs, the encrypted message, $\text{MAC}(\text{Enc}_{EK}(M), \text{MK})$. Bob computes the same EK and MK from the common secret, used to verify the MAC and decrypt the message.

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 [12]. 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, but published in the next step.

Step 4: Publish MK

The next step of OTR is to publish the old MAC keys by adding them to the next message that Alice or Bob sends to each other, in plaintext. Alice and Bob both know that they have moved over to MK'; hence, if one of them receives a message with the old MK, they will know that the message has been forged. Publishing MK allows others to forge transcripts of conversations between Alice and Bob. This is useful since it provides extra deniability to both parties [50]. In short, Alice's secrecy relies on Bob deleting the encryption keys, whereas Alice's deniability relies only on Alice publishing her MK.

A.1. 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) [51,52]. 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 $\text{ss}^A = \text{ss}^B$, i.e., the respective computed secrets, without revealing these secrets to anyone [13].

B. 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 become known as ratcheting and adopted in Signal as well [6]. The Signal protocol is designed by Moxie Marlinspike and Trevor Perrin from Open Whisper Systems (https://whispersystems.org/) to work in both synchronous and asynchronous messaging environments (advanced ratcheting, by Moxie Marlinspike at Open Whisper Systems, November 26, 2013, https://whispersystems.org/blog/advanced-ratcheting/). The goals of Signal include end-to-end encryption and advanced security properties such as forward secrecy and future secrecy [6]. Initially, Signal was divided into two different applications, TextSecure (https://whispersystems.org/blog/the-new-textsecure/) and RedPhone (https://whispersystems.org/blog/low-latency-switching/). The former was for SMS and instant messaging, while the latter was an encrypted VoIP (https://www.voip-info.org/wiki/view/What+is+VOIP) application. TextSecure was based on the OTR protocol, extending the ratcheting into a Double Ratchet, combining OTR's asymmetric ratchet with a symmetric ratchet [6], and 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 (https://whatsapp.com) by Facebook, the Messenger (https://messenger.com) also by Facebook, and Google's new messaging app, Allo (https://allo.google.com).

Signal uses the Double Ratchet algorithm [53] 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 [54] protocol (standing for extended triple Diffie–Hellman (https://whispersystems.org/docs/specifications/x3dh/)) which we describe later in Appendix Section B.3. Full details can be found in [4] (Chapter 3).

B.1. The Double Ratchet Algorithm. The Double Ratchet Algorithm uses key derivation function chains (KDF chains) [53] 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 [53]:

(i) 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.

(ii) Forward security: output keys from the past appear random to an adversary who learns the KDF key at some future point.

(iii) 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 Double Ratchet generates and maintains 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; see Figure 21(a)). 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, and 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 respective message. See more about out-of-order messages in Appendix Section B.2.

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 [53].

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 [53].

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 key has little use of it since this will soon be replaced with a new, unrelated key [53]. The DH ratchet produces as output the sending and the receiving chain keys, which have to correspond; that is, the sending chain of one party is the same as the receiving chain of the other party (see Figure 21(a)). Using a KDF chain here improves the resilience and break-in recovery.

Combining the symmetric-key ratchet and Diffie–Hellman ratchet is as follows: (a) 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 and (b) 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 21(b) 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 22(a) 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.

B.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) [53]. 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 22(b) 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 $B4$.

B.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 [54]. 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 [54]. 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, and (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. The server will delete a one-time prekey each time it sends it to another user.

(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 [54]) 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: (i) Alice's identity key IK_A; (ii) Alice's ephemeral key EK_A; (iii) identifiers stating which of Bob's prekeys Alice used; and (iv) an initial ciphertext encrypted with some AEAD encryption scheme [55] using AD as associated data and using an encryption key which is either SK or the output from some cryptographic pseudorandom 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 [54]. 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 [54]. 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.

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] J. Christian, M. Aulon, A. Hamed, and J. Noll, *Comparing Implementations of Secure Messaging Protocols (Long Version)*, Vol. 475, Department of Informatics, University of Oslo, Oslo, Norway, 2017.

[2] M. Aulon, "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 Oslo2017, Oslo, Norway, 2017.

[3] S. Bruce, *Data and Goliath: The Hidden Battles to Collect Your Data and Control Your World*, W.W. Norton & Company, New York, NY, USA, 2015.

[4] N. Unger, D. Sergej, B. Joseph et al., "SoK: secure messaging," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pp. 232–249, IEEE, San Jose, CA, USA, May 2015.

[5] F. Tilman, M. Christian, B. Christoph, B. Florian, S. Jörg, and T. Holz, "How secure is TextSecure?" in *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 457–472, IEEE, Saarbrucken, Germany, March 2016.

[6] K. Cohn-Gordon, C. Cas, D. Benjamin, G. Luke, and S. Douglas, "A formal security analysis of the signal messaging protocol," in *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, p. 466, Paris, France, April 2017.

[7] N. Kobeissi, K. Bhargavan, and B. Blanchet, "automated verification for secure messaging protocols and their implementations: a symbolic and computational approach," in *Proceedings of the European Symposium on Security and Privacy (EuroS&P)*, pp. 435–450, Paris, France, April 2017.

[8] S. Kristoffer, J. Christian, and B. Sergiu, "Securing the endpoints of the signal protocol using intel SGX based containers," in *Proceedings of the 5th Workshop on Hot Issues in Security Principles and Trust (HotSpot 2017)*, pp. 40–47, University of Stuttgart, Uppsala, Sweden, April 2017.

[9] S. Schröder, M. Huber, D. Wind, and R. Christoph, "When SIGNAL hits the fan: on the usability and security of state-of-the-art secure mobile messaging," in *Proceedings of the 1st European Workshop on Usable Security*, Darmstadt, Germany, July 2016.

[10] B. Nikita, I. Goldberg, and E. Brewer, "Off-the-record communication, or, why not to use PGP," in *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, pp. 77–84, ACM, Washington, DC, USA, October 2004.

[11] S. Ryan, K. Yoshida, and I. Goldberg, "A user study of off-the-record messaging," in *Proceedings of the 4th Symposium on Usable Privacy and security*, pp. 95–104, ACM, Pittsburgh, PE, USA, July 2008.

[12] D. R. Mario, G. Rosario, and K. Hugo, "Secure off-the-record messaging," in *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, pp. 81–89, ACM, Alexandria, VA, USA, November 2005.

[13] A. Chris and I. Goldberg, "Improved user authentication in off-the-record messaging," in *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, pp. 41–47, ACM, New York, NY, USA, October 2007.

[14] B. Jiang, R. Seker, and T. Umit, "Off-the-record instant messaging for group conversation," in *Proceedings of the IEEE International Conference on Information Reuse and Integration*, pp. 79–84, Las Vegas, NV, USA, August 2007.

[15] H. Liu, Y. Vasserman Eugene, and H. Nicholas, "Improved group off-the-record messaging," in *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, pp. 249–254, ACM, New York, NY, USA, November 2013.

[16] I. Goldberg, U. Berkant, D. Van Gundy Matthew, and H. Chen, "Multi-party off-the-record messaging," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pp. 358–368, ACM, Chicago, IL, USA, November 2009.

[17] D. Danny and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[18] D. Tim, "The transport layer security (TLS) protocol Version 1.2," 2008, https://rfc-editor.org/rfc/rfc5246.txt.

[19] E. Justin and M. Cara, *Secure Messaging for Normal People*, NCC Group, New Delhi, India, 2015, https://www.nccgroup.trust/uk/our-research/secure-messaging-for-normal-people/.

[20] S. Harris, *CISSP All-in-One Exam Guide*, McGraw-Hill Education, New York, NY, USA, 6th edition, 2012.

[21] C. Boyd and M. Anish, *Protocols for Authentication and Key Establishment*, Information Security and Cryptography-Springer, New York, NY, USA, 2003.

[22] C. S. Ashraf, Y. Khalid, Al-T Fadi, and Y. Ming-Hour, "A secure and reliable device access control scheme for IoT based sensor cloud systems," *IEEE Access*, vol. 8, pp. 139244–139254, 2020.

[23] C. S. Ashraf, F. Mohammad Sabzinejad, N. Kumar, and H. Alsharif Mohammed, "Pflua-DIoT: a pairing free lightweight and unlinkable user access control scheme for distributed IoT environments," *IEEE Systems Journal*, vol. 99, pp. 1–8, 2020.

[24] I. Azeem, C. S. Ashraf, A. Osama Ahmad, Y. Khalid, and N. Kumar, "A novel pairing-free lightweight authentication protocol for mobile cloud computing framework," *IEEE Systems Journal*, vol. 99, pp. 1–9, 2020.

[25] A. Hamed, N. Morteza, A. Sara, and N. Mahboubeh, "Design and FPGA implementation of an efficient security mechanism for mobile pay-TV systems," *International Journal of Communication Systems*, vol. 30, no. 15, Article ID e3305, 2017.

[26] J. Black, H. Shai, K. Hugo, K. Ted, and R. Phillip, "UMAC: fast and secure message authentication," in *Proceedings of the Annual International Cryptology Conference*, pp. 216–233, Springer, Santa Barbara, CA, USA, August 1999.

[27] J. Black and R. Phillip, "CBC MACs for arbitrary-length messages: the three-key constructions," in *Proceedings of the Annual International Cryptology Conference*, pp. 197–215, Springer, Barbara, CA, USA, August 2000.

[28] D. Gollmann, *Computer Security*, Wiley, Hoboken, NJ, USA, 2011.

[29] I. Azeem and A. Chaudhry Shehzad, "Comment on "ElGamal cryptosystem-based secure authentication system for cloud-based IoT applications"" *IET Networks*, vol. 1880, 2021.

[30] I. Azeem and A. Chaudhry Shehzad, "Comment on "SFVCC: chaotic map-based security framework for vehicular cloud computing"" *IET Intelligent Transport Systems*, vol. 14, no. 12, p. 1723, 2020.

[31] I. Azeem, K. Saru, L. Xiong, W. Fan, A. Chaudhry Shehzad, and A. Hamed, "An improved SIP authentication scheme based on server-oriented biometric verification," *Wireless Personal Communications*, vol. 97, no. 2, pp. 2145–2166, 2017.

[32] A. Chaudhry Shehzad, "Correcting "PALK: password-based anonymous lightweight key agreement framework for smart grid"" *International Journal of Electrical Power & Energy Systems*, vol. 125, Article ID 106529, 2021.

[33] M. Moxie, *Advanced Ratcheting*, Open Whisper Systems, San Francisco, CA, USA, 2013, https://whispersystems.org/blog/advanced-ratcheting/.

[34] I. Goldberg, "Off-the-record messaging," 2020, https://otr.cypherpunks.ca/.

[35] D. Roger, M. Nick, and S. Paul, *Tor: The Second-Generation Onion Router*, Naval Research Lab Washington DC, Washington, DC, USA, 2004.

[36] McC. Damon, K. Bauer, G. Dirk, T. Kohno, and S. Douglas, "Shining light in dark places: understanding the tor network," in *Privacy Enhancing Technologies*, pp. 63–76, Springer, Berlin, Germany, 2008.

[37] A. Balducci and J. Meredith, *Olm Cryptogrpahic Review*, NCC Group PLC, Manchester, UK, USA, 2016, https://www.nccgroup.trust/us/our-research/matrix-olm-cryptographic-review/.

[38] H. Krawczyk, "SIGMA: the "SIGn-and-MAc" approach to authenticated diffie-hellman and its use in the IKE protocols," in *Proceedings of the 23rd Annual International Cryptology Conference—CRYPTO*, pp. 400–425, Springer, Santa Barbara, CA, USA, August 2003.

[39] H. Matthew, "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*, Buenos Aires, Argentina, May 2017.

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

[41] S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly Media, Inc., Newton, MA, USA, 1995.

[42] A. Eric, *Go Ahead, Make Some Free, End-To-End Encrypted Video Calls on Wire*, Reuters, London, UK, 2016, http://www.reuters.com/article/us-dataprotection-messaging-wire-idUSKCN0WC2GM.

[43] L. Ingrid, "Viber follows messenger, launches public accounts for businesses and brands," 2016, https://techcrunch.com/2016/11/09/viber-follows-messenger-with-the-launch-of-public-accounts-for-businesses-and-brands/.

[44] M. Denis, J. Christian, and J. Audun, "DEMO: Off-PAD—offline personal authenticating device with applications in hospitals and e-banking," in *Proceedings of the 23rd Conference Computer and Communication Security*, pp. 1847–1849, ACM, Vienna, Austria, October 2016.

[45] M. Denis, J. Christian, and J. Audun, "Offline trusted device and proxy architecture based on a new TLS switching technique," in *Proceedings of the International Workshop on Secure Internet of Things (SIOT) IEEE*, Oslo, Norway, September 2017.

[46] I. Goldberg, D. Goulet, A. Jacob, and B. Jurre, *Off-the-Record Messaging Protocol Version 3*, University of Waterloo, Ontario, Canada, 2012, https://otr.cypherpunks.ca/Protocol-v3-4.0.0.html.

[47] J. Daemen and R. Vincent, *The Design of Rijndael: AES—The Advanced Encryption Standard*, Springer, Berlin, Germany, 2002.

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

[49] B. Mihir, C. Ran, and K. Hugo, "Keying hash functions for message authentication," in *Proceedings of the 16th Annual International Cryptology Conference—CRYPTO*, pp. 1–15, Springer, Santa Barbara, CA, USA, August 1996.

[50] M. Moxie, *Simplifying OTR Deniability*, Open Whisper Systems, San Francisco, CA, USA, 2013, https://whispersystems.org/blog/simplifying-otr-deniability/.

[51] J. Markus and Y. Moti, "Proving without knowing: on oblivious, agnostic and blindfolded provers," in *Proceedings of the 16th Annual International Cryptology Conference—CRYPTO*, pp. 186–200, Springer, Santa Barbara, CA, USA, August 1996.

[52] C. Yao Andrew, "Protocols for secure computations," in *Proceeings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*, pp. 160–164, IEEE, Chicago, IL, USA, November 1982.

[53] M. Moxie and T. Perrin, *The Double Ratchet Algorithm*, Open Whisper Systems, San Francisco, CA, USA, 2016, https://whispersystems.org/docs/specifications/doubleratchet/.

[54] M. Moxie and T. Perrin, *The X3DH Key Agreement Protocol*, Open Whisper Systems, San Francisco, CA, USA, 2016, https://whispersystems.org/docs/specifications/x3dh/.

[55] R. Phillip, "Authenticated-encryption with associated-data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 98–107, ACM, Washington, DC, USA, November 2002.