SoK: An Analysis of End-to-End Encryption and Authentication Ceremonies in Secure Messaging Systems

Mashari Alatawi
Texas A&M University
College Station, Texas, USA
mashari@tamu.edu

Nitesh Saxena
Texas A&M University
College Station, Texas, USA
nsaxena@tamu.edu

ABSTRACT

Instant-messaging (IM) and voice over IP (VoIP) applications like WhatsApp, Zoom, and Skype have made people extremely reliant on online communications for their audio, video, and text conversations. Since more people are using these platforms to talk to each other and share sensitive information, many ongoing concerns have been raised about how the government and law enforcement monitor these platforms. Due to these concerns, the need for a method to secure confidential messages and electronic conversations has grown. This solution could be achieved by implementing an end-to-end encryption (E2EE) system without relying on any first or third parties, such as an online service or a centralized infrastructure like a public key infrastructure (PKI), which may be attacked, malicious, or coerced by law enforcement and government surveillance programs. In this systematization of knowledge paper, we first introduce the most popular E2EE apps, including their underlying E2EE messaging protocols. Then, based on the existing research literature, we investigate and systematize their E2EE features, including their underlying authentication ceremonies. Even though many research studies have examined some messaging services, we analyze and evaluate a broader set of the most popular E2EE apps and their underlying authentication ceremonies. Based on our evaluation, we have determined that all current E2EE apps, particularly when operating in opportunistic E2EE mode, are incapable of repelling active man-in-the-middle (MitM) attacks. In addition, we find that none of the current E2EE apps provide better and more usable authentication ceremonies, resulting in insecure E2EE communications against active MitM attacks. The conclusions of this systematization paper could influence future research in the field, including any improvements to the implementation of E2EE systems and authentication ceremonies that provide powerful protections against eavesdropping and MitM attacks.

CCS CONCEPTS

• **Security and privacy**;

KEYWORDS

E2EE apps, authentication ceremony, MitM attacks

WiSec '23, May 29-June 1, 2023, Guildford, United Kingdom

ACM ISBN 978-1-4503-9859-6/23/05…$15.00
https://doi.org/10.1145/3558482.3581773

ACM Reference Format:
Mashari Alatawi and Nitesh Saxena. 2023. SoK: An Analysis of End-to-End Encryption and Authentication Ceremonies in Secure Messaging Systems. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '23), May 29-June 1, 2023, Guildford, United Kingdom.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3558482.3581773

1 INTRODUCTION

End-to-end encryption (E2EE) ensures that the contents of a message are unintelligible to anyone except the sender and the intended recipient, preventing service providers, government surveillance programs, or man-in-the-middle (MitM) attackers from reading the exchanged messages. After the Snowden revelations about government snooping in 2013 [56], more instant messaging (IM) and voice over IP (VoIP) applications claimed to support E2EE. However, these IM and VoIP apps are susceptible to numerous forms of attack, including MitM and eavesdropping attacks [6, 72]. Government agencies, hackers, and even IM and VoIP service providers could monitor these apps to conduct surveillance programs or steal sensitive information [5, 10].

The community of security researchers began considering private chat apps and secure online communication systems long ago. In 2004, the off-the-record (OTR) protocol was proposed to provide E2EE [4]. It has been implemented as a plugin in common IM clients such as Pidgin [44], but it is not widely used due to usability issues [55, 62]. In 2013, Snowden's revelations increased public awareness of underlying privacy concerns [35]. Consequently, other alternatives have emerged in the form of new encrypted messaging systems that provide E2EE communications by adopting and expanding the OTR protocol. Open Whisper Systems released Signal, a new breakthrough E2EE protocol, in 2013 to provide E2EE as well as advanced security features such as forward secrecy and future secrecy [7, 16]. Signal was designed to enable both synchronous and asynchronous messaging environments [36]. The Signal protocol has been used by other IM and VoIP apps in recent years, including Signal and WhatsApp. Google is now implementing E2EE for rich communication services (RCSs) in Android Messages, and Zoom has also recently implemented it for meetings [3, 29]. Despite the fact that numerous IM and VoIP apps have included E2EE features, they vary in terms of their security and usability properties, privacy concepts, threat models, and security claims [62].

Our Contributions: Our contributions in this work are two fold. First, we examine the most popular E2EE apps [11, 14, 15, 20, 27, 32, 33, 39, 40, 51, 53, 54, 58, 60, 66, 69–71, 74] and their underlying E2EE messaging protocols. Based on the current research literature [1, 4, 7, 9, 16, 17, 21, 26, 36, 47–50, 57, 64], we then scrutinize and systematize their E2EE features, including their underlying

authentication ceremonies. Specifically, we examine the most crucial aspect of key management, i.e., verifying and authenticating key fingerprints (an authentication ceremony), and whether the verification process is susceptible to human errors, which could lead to MitM attacks. We also investigate and systematize the security and usability of authentication ceremonies used in E2EE apps. Even though several research studies have been done to explore popular messaging services in terms of their security, usability, and adoption, this work will examine a broader set of the most popular E2EE apps and their underlying authentication ceremonies. The outcomes of this systematization paper could provide valuable suggestions for future research to strengthen current E2EE implementations and enhance authentication ceremonies in E2EE systems.

2 BACKGROUND

2.1 State-of-the-Art End-to-End Encryption

The state-of-the-art E2EE implementation ensures that messages cannot be read by anyone except the endpoints of communication. Figure 1 displays how Alice and Bob encrypt messages using state-of-the-art E2EE. Therefore, the majority of E2EE apps utilize this E2EE scheme since it ensures robust end-to-end data confidentiality [62]. However, these apps use a service provider to store users' public keys, exchange public keys, and relay encrypted data between endpoints. This type of E2EE implementation, which relies on a server to distribute keys, can thwart a passive MitM attacker but cannot thwart an active MitM attacker, who can substitute keys and thereby compromise the entire communication between authorized users. Consequently, a malicious or hacked server can easily mount an attack known as a *key substitution attack* during the key-exchange service, compromising the entire E2EE system. Many E2EE apps let users take part in a hidden task called an authentication ceremony. During this task, users verify their key fingerprints and, if they do it right, defeat active MitM attackers.

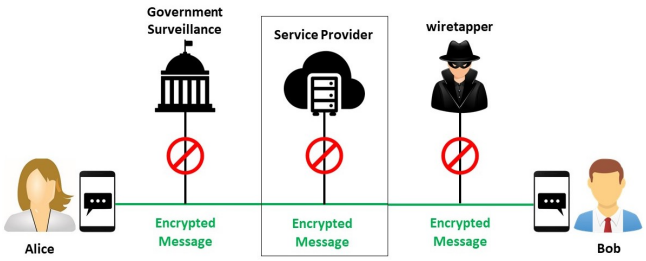


Figure 1: A high-level diagram of the state-of-the-art end-to-end encryption

2.2 Properties for Secure Messaging Systems

- **Confidentiality:** It keeps the contents of a message from being shared without permission. This means that only the sender and the intended recipient can read messages.
- **Integrity:** It ensures that a message has not been changed while being sent, so that the intended recipient gets the original message.
- **Authentication:** It exposes the identities of both the sender and the receiver in a private conversation, which ensures that a message was sent from the claimed sender.
- **Perfect Forward Secrecy:** It guarantees that data that has already been encrypted cannot be decrypted, even if all the key materials are compromised.
- **Future Secrecy:** It is also called *backward secrecy*, which ensures that encrypted data cannot be decrypted in the future, even if all the key materials are compromised.
- **Deniability:** To achieve this property, both conversation parties must be able to deny that they sent or made a message. This makes it impossible for other people to prove that a certain message was sent by a certain conversation party.

2.3 Threat Model

Inspired by a comprehensive survey on secure messaging by Unger et al. [62], we assume the existence of the following attackers:

- **Local Adversary:** An (active/passive) attacker who can control local networks on either side of a conversation, such as the owners of open wireless access points.
- **Global Adversary:** An (active/passive) attacker who can take over many parts of Internet service (e.g., powerful nation-states or large Internet service providers).
- **Service Providers:** All service operators could be considered as potential attackers when E2EE apps utilize a centralized infrastructure for distributing public keys and storing or forwarding messages, such as using a public-key directory.

As stated in [62], we assume that attackers can utilize E2EE apps, allowing them to create accounts and send messages as legitimate users. We also assume that the endpoints of E2EE apps are secure.

2.4 End-to-End Encrypted Messaging Protocols

2.4.1 Off-the-Record In 2004, the OTR protocol was introduced as a cryptographic protocol to enable the E2EE feature [4]. It was a substitute for pretty good privacy (PGP) to provide complete forward secrecy and deniable authentication, simulating private communication in the real world. Due to the vulnerability of the basic Diffie-Hellman key exchange protocol to MitM attacks, the OTR protocol uses a variation of the SIGMA protocol [28] as the authenticated key exchange to provide authentication [45]. The OTR protocol has been implemented as a plugin in standard IM clients such as Pidgin; however, researchers have found that these implementations have a number of usability problems [55, 62]. Furthermore, the OTR protocol does not support asynchronous messaging environments or group messaging because it was designed for synchronous messaging environments [12].

2.4.2 Signal The Signal Protocol was introduced in 2013 by Open Whisper Systems to provide E2EE as well as enhanced security features such as forward secrecy and future secrecy [7, 16]. It supports both synchronous and asynchronous messaging environments [36]. Signal uses the Extended Triple Diffie-Hellman (X3DH) key agreement protocol to establish a shared secret key between two users, who mutually authenticate one another based on their public keys, thereby ensuring forward secrecy and cryptographic deniability [37]. The X3DH protocol is designed for asynchronous environments, in which a user (Bob) can go offline after uploading information to a server, and another user (Alice) can use that information to send encrypted data to Bob, thereby establishing a shared secret key for future communication. Using the shared secret key, both

users can use the Double Ratchet algorithm to exchange encrypted messages [43]. The Double Ratchet algorithm leverages the key derivation function (KDF) chain to derive secret keys for encrypting messages. In recent years, the Signal protocol has been adopted by several E2EE apps. Furthermore, some protocols customize their own specifications to copy certain security features from the Signal protocol and thus implement the E2EE feature. For instance, the Matrix protocol [38] uses the Olm encryption library, which is based on the Signal protocol, to implement the E2EE feature in the Element app [11].

2.4.3 Proprietary and Other Protocols Several E2EE apps use their own proprietary protocols, such as Apple's iMessage, Telegram's MTProto protocol, and many other E2EE apps (discussed further in Section 4). Linphone [33] and Silent Phone [53] use the Zimmermann Real-time Transport Protocol (ZRTP) [73] to implement the E2EE feature for voice and video communications. ZRTP is a key agreement protocol that uses Diffie-Hellman key exchange to establish a shared secret between two endpoints. This shared secret is then used to establish secure real-time transport protocol (SRTP) sessions for VoIP apps [73]. However, the Diffie-Hellman key exchange is known to be susceptible to MitM attacks, and therefore, ZRTP uses a mechanism based on a short authentication string (SAS) to prevent this type of attack [63]. This SAS can be validated by end users to guarantee that no MitM attack has occurred.

2.5 Related Work

Even though various aspects of the secure messaging landscape have been systematized in prior research studies, this systematization of knowledge paper provides a unique and complementary perspective. Prior work has focused on secure messaging and conducted only a high-level investigation of the basic concepts and features of E2EE messaging protocols [4, 7, 16, 45]. Our work is, to the best of our knowledge, the first to scrutinize a broader set of the most popular E2EE apps, including their underlying authentication ceremonies. Some other papers also investigate the security of E2EE apps and the usability of their underlying authentication ceremonies; however, they do so without conducting a systematic study that covers a large number of E2EE apps, instead focusing only on one or a few apps [1, 17, 21, 26, 46, 47, 50, 64, 65]. These studies also lack a focus on E2EE security and the usability of the authentication ceremony in group-based scenarios. The most closely related work is by Herzberg et al. [22], which reveals the problems and limitations of the current authentication ceremony in some secure messaging apps. We share a common approach of bringing to light the importance of the authentication ceremony and its usability in current E2EE apps; however, we cover a large number of popular E2EE apps, and we focus not only on the authentication ceremony but also on the implementation of the E2EE feature in these E2EE apps. We also make a greater effort to apply a methodology with which to evaluate implementing the E2EE feature and authentication ceremonies in group-based scenarios.

3 SYSTEMATIZATION METHODOLOGY

Recent claims have been made that numerous IM and VoIP apps provide secure messaging solutions. However, they have been plagued by unclear security claims and usability issues [62]. To do a systematization of knowledge on the most popular E2EE apps, we developed and implemented the approach described in this section. Since the Snowden disclosures about widespread government surveillance in 2013 [56], both academia and industry have shown an increasing interest in developing secure communications solutions. In recent years, the number of E2EE apps has also increased significantly. Based on the existing research literature and publicly available messaging apps, we restricted our analysis to the most popular E2EE apps, focusing on the systematization and evaluation of how they implemented their E2EE functionality and their underlying authentication ceremonies. We examined the pertinent white papers, documentation, research literature, and E2EE protocol definitions. In addition to examining their E2EE functionality and implementation, we also investigated their underlying authentication ceremonies. We meticulously examined prominent E2EE apps (see Section 4). We limited our study to a collection of highly popular E2EE apps compatible with Android or iOS, based on the number of installations and ratings derived from the Google Play Store. The Apple App Store does not publicly disclose the number of app installations; however, we believe that the data currently available from the Google Play Store provides adequate information on app popularity. Table 3 in Appendix A.3 shows 17 highly popular E2EE apps and was last updated on December 25, 2022. We covered apps that implement the state-of-the-art E2EE feature and provide documentation of their E2EE functionality. Because the apps listed in Table 3 are all compatible with both Android and iOS, we had to include two additional apps in our study (namely, FaceTime and Messages by Apple) that are only supported on Apple devices as default apps but not supported on Android devices. Both apps also implement the state-of-the-art E2EE feature and provide documentation of their E2EE functionality. We looked at relevant white papers, E2EE documentation, and research literature based on top-tier conferences and Google Scholar citations. These academic and non-academic references were used to investigate how E2EE functionality is currently being used in E2EE apps. We specifically looked for the main E2EE protocol that the E2EE app uses to implement E2EE functionality and the cryptographic primitives that the main E2EE protocol depends on. We also conducted a practical analysis of the E2EE features provided by E2EE apps and the various code verification methods used by E2EE apps during their underlying authentication ceremonies. During this stage, we examined the usability of authentication ceremonies in E2EE apps and how human errors may impact usability and lead to MitM attacks based on the existing research literature. For the E2EE apps in focus, we intended to evaluate several criteria regarding their implementation of the E2EE feature, including their underlying E2EE message protocols and authentication ceremonies. The criteria being evaluated can be classified into two categories:

A. **Security**
   - The E2EE protocols used by E2EE applications to implement the E2EE feature.
   - Whether the E2EE feature is provided by default or as an optional property.
   - Whether or not an E2EE application implements the E2EE feature for text messaging and audio/video calls.

- Whether or not E2EE applications implement the E2EE feature in group scenarios, such as group messages and group audio or video calls.
- Whether the opportunistic E2EE mode is vulnerable to active MitM attacks.
- Whether an E2EE application provides a way for verifying and authenticating the key fingerprints (the authentication ceremony) to thwart active MitM attacks.
- Whether the authentication ceremony is a primary task or not.

B. **Usability**

- How users find and locate the authentication ceremony in order to perform it.
- The terminology that E2EE applications use to refer to the authentication ceremony.
- How a key fingerprint is represented to a user to participate in an authentication ceremony.
- How users are asked to do the authentication ceremony.
- How users perform the authentication ceremony in groups.
- Whether an E2EE application allows users to exchange their fingerprint codes via an out-of-band (OOB) channel directly from the app.
- Whether the authentication ceremony is vulnerable to human errors, which could lead to MitM attacks.

4 ANALYSIS OF E2EE APPLICATIONS

This section will compare the evaluated E2EE apps regarding criteria related to implementing the E2EE feature. A brief summary of these implementations can be found in Table 1. The results were primarily taken from our experiments examining the E2EE apps, as well as the E2EE documentation and the official security white papers of the corresponding E2EE apps. We examine how the E2EE feature is currently implemented in the most popular E2EE apps (only smartphone apps) that claim to offer E2EE messaging solutions. On these apps, the E2EE feature is either turned on by default or can be turned on by the user. In both cases, these apps use an opportunistic E2EE mode, which means they set up a secure channel between two parties without authenticating the other party [31]. This opportunistic E2EE mode can defeat a passive MitM attacker, but it cannot defeat an active MitM attacker who can change keys and put all communication between legitimate users at risk [21].

In our experiments, we examined every E2EE app in two stages. In the first stage, we analyzed relevant white papers and E2EE documentation to determine which E2EE protocol is used by each E2EE app and what cryptographic primitives are implemented by the E2EE protocol. In the second stage, we did our own tests to see how the E2EE feature worked in each E2EE app and how it was implemented in both one-to-one and group-based conversations. To this end, we used four different phone devices (namely Apple iPhone X, Apple iPhone 7 Plus, Samsung Android 5, and Google Pixel) and installed the latest version of each E2EE app on them. For one-to-one scenarios, we used the installed E2EE app to send a text message from one phone to another. This allowed us to see if the text message was encrypted by default in E2EE mode or if the user had to turn on the optional E2EE mode. We also followed the same procedure for audio and video conversations between two separate smartphones to assess whether the E2EE app offers the E2EE feature by default or as an opt-in when initiating audio and video calls. For group-based scenarios, we set up a group of three different smartphones in the E2EE app that supports group messaging. We then followed the same procedures for sending text messages as well as making audio and video calls in group-based scenarios. This enabled us to determine whether group-based text messaging, audio, and video conversations implement the E2EE functionality by default or as an opt-in. In E2EE apps, any conversation between two users is called a *one-to-one scenario*, while a conversation between more than two users is called a *group-based scenario*. Therefore, we elected to confine our analysis to a group size of three different smartphones. This was very helpful in providing results and lessons for our current study. However, in future studies, the group size could be increased to more than three to further investigate the E2EE functionality in contemporary messaging apps.

4.1 E2EE Apps Using the Signal Protocol

Most E2EE apps use the Signal protocol or rely heavily on custom protocols that copy some of the Signal protocol's security features. The Signal protocol is designed to work in both synchronous and asynchronous messaging environments, so it uses a key-distribution server to store the identities and ephemeral keys of its users. Frosch et al. [16] and Cohn-Gordon et al. [7] examined the security of the Signal protocol in their research studies. They found that users had to sign up and upload their long-term, medium-term, and ephemeral public keys to a key distribution server as part of the registration process. In [7], the authors also found that the key-distribution server could become a malicious server and, as a result, be used in MitM attacks. They found that Signal has an authentication ceremony that lets users verify public keys through an OOB channel. However, they had doubts about some implementations of the Signal protocol that might not require such a ceremony to be done. This would let a rogue server or an attacker with control over identity registration change keys and get messages from the other end. Herzberg et al. [21] examined how WhatsApp, Viber, Telegram, and Signal utilized E2EE and found that all four apps supported both the opportunistic E2EE and the authenticated E2EE modes. The authors stated that the authenticated E2EE mode matches the classical definition of E2EE, which protects users from a rogue MitM operator, while the opportunistic E2EE mode alone is not safe against this type of attack. They found that most users did not know what the difference was between these two modes and did not use them effectively. In the following, we introduce each E2EE app in more detail. We also present our evaluation, which goes into more depth about the E2EE features that these E2EE apps offer.

4.1.1 Facebook Messenger It is an IM application with voice and video calling capabilities, developed by Meta [14]. It uses the Signal protocol to implement E2EE functionality in chats and calls through a feature called *Secret Conversation* [13]. However, the *Secret Conversation* feature is not the default option, and therefore, users must enable the *Secret Conversation* mode manually and ask their intended recipients to enable the *Secret Conversation* mode on their devices as well. In addition to individual chats and calls, the Facebook Messenger app also implements the E2EE functionality in group chats and calls through the *Secret Conversation* feature.

4.1.2 Signal It is an application for IM and VoIP services [51]. It uses the Signal protocol to implement E2EE in all individual and group chats by default [52]. It also supports E2EE for voice and video communications between two parties and group video calls. RingRTC, an open-source video calling library written in Rust, is used by the Signal app to provide video and voice calling services on top of web real-time communication (WebRTC).

4.1.3 WhatsApp It is an application owned by Meta for IM and VoIP services [69]. It uses the Signal protocol to implement the E2EE feature by default in all messages and calls for all one-to-one and group scenarios [68]. In all one-to-one and group calls, a user initiates a voice or video call by establishing encrypted sessions with each of the devices of the recipient, such as those used in a messaging scenario. Once the call is made, SRTP is used to protect it with master secret keys that are made for each device of the recipient.

4.1.4 Other E2EE Applications Due to space constraints, other E2EE apps that use the Signal protocol are included in Appendix A.1. These E2EE apps are listed in Tables 1 and 2, but readers unfamiliar with them can refer to Appendix A.1.

4.2 E2EE Apps Using Proprietary Protocols

Here, we will introduce E2EE apps that implement their own proprietary protocols to provide the E2EE feature. We will also present our evaluation, in which we investigate their implementations of the E2EE feature in more detail.

4.2.1 Telegram It is a cloud-based messenger for IM and VoIP services [58]. It uses its customized protocol, called the MTProto protocol, to implement the E2EE feature in one-to-one chats and calls [59]. However, the E2EE feature is not supported in group scenarios. In all one-to-one scenarios, the Telegram app does not implement the E2EE functionality by default; thus, users must enable the *Secret Chat* option to protect their communications in an E2EE fashion. The Diffie-Hellman protocol is used to exchange cryptographic keys in the MTProto protocol. Once a *Secret Chat* is set up, the devices that are taking part in it exchange these keys.

4.2.2 Viber It is an IM and VoIP application owned by Rakuten [66]. It implements the E2EE feature by using the same concepts as the Signal protocol [67]. However, the Viber app uses its own implementation to protect all messages and calls in an E2EE fashion. In the Viber app, the E2EE feature is enabled by default in all one-to-one and group scenarios. In Viber calls, the audio and video call stream is converted to the SRTP protocol and encrypted with the Salsa20 algorithm.

4.2.3 Zoom It is a cloud platform for video meetings, VoIP, and team chat [74]. Zoom recently added the E2EE feature to Zoom meetings and Zoom Phone calls between two end users [29, 30]. By default, Zoom meetings and Zoom Phone calls between two end users are not E2EE. This means that users must turn on the E2EE feature through the Zoom web portal. To implement the E2EE feature, Zoom uses public-key cryptography to distribute a session key to all users who are taking part in a Zoom meeting [75]. Zoom uses Diffie-Hellman over Curve25519, the Edwards-curve digital signature algorithm (EdDSA), and the advanced encryption standard (AES) in GCM mode for its E2EE feature in Zoom meetings. Key derivation is done via the HMAC-based key derivation function (HKDF). Zoom uses the same cryptographic techniques and key management system as Zoom meetings for the E2EE feature in one-to-one Zoom Phone calls.

4.2.4 Other E2EE Applications Due to space constraints, other E2EE apps that use their own E2EE protocols can be found in Appendix A.2. These E2EE apps are listed in Tables 1 and 2, but readers unfamiliar with them can refer to Appendix A.2.

4.3 The Opacity of E2EE Applications

Many E2EE apps mislead users by claiming to be encrypted or secure communications platforms. According to a comprehensive survey of secure messaging conducted by Unger et al. [62], several of these apps do not provide E2EE messaging solutions as advertised. Not all E2EE apps support the E2EE feature by default, and that may confuse new users who use these apps for sending sensitive information. Additionally, as was mentioned before, the opportunistic E2EE mode is resistant to passive MitM attacks but susceptible to active MitM attacks. The majority of E2EE apps alert users that the opportunistic E2EE mode is activated and their communications are E2EE by using various indicators, such as special notification messages and lock icons, to indicate that the mode is enabled (see Figure 3 in Appendix A.3). This could make it more difficult for average users to detect active MitM attacks, especially if they are unaware of the security risks caused by not verifying key fingerprints. In [1], Abu-Salma et al. conducted a user study with 22 participants (eleven of whom were Telegram users) and investigated several security elements of the Telegram app. The authors reported that the design of the user interface had a detrimental impact on the behavior of the participants during the authentication ceremony due to several design issues. They also found that all participants were unaware of the usefulness of fingerprints. In addition, they observed that, despite having prior experience with Telegram, none of the eleven users had used the key fingerprints. Users must therefore participate in an authentication ceremony to verify their key fingerprints and thwart active MitM attacks. Participation in the verification and authentication of these key fingerprints will enable the authenticated E2EE mode, which is supported by the majority of E2EE apps (discussed further in Section 5).

5 ANALYSIS OF THE AUTHENTICATION CEREMONY

After the previous section investigated the current implementation of the E2EE feature in many E2EE apps, this section builds on that and examines the usability of the authentication ceremony in E2EE apps and how users can participate in such an authentication ceremony. For each E2EE app, we evaluate the implementation of the authentication-ceremony-related criteria outlined in Section 3. This provides an overview of the differences between E2EE apps as well as their underlying authentication ceremonies. We examine the current implementation of the authentication ceremony in E2EE apps based on relevant white papers, documentation, and academic literature, while some information was collected by examining the E2EE apps. All E2EE apps analyzed in this study use the same approach to implement the authentication ceremony, which consists

Table 1: The Security of Implementing End-to-End Encryption Feature in End-to-End Encrypted Applications

Application	E2EE Protocol	E2EE Feature in One-to-One Scenario				E2EE Feature in Group Scenario				Vulnerable to Active MitM Attack	Providing a Method to Switch to the Authenticated E2EE Mode
		E2EE Mode	In Chat	In Audio Call	In Video Call	E2EE Mode	In Chat	In Audio Call	In Video Call		
Element	Proprietary	Opportunistic by default	✓	✓	✓	Opportunistic by default	✓	✓	✓	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Facebook Messenger	Signal	Opportunistic via an opt-in	✓	✓	✓	Opportunistic via an opt-in	✓	✓	✓	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
FaceTime	Proprietary	Opportunistic by default	N/A	✓	✓	Opportunistic by default	N/A	✓	✓	Yes	No
Google Meet	Signal	Opportunistic by default	✓	✓	✓	Opportunistic by default	✓	✓	✓	Yes	No
KakaoTalk	Proprietary	Opportunistic via an opt-in	✓	✗	✗	Opportunistic via an opt-in	✓	✗	✗	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
LINE	Proprietary	Opportunistic by default	✓	✓	✓	Opportunistic by default	✓	✗	✗	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Linphone	Proprietary	Opportunistic via an opt-in	✓	✓	✓	Opportunistic via an opt-in	✓	✗	✗	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Messages by Apple	Proprietary	Opportunistic by default via iMessage	✓	N/A	N/A	Opportunistic by default via iMessage	✓	N/A	N/A	Yes	No
Messages by Google	Signal	Opportunistic by default in RCS	✓	N/A	N/A	N/A	N/A	N/A	N/A	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Signal	Signal	Opportunistic by default	✓	✓	✓	Opportunistic by default	✓	✓	✓	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Silent Phone	Proprietary	Opportunistic by default	✓	✓	✓	Opportunistic by default	✓	✓	✓	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Skype	Signal	Opportunistic via an opt-in	✓	✓	✗	N/A	N/A	N/A	N/A	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Telegram	Proprietary	Opportunistic via an opt-in	✓	✓	✓	N/A	N/A	N/A	N/A	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Threema	Proprietary	Opportunistic by default	✓	✓	✓	Opportunistic by default	✓	✓	✓	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Viber	Proprietary	Opportunistic by default	✓	✓	✓	Opportunistic by default	✓	✓	✓	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
WhatsApp	Signal	Opportunistic by default	✓	✓	✓	Opportunistic by default	✓	✓	✓	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Wickr	Proprietary	Opportunistic by default	✓	✓	✓	Opportunistic by default	✓	✓	✓	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Wire	Proprietary	Opportunistic by default	✓	✓	✓	Opportunistic by default	✓	✓	✓	Yes	Yes, it relies on users to optionally perform an authentication ceremony.
Zoom	Proprietary	Opportunistic via an opt-in	✓	✓	✓	Opportunistic via an opt-in	✓	✓	✓	Yes	Yes, it relies on users to optionally perform an authentication ceremony.

✓ indicates that the E2EE feature is provided, and ✗ indicates that the E2EE feature is not provided.

of making this task optional, relying on users to find and perform it, and providing users with similar code representations to compare and verify their key fingerprints. Therefore, instead of focusing on a single app as in the previous section, this section examines and evaluates the authentication ceremony as a whole, using the knowledge gained from examining the E2EE apps and relevant references. In the following subsections, we will provide an in-depth analysis of the authentication ceremony and its usability in all E2EE apps. Note that in practice, some E2EE apps have the same usability challenges and technical concerns with the authentication ceremony. A brief summary of this analysis can be found in Table 2.

5.1 Finding and Performing the Ceremony

Participating in an authentication ceremony and successfully completing it will enable the authenticated E2EE mode, which is consistent with the traditional definition of E2EE. In contrast to the opportunistic E2EE mode, the authenticated E2EE mode guarantees that no active MitM attackers are involved in any private conversation between two end users. Whenever Alice and Bob want to communicate using an E2EE app, they both use a service provider to exchange their public encryption keys and establish a secret shared key for future communication. This secret shared key is only known to Alice and Bob. No one else, not even the service provider, can find out what the value of the secret shared key is or decrypt any of the messages being sent. However, these service providers could use fake public keys during the key-exchange service to get around the protection that E2EE apps offer against rogue or compromised service providers. For example, when Alice wants to talk to Bob through an E2EE app, she will get his public key from a service provider to encrypt a shared secret key and then send it to him through the service provider. However, a malicious provider can easily mount a *key substitution attack* and furnish her with a phony public key under its control. The rogue provider can now decrypt the shared secret key, re-encrypt it using Bob's real

public key, and then send it to Bob, claiming that it was sent by Alice. Thus, the rogue provider has become an active MitM attacker between Alice and Bob. This means that the attacker can read or change the messages Alice and Bob send each other without the knowledge of the two parties being attacked. In the real world, all E2EE apps mentioned in Section 4 implement this opportunistic E2EE mode by default, which is only known to be secure against passive MitM attacks. To switch to the authenticated E2EE mode and thwart active MitM attacks, both end users need to participate in an authentication ceremony and successfully complete it.

Despite the significance of the authentication ceremony in detecting active MitM attacks, the authentication ceremony is optional in all current E2EE apps. Consequently, users may be susceptible to human errors, which can result in MitM attacks. Therefore, the authenticated E2EE mode depends on the users and how they interact in the authentication ceremony to establish trust and enable secure communication. It is also common for users to ignore the authentication process until they are encouraged to do so, at which point they may struggle and misunderstand the steps, leaving themselves vulnerable to MitM attacks. In practice, all the E2EE apps listed in Section 4 (that provide a mechanism for performing the authentication ceremony) rely on end-users to activate the authenticated E2EE mode, from being aware of the security risks and the importance of authentication in preventing such an attack, to taking the necessary steps for the authentication ceremony to be successful. This includes navigating the app's settings and menu system to find the terminology used to refer to the authentication ceremony. Figure 4 in Appendix A.3 depicts some E2EE apps and the terminologies they use to refer to the authentication ceremony. After locating the authentication ceremony, the end-users must compare and verify the key fingerprints before deciding whether or not to continue communicating. Also, end users must comprehend the meaning of failure (non-matching key fingerprints) to cease communication. As shown in Table 2, all E2EE apps in Section 4 (that offer a mechanism for performing the authentication ceremony) use different terminologies and representations of the key fingerprints in their authentication ceremonies.

5.2 Fingerprint Representations

During the authentication ceremony, many E2EE apps use textual (words and sentences), numeric, hexadecimal, and graphical fingerprint representations. The representation of the key fingerprints is an essential component of the authentication ceremony in all E2EE apps, and it plays a significant role in assisting users to perform the authentication ceremony correctly and thwart active MitM attacks. In such authentication ceremonies, E2EE apps represent the fingerprints of the encryption key or the fingerprints of the public keys of other users using a variety of approaches (see Table 2). These fingerprints can be compared and verified in person or over an OOB channel, such as a text message, email, or phone call. Such a fingerprint is encoded into a readable/exchangeable code to facilitate manual comparison and verification. In the real world, all E2EE apps listed in Section 4 (that offer a mechanism for performing the authentication ceremony) generate the fingerprint and represent it as a human-readable code or an exchangeable object. Figure 5 in Appendix A.3 displays the common fingerprint representations used by E2EE apps and described here:

5.2.1 QR Code The key fingerprint is encoded into a QR code that can be automatically captured and compared by the E2EE app without the need for end-user intervention. This method works best when the authentication ceremony is performed in person and the QR code is scanned in person. Only 5 E2EE apps analyzed in this study (Element, Signal, Threema, WhatsApp, and Wickr) offer this method to users who are located in close proximity to one another, allowing them to perform the authentication ceremony in person. Figure 5a in Appendix A.3 shows the QR code representation for the Signal app.

5.2.2 Numeric Representation The key fingerprint is represented as a sequence of numerical digits to facilitate comparison and verification. To make a long code more readable, this method is organized as blocks (or chunks) of numbers with few digits. For example, the WhatsApp app, shown in Figure 5b in Appendix A.3, uses a 60-digit numeric string that is broken up into 12 blocks of five-digit numbers. This method can be used either in person or remotely to compare and verify the key fingerprint. It is useful for people who are in distant locations and unlikely to meet in person prior to communicating via an E2EE app. Only 6 E2EE apps analyzed in this paper (Messages by Google, Signal, Skype, Viber, WhatsApp, and Zoom) offer this method to their users, whether they are nearby or remote. However, in the real world, only Signal and WhatsApp offer a feature for directly exchanging the key fingerprint from the app over an OOB channel in remote communications. Other apps only rely on users to compare and verify the key fingerprint over an OOB channel of their choice. Figure 5b in Appendix A.3 shows the WhatsApp app's numerical key fingerprint and the share icon at the top-right corner of the phone's screen, which is used to directly exchange the key fingerprint from the WhatsApp app over an OOB channel to perform the authentication ceremony remotely.

5.2.3 Alphanumeric Representation For the purposes of comparison and verification, the key fingerprint is displayed both numerically and alphabetically. This approach can be used to divide a string of characters into equal-sized chunks, improving the text's readability. It can be used in hexadecimal, base32, or base64 format. Only 7 E2EE apps analyzed in this work (Facebook Messenger, KakaoTalk, LINE, Telegram, Threema, Wickr, and Wire) offer this method for comparing and verifying the key fingerprint, either in person or remotely. In practice, all of the aforementioned E2EE apps display the key fingerprint in hexadecimal characters (Figure 5c in Appendix A.3 shows the hexadecimal representation for the Telegram app), and none of them offer a feature to directly exchange the key fingerprint within the app, with the exception of the Wickr app, which displays the key fingerprint in base32 characters (as shown in Figure 5d in Appendix A.3). Although the Base64 representation has also been proposed in the literature, none of the E2EE apps we analyze in this paper currently use it.

5.2.4 Graphical Representation The key fingerprint is encoded into an image or a sequence of emojis for comparison and verification. This method can be used to replace textual fingerprint representations and has been suggested to improve usability in the prior literature. Only 3 E2EE apps analyzed in this study (Element, KakaoTalk, and Telegram) offer this method for comparing and

verifying the key fingerprint, either in person or remotely. For example, Figure 5e in Appendix A.3 depicts the image derived from the encryption key for the KakaoTalk app. Also, Figure 5f in Appendix A.3 shows the emoji that the Element app uses to compare and verify the key fingerprint.

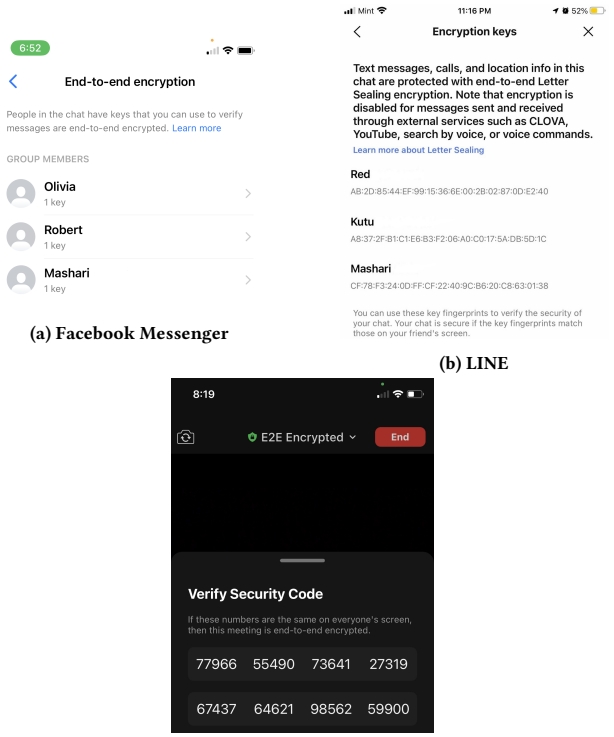
5.2.5 Short Authentication String Only two E2EE apps analyzed in this work (Linphone and Silent Phone) use a SAS-based verification mechanism (e.g., four characters or two words) instead of alphanumeric or numeric representations to make comparison and verification tasks during authentication ceremonies more readable. Figure 5g in Appendix A.3 shows the authentication ceremony for the Linphone app, which uses a SAS code with four characters to verify the user's identity. Figure 5h in Appendix A.3 shows the authentication ceremony for the Silent Phone app, which uses a SAS code with two words to verify the user's identity.

5.3 Supporting OOB Channels

Unfortunately, only 7 of the apps analyzed in this study (Linphone, Signal, Silent Phone, Threema, Viber, WhatsApp, and Wickr) provide a feature for directly exchanging the key fingerprint from inside the app using an OOB channel, e.g., a text message, email, or phone call. These apps rely on users' decisions to use another trusted means of communication to compare and verify their key fingerprints. Having such an OOB channel can facilitate authentication ceremonies, especially for users who are not close to one another. Additionally, some E2EE apps use numeric representations and do not support OOB channels via which users can perform the authentication ceremony. For example, Zoom uses numeric representations for its E2EE meeting codes, but it does not support OOB channels to perform the authentication ceremony manually or even any automated verification process. Instead, the meeting host can read the security code aloud, and therefore, users can compare and verify that their clients display the same security code. However, an adversary can sit and copy the meeting host's voice when announcing the string of digits 0-9 from previous meetings and, thus, mount a voice-reordering attack to create any digits the adversary wants and thus compromise the E2EE feature in a future meeting [48].

5.4 E2EE Group Communications

In general, the security issues and usability obstacles of E2EE apps in group scenarios are similar to those in the one-on-one scenarios. In [26], the authors conducted a security analysis of *Letter Sealing*, which is the E2EE scheme for LINE. They found that *Letter Sealing* does not meet one of the fundamental security requirements of E2EE, which is the integrity of the message. Their results demonstrated the feasibility of attacks by exploiting several vulnerabilities in LINE's E2EE system. These attacks can be mounted by an end-to-end adversary, a malicious group member, or a malicious user. For instance, a malicious group member can mount impersonation or forgery attacks on the group message encryption scheme by exploiting the vulnerability of the key-derivation stage in group message encryption.



(a) Facebook Messenger **(b) LINE**

(c) Zoom

Figure 2: Authentication ceremonies in E2EE group scenarios

Most of the E2EE apps analyzed in this study support E2EE group communications. However, participating in an authentication ceremony is not group-based authentication. It is pairwise authentication, which is similar to a one-to-one authenticated scheme, such as the authentication ceremony for two parties. Therefore, having a group with an enormous number of group members will make the authentication ceremony hugely problematic and more challenging to perform. Like in one-to-one encrypted communication, the group members in end-to-end encrypted group communications must perform an authentication ceremony to guarantee that conversations between group members are confidential and authenticated, thereby ensuring that there are no MitM attacks. In fact, the vast majority of E2EE apps follow the same procedures for the authentication ceremony in all one-to-one and group scenarios. For instance, WhatsApp does not allow its users, when in a group chat, to authenticate one another in a group-based manner; rather, it relies on pairwise individual authentication. However, some other E2EE apps vary in terms of their authentication ceremony settings regarding presenting keys to their users during the authentication ceremony while keeping the fingerprint verification pairwise. For instance, the Facebook Messenger app (as shown in Figure 2a) uses different terminology to refer to its group-authentication ceremony. It lists all group members in a single list to help users find each member's device key, which may increase the usability of the authentication ceremony in a group scenario. Another app (LINE in Figure 2b) lists all group members with their keys in one list, which could be helpful for group members in terms of participating in the group authentication ceremony as well. On the other hand, there is only one app (Zoom) in which the authentication ceremony

Table 2: The Usability of the Authentication Ceremony in End-to-End Encrypted Applications

Application	Terminology Used to Refer to the Authentication Ceremony	Fingerprint Representation	Authentication Ceremony to Enable the Authenticated E2EE Mode	Group Ceremony	Supporting OOB Channels	Vulnerable to Human Errors
Element	In 1-to-1 and group scenarios: (Verify) One-time code	QR code, On-screen emojis (7 emojis)	Scan QR code, Manually compare emojis	Pairwise Based Authentication	No	Yes, since users manually compare emojis.
Facebook Messenger	In 1-to-1 scenario: (Device keys), In a group scenario: (End-to-end encryption) Device keys	66 hexadecimal characters for each device	Manually compare characters	Pairwise Based Authentication	No	Yes, since users manually compare characters.
FaceTime	N/A	N/A	N/A	N/A	N/A	N/A
Google Meet	N/A	N/A	N/A	N/A	N/A	N/A
KakaoTalk	In 1-to-1 and group scenarios: (Public Key)	Image, 32 hexadecimal characters for each party	Manually compare image or characters	Pairwise Based Authentication	No	Yes, since users manually compare image or characters.
LINE	In 1-to-1 and group scenarios: (Encryption keys)	32 hexadecimal characters for each party	Manually compare characters	Pairwise Based Authentication	No	Yes, since users manually compare characters.
Linphone	In 1-to-1 and group scenarios: (Call the contact) Communication security	SAS code of 4 characters	Manually compare SAS via audio call	Pairwise Based Authentication	Yes	Yes, since users manually compare SAS via audio call.
Messages by Apple	N/A	N/A	N/A	N/A	N/A	N/A
Messages by Google	In 1-to-1 scenario: (Verify encryption) Verification code	60-digit numeric	Manually compare numbers	N/A	No	Yes, since users manually compare numbers.
Signal	In 1-to-1 and group scenarios: (View Safety Number) Verify Safety Number	QR code, 60-digit numeric	Scan QR code, Manually compare numbers	Pairwise Based Authentication	Yes	Yes, since users manually compare numbers.
Silent Phone	In 1-to-1 and group scenarios: (Call the contact) Verify SAS	Two words	Manually compare words via audio call	Pairwise Based Authentication	Yes	Yes, since users manually compare words via audio call.
Skype	In 1-to-1 scenario: (Security Code)	60-digit numeric	Manually compare numbers	N/A	No	Yes, since users manually compare numbers.
Telegram	In 1-to-1 scenario: (Encryption Key)	Image, 64 hexadecimal characters	Manually compare image or characters	N/A	No	Yes, since users manually compare image or characters.
Threema	In 1-to-1 and group scenarios: (Key Fingerprint)	QR code, 32 hexadecimal characters	Scan QR code, Manually compare characters	Pairwise Based Authentication	Yes	Yes, since users manually compare characters.
Viber	In 1-to-1 and group scenarios: Turn on (Trusted Contacts) (Call your contact) Verify secret identification key	48-digit numeric	Manually compare numbers via audio call	Pairwise Based Authentication	Yes	Yes, since users manually compare numbers via audio call.
WhatsApp	In 1-to-1 and group scenarios: (Encryption) Verify Security Code	QR code, 60-digit numeric	Scan QR code, Manually compare numbers	Pairwise Based Authentication	Yes	Yes, since users manually compare numbers.
Wickr	In 1-to-1 and group scenarios: (Security Verification) Compare security code	QR code, 51 characters using Base32 scheme	Scan QR code, Manually compare characters	Pairwise Based Authentication	Yes	Yes, since users manually compare characters.
Wire	In 1-to-1 and group scenarios: (Devices) Verify device fingerprint	64 hexadecimal characters for each device	Manually compare characters	Pairwise Based Authentication	No	Yes, since users manually compare characters.
Zoom	In 1-to-1 and group scenarios: (Encryption) Verify security code	40-digit numeric	Manually compare numbers	Group-based Authentication	No	Yes, since users manually compare numbers.

is group-based. The Zoom app, as shown in Figure 2c, uses only one security code for its Zoom meeting setting to verify the security code for all Zoom meeting members in the current session. Here, Zoom allows users to compare and verify a 40-digit number represented as 8 blocks of five-digit numbers to verify the secure connection of their Zoom session. Therefore, the meeting host may read the security code aloud, and then all users can compare and verify that their clients display the same security code.

6 DISCUSSION AND RECOMMENDATIONS

In this section, we will discuss and recommend some possible improvements for implementing E2EE functionality and authentication ceremonies in current E2EE apps. These recommendations are based on the knowledge gained from our test scenarios. Also, it is important to note that these recommendations should go through testing before being deployed in E2EE apps.

Some E2EE apps analyzed in this study (Facebook Messenger, KakaoTalk, Linphone, Skype, Telegram, and Zoom) do not implement the E2EE feature by default. Users will have to manually turn on the E2EE feature to keep their conversations secure. This may cause confusion for users unfamiliar with the E2EE scheme. Abu-Salma et al. [2] investigated users' experiences with various communication tools and their perceptions of the tools' security features. They found that users sent sensitive information using Telegram's default chat, which is not E2EE. In practice, regular users may feel misled by E2EE claims. Therefore, we suggest that any application that purports to offer a secure E2EE messaging solution should implement E2EE functionality by default rather than as an opt-in feature. We also suggest that E2EE apps should ask their users to perform the authentication ceremony as a primary task. All E2EE apps analyzed in this work implement the E2EE feature in an opportunistic E2EE mode, whether as a default or an opt-in option. In practice, this opportunistic E2EE mode is susceptible to active MitM attacks; hence, users must complete the authentication ceremony to activate the authenticated E2EE mode. However, the authentication ceremony is optional in all existing E2EE apps. This might make active MitM attacks more difficult to detect, especially for average users who are unaware of the security risks associated with skipping or clicking through the authentication ceremony.

The performance of the authentication ceremony in current messaging apps has been the focus of numerous published academic studies. Due to usability flaws and human mistakes, it has been proven that users cannot perform the authentication ceremony and are hence vulnerable to MitM attacks. In a research study conducted by Schröder et al. [47], the authors found that users failed to complete the authentication ceremony in the Signal app and were therefore susceptible to MitM attacks due to usability issues. In a study by Vaziripour et al. [64], the authors investigated the ease of locating and completing the authentication ceremony in WhatsApp, Viber, and Facebook Messenger. They found that, due to a lack of security knowledge and various user interface design flaws, participants struggled to locate and perform the authentication ceremony. Furthermore, studies conducted by Herzberg et al. [21] and Shirvanian et al. [50] investigated the usability of performing the authentication ceremony in WhatsApp, Viber, Telegram, and Signal and found that participants were vulnerable to MitM attacks. In [21], the authors showed that the majority of participants failed to authenticate even when they were shown how to authenticate. In [50], the authors demonstrated that participants did not perform remote authentication ceremonies correctly due to usability difficulties and human errors. To help users locate and find the authentication ceremony, we suggest that E2EE apps should give a notification message at the beginning of the conversation. This message can help inform users about the importance of completing the authentication ceremony to prevent MitM attacks. Also, we suggest that E2EE apps should give users the possibility of navigating to the authentication ceremony from the conversation interface if they want to. This is because the primary task of the users in all current E2EE apps is to pursue a conversation, and the authentication ceremony is only an optional task. On the other hand, many but not all E2EE apps analyzed in this study do not provide a feature for directly exchanging the key fingerprint from inside the app using an OOB channel, e.g., a text message or email. This feature can help users complete the authentication ceremony, especially if they are not nearby. Therefore, we think that all E2EE apps should have this feature so that users can exchange their key fingerprints from inside the app via an OOB channel.

Most of the E2EE apps mentioned in Section 4 still use numeric or hexadecimal representations of fingerprints, even though many studies have shown that other representations, like words and sentences, are better at helping users detect attacks. Dechand et al. [9] conducted a user study to investigate the performance and usability of six textual key-fingerprint representations. They found that participants were more resistant to attacks when using words and sentences as compared to numeric or alphanumeric (Hexadecimal and Base32) representations. The authors reported that the hexadecimal representation scheme fared considerably worse than other representation schemes in terms of detecting attacks and usability evaluations. Similarly, another work by Tan et al. [57] examined the usability and security of eight textual and visual fingerprint representations. They found that visual fingerprint representations were more vulnerable to attacks than other methods, even though they were easy to use and quick to process. In [64], the authors investigated the authentication ceremony in WhatsApp, Viber and Facebook Messenger. During this study, the authors observed that many participants felt that the string of digits and the hexadecimal string used for fingerprint representation were excessively long. In addition, there are some studies that have found that E2EE phones are susceptible to MitM attacks due to human errors. For example, a study by Shirvanian et al. [49] examined the security and usability of E2EE phones. The authors considered two words and four words in the checksum-comparison and speaker-verification tasks. They found that users were vulnerable to MitM attacks due to their failures in the checksum-comparison and speaker-verification tasks. Furthermore, most contemporary E2EE apps use numeric representations in their authentication mechanisms, which users have complained about, according to the research literature. Therefore, we recommend that E2EE apps use textual and visual representations that make the authentication process easier for users. However, more research is needed to study the security vulnerabilities of these representations.

7 CONCLUSION

In this paper, we examined the most popular E2EE apps, including their underlying E2EE messaging protocols and authentication ceremonies. Even though the authentication ceremony plays a vital role in helping to thwart active MitM attacks, a few E2EE apps do not offer any authentication ceremony to their users. We found that the current implementations of the E2EE feature in various E2EE apps, particularly in the opportunistic E2EE mode, can defeat a passive MitM attacker but cannot defeat an active MitM attacker. We also found that their actual implementations of the E2EE feature in authenticated E2EE mode depend crucially on users to successfully perform and complete authentication ceremonies. However, several studies have shown that users are unable to successfully perform and complete the authentication ceremony and, therefore, become vulnerable to active MitM attacks due to usability issues and human errors. This systematization reveals avenues that require further investigation. First, further research is needed to automate the authentication ceremony or implement a semi-automated authentication ceremony to reduce the effort on the part of the user when performing the authentication ceremony. Additionally, more research is needed to extend studies to the context of group communication. Most research studies focus only on two-party E2EE but having more than two parties will make the authentication ceremony more challenging to perform. Lastly, new research can be focused on running the E2EE protocol over the audio channel only. Most research studies focus only on phones, which always have two channels (a data channel and an audio channel). Therefore, new research is needed to demonstrate how to establish this E2EE protocol on line phones, which have only audio channels.

ACKNOWLEDGMENTS

We would like to give special thanks to our shepherd and the anonymous reviewers for their valuable feedback on this paper.

REFERENCES

[1] Ruba Abu-Salma, Kat Krol, Simon Parkin, Victoria Koh, Kevin Kwan, Jazib Mahboob, Zahra Traboulsi, and M Angela Sasse. 2017. The Security Blanket of the Chat World: An Analytic Evaluation and a User Study of Telegram. Internet Society. https://doi.org/10.14722/eurousec.2017.23006

[2] Ruba Abu-Salma, M. Angela Sasse, Joseph Bonneau, Anastasia Danilova, Alena Naiakshina, and Matthew Smith. 2017. Obstacles to the Adoption of Secure Communication Tools. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 137–153. https://doi.org/10.1109/SP.2017.65

[3] Dieter Bohn. 2020. *Google is rolling out end-to-end encryption for RCS in Android Messages beta.* Retrieved July 11, 2022 from https://www.theverge.com/2020/11/19/21574451/android-rcs-encryption-message-end-to-end-beta

[4] Nikita Borisov, Ian Goldberg, and Eric Brewer. 2004. Off-the-Record Communication, or, Why Not to Use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society (WPES '04)*. Association for Computing Machinery, New York, NY, USA, 77–84. https://doi.org/10.1145/1029179.1029200

[5] Pew Research Center. 2017. *Most Americans think the government could be monitoring their phone calls and emails.* Retrieved July 03, 2022 from https://pewrsr.ch/3nI8hIf

[6] Don Clark. 2015. *Microsoft to Alert Users to Suspected Government Snooping.* Retrieved July 03, 2022 from https://www.wsj.com/articles/microsoft-to-alert-users-to-suspected-government-snooping-1451528624

[7] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2020. A formal security analysis of the signal messaging protocol. *Journal of Cryptology* 33, 4 (2020), 1914–1983. https://doi.org/10.1007/s00145-020-09360-1

[8] LINE Corporation. 2021. *LINE Encryption Overview.* Retrieved August 17, 2022 from https://d.line-scdn.net/stf/linecorp/en/csr/line-encryption-whitepaper-ver2.1.pdf

[9] Sergej Dechand, Dominik Schürmann, Karoline Busse, Yasemin Acar, Sascha Fahl, and Matthew Smith. 2016. An Empirical Study of Textual Key-Fingerprint Representations. In *Proceedings of the 25th USENIX Conference on Security Symposium (SEC'16)*. USENIX Association, USA, 193–208.

[10] Kitty Donaldson and Mark Burton. 2019. *Facebook, WhatsApp Will Have to Share Messages With U.K.* Retrieved July 03, 2022 from https://www.bloomberg.com/news/articles/2019-09-28/facebook-whatsapp-will-have-to-share-messages-with-u-k-police

[11] Element 2022. https://element.io/.

[12] Ksenia Ermoshina, Francesca Musiani, and Harry Halpin. 2016. End-to-End Encrypted Messaging Protocols: An Overview. In *International Conference on Internet Science*. Springer, 244–254. https://doi.org/10.1007/978-3-319-45982-0_22

[13] Facebook. 2017. *Messenger Secret Conversations. Technical Whitepaper.* Retrieved July 18, 2022 from https://about.fb.com/wp-content/uploads/2016/07/messenger-secret-conversations-technical-whitepaper.pdf

[14] Facebook Messenger 2022. https://www.messenger.com/.

[15] FaceTime 2022. https://support.apple.com/en-us/HT204380.

[16] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk, and Thorsten Holz. 2016. How Secure is TextSecure?. In *2016 IEEE European Symposium on Security and Privacy (EuroSP)*. IEEE, 457–472. https://doi.org/10.1109/EuroSP.2016.41

[17] Christina Garman, Matthew Green, Gabriel Kaptchuk, Ian Miers, and Michael Rushanan. 2016. Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple Imessage. In *Proceedings of the 25th USENIX Conference on Security Symposium (SEC'16)*. USENIX Association, USA, 655–672.

[18] Wire Swiss GmbH. 2021. *Wire Security Whitepaper.* Retrieved August 20, 2022 from https://wire-docs.wire.com/download/Wire+Security+Whitepaper.pdf

[19] Google. 2022. *Messages End-to-End Encryption Overview.* Retrieved July 18, 2022 from https://www.gstatic.com/messages/papers/messages_e2ee.pdf

[20] Google Meet 2022. https://apps.google.com/meet/.

[21] Amir Herzberg and Hemi Leibowitz. 2016. Can Johnny Finally Encrypt? Evaluating E2E-Encryption in Popular IM Applications. In *Proceedings of the 6th Workshop on Socio-Technical Aspects in Security and Trust (STAST '16)*. Association for Computing Machinery, New York, NY, USA, 17–28. https://doi.org/10.1145/3046055.3046059

[22] Amir Herzberg, Hemi Leibowitz, Kent Seamons, Elham Vaziripour, Justin Wu, and Daniel Zappala. 2021. Secure Messaging Authentication Ceremonies Are Broken. *IEEE Security & Privacy* 19, 2 (2021), 29–37. https://doi.org/10.1109/MSEC.2020.3039727

[23] Chris Howell, Tom Leavy, and Joël Alwen. 2017. *Wickr Messaging Protocol. TECHNICAL PAPER.* Retrieved August 04, 2022 from https://wickr.com/wp-content/uploads/2019/12/WhitePaper_WickrMessagingProtocol.pdf

[24] Apple Inc. 2021. *Apple Platform Security. iMessage security overview.* Retrieved July 27, 2022 from https://support.apple.com/guide/security/imessage-security-overview-secd9764312f/web

[25] Apple Inc. 2022. *Apple Platform Security. FaceTime security.* Retrieved July 27, 2022 from https://support.apple.com/guide/security/facetime-security-seca331c55cd/web

[26] Takanori Isobe and Kazuhiko Minematsu. 2018. Breaking Message Integrity of an End-to-End Encryption Scheme of LINE. In *European Symposium on Research in Computer Security*, Javier Lopez, Jianying Zhou, and Miguel Soriano (Eds.). Springer, Springer International Publishing, Cham, 249–268. https://doi.org/10.1007/978-3-319-98989-1_13

[27] KakaoTalk 2022. https://www.kakaocorp.com/service/KakaoTalk?lang=en.

[28] Hugo Krawczyk. 2003. SIGMA: The 'SIGn-and-MAc'approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *Annual International Cryptology Conference*. Springer, 400–425. https://doi.org/10.1007/978-3-540-45146-4_24

[29] Max Krohn. 2020. *Zoom Rolling Out End-to-End Encryption Offering.* Retrieved July 11, 2022 from https://blog.zoom.us/zoom-rolling-out-end-to-end-encryption-offering/

[30] Max Krohn. 2022. *End-to-End Encryption Expands to Zoom Phone and Breakout Rooms.* Retrieved August 11, 2022 from https://blog.zoom.us/end-to-end-encryption-zoom-phone-breakout-rooms/

[31] Adam Langley. 2009. Opportunistic encryption everywhere. *In W2SP* (2009).

[32] Line 2022. https://line.me/en/.

[33] Linphone 2020. https://www.linphone.org/.

[34] Linphone. 2020. *LIME.* Retrieved August 17, 2022 from https://www.linphone.org/technical-corner/lime

[35] MARY MADDEN. 2014. *Public Perceptions of Privacy and Security in the Post-Snowden Era.* Retrieved July 03, 2022 from https://www.pewresearch.org/internet/2014/11/12/public-privacy-perceptions/

[36] Moxie Marlinspike. 2013. *Advanced cryptographic ratcheting.* Retrieved July 11, 2022 from https://signal.org/blog/advanced-ratcheting/

[37] Moxie Marlinspike and Trevor Perrin. 2016. The x3dh key agreement protocol. *Open Whisper Systems* (2016).

[38] Matrix 2022. https://matrix.org/.

[39] Messages by Apple 2022. https://support.apple.com/explore/messages.

[40] Messages by Google 2022. https://messages.google.com/.

[41] Microsoft. 2018. *Skype Private Conversation. Technical white paper.* Retrieved July 21, 2022 from https://az705183.vo.msecnd.net/onlinesupportmedia/onlinesupport/media/skype/documents/skype-private-conversation-white-paper.pdf

[42] Emad Omara. 2020. *Google Duo End-to-End Encryption Overview.* Retrieved July 18, 2022 from https://www.gstatic.com/duo/papers/duo_e2ee.pdf

[43] Trevor Perrin and Moxie Marlinspike. 2016. The double ratchet algorithm. *GitHub wiki* (2016).

[44] Pidgin 2020. https://pidgin.im/.

[45] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. 2005. Secure Off-the-Record Messaging. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society (WPES '05)*. Association for Computing Machinery, New York, NY, USA, 81–89. https://doi.org/10.1145/1102199.1102216

[46] Dawin Schmidt. 2016. *A security and privacy audit of KakaoTalk's end-to-end encryption.* Master's thesis.

[47] Svenja Schröder, Markus Huber, David Wind, and Christoph Rottermanner. 2016. When SIGNAL hits the Fan: On the Usability and Security of State-of-the-Art Secure Mobile Messaging. In *European Workshop on Usable Security. IEEE.* 1–7. https://doi.org/10.14722/eurousec.2016.23012

[48] Maliheh Shirvanian and Nitesh Saxena. 2014. Wiretapping via Mimicry: Short Voice Imitation Man-in-the-Middle Attacks on Crypto Phones. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. Association for Computing Machinery, New York, NY, USA, 868–879. https://doi.org/10.1145/2660267.2660274

[49] Maliheh Shirvanian and Nitesh Saxena. 2015. On the Security and Usability of Crypto Phones. In *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC '15)*. Association for Computing Machinery, New York, NY, USA, 21–30. https://doi.org/10.1145/2818000.2818007

[50] Maliheh Shirvanian, Nitesh Saxena, and Jesvin James George. 2017. On the Pitfalls of End-to-End Encrypted Communications: A Study of Remote Key-Fingerprint Verification. In *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC '17)*. Association for Computing Machinery, New York, NY, USA, 499–511. https://doi.org/10.1145/3134600.3134610

[51] Signal 2022. https://signal.org/.

[52] Signal. 2022. *Technical information.* Retrieved July 21, 2022 from https://signal.org/docs/

[53] Silent Phone 2022. https://www.silentcircle.com/products-and-solutions/silent-phone/.

[54] Skype 2022. https://www.skype.com/en/.

[55] Ryan Stedman, Kayo Yoshida, and Ian Goldberg. 2008. A User Study of Off-the-Record Messaging. In *Proceedings of the 4th Symposium on Usable Privacy and Security (SOUPS '08)*. Association for Computing Machinery, New York, NY, USA, 95–104. https://doi.org/10.1145/1408664.1408678

[56] Paul Szoldra. 2016. *This is everything Edward Snowden revealed in one year of unprecedented top-secret leaks.* Retrieved July 03, 2022 from https://www.businessinsider.com/snowden-leaks-timeline-2016-9

[57] Joshua Tan, Lujo Bauer, Joseph Bonneau, Lorrie Faith Cranor, Jeremy Thomas, and Blase Ur. 2017. Can Unicorns Help Users Compare Crypto Key Fingerprints?. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 3787–3798. https://doi.org/10.1145/3025453.3025733

[58] Telegram 2022. https://telegram.org/.

[59] Telegram. 2022. *End-to-End Encryption, Secret Chats.* Retrieved August 04, 2022 from https://core.telegram.org/api/end-to-end

[60] Threema 2022. https://threema.ch/en.

[61] Threema. 2022. *Cryptography Whitepaper.* Retrieved December 24, 2022 from https://threema.ch/press-files/2_documentation/cryptography_whitepaper.pdf

[62] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. 2015. SoK: Secure Messaging. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 232–249. https://doi.org/10.1109/SP.2015.22
[63] Serge Vaudenay. 2005. Secure Communications over Insecure Channels Based on Short Authenticated Strings. In *Annual International Cryptology Conference*. Springer, 309–326. https://doi.org/10.1007/11535218_19
[64] Elham Vaziripour, Justin Wu, Mark O'Neill, Ray Clinton, Jordan Whitehead, Scott Heidbrink, Kent Seamons, and Daniel Zappala. 2017. Is That You, Alice? A Usability Study of the Authentication Ceremony of Secure Messaging Applications. In *Proceedings of the Thirteenth USENIX Conference on Usable Privacy and Security (SOUPS '17)*. USENIX Association, USA, 29–47.
[65] Sebastian R. Verschoor and Tanja Lange. 2016. (In-)Secure messaging with the Silent Circle instant messaging protocol. Cryptology ePrint Archive, Paper 2016/703. https://eprint.iacr.org/2016/703
[66] Viber 2022. https://www.viber.com/en/.
[67] Rakuten Viber. 2022. *Viber Encryption Overview*. Retrieved July 21, 2022 from https://www.viber.com/app/uploads/viber-encryption-overview.pdf
[68] WhatsApp. 2021. *WhatsApp Encryption Overview. Technical white paper*. Retrieved July 21, 2022 from https://scontent-iad3-1.xx.fbcdn.net/v/t39.8562-6/326130579_868561330899040_2694856431949694281_n.pdf?_nc_cat=107&ccb=1-7&_nc_sid=ae5e01&_nc_ohc=GiHwGuhmURAAX_u-okb&_nc_ht=scontent-iad3-1.xx&oh=00_AfDBUqimHHncuLDOeqED0EJOAeSSwksocCW-XdIkabxGPA&oe=63DDCC24
[69] WhatsApp 2022. https://www.whatsapp.com/.
[70] Wickr 2022. https://wickr.com/.
[71] Wire 2022. https://wire.com/en/.
[72] Ruishan Zhang, Xinyuan Wang, Ryan Farley, Xiaohui Yang, and Xuxian Jiang. 2009. On the Feasibility of Launching the Man-in-the-Middle Attacks on VoIP from Remote Attackers. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security (ASIACCS '09)*. Association for Computing Machinery, New York, NY, USA, 61–69. https://doi.org/10.1145/1533057.1533069
[73] Phil Zimmermann, Alan Johnston, and Jon Callas. 2011. ZRTP: Media path key agreement for unicast secure RTP. *Internet Engineering Task Force (IETF)* (2011), 2070–1721.
[74] Zoom 2022. https://zoom.us/.
[75] Zoom. 2022. *Zoom End-to-End Encryption Whitepaper*. Retrieved August 04, 2022 from https://github.com/zoom/zoom-e2e-whitepaper

A APPENDIX

A.1 Additional E2EE Applications Using the Signal Protocol

A.1.1 Google Meet It is an app developed by Google for video meetings and calls [20]. Google has upgraded the Google Duo app and merged it into the Google Meet app to include both video calling and meetings in one app. Therefore, the Google Meet app claims to provide an E2EE feature in one-to-one and group video calling using Google Duo's end-to-end encryption [42]. The Google Meet app uses the Signal protocol to implement the E2EE protocol. It uses E2EE mode by default for all voice and video messages and calls in all one-to-one and group conversations. In one-to-one calls, the Google Meet app uses WebRTC which supports E2EE for individual calls utilizing DTLS-SRTP. Datagram transport layer security (DTLS) is used to establish a secure connection between the two participants in the call, whereas SRTP is used to provide real-time and encrypted media streams. On the other hand, meetings in the Google Meet app are not end-to-end encrypted. Instead of E2EE, the Google Meet app uses cloud encryption for its meetings.

A.1.2 Messages by Google It is an app developed by Google to send messages using Short Message Service (SMS)/Multimedia Messaging Service (MMS) and chat with RCS [40]. Google provides RCS chat services via its Android Messages app. Recently, Google began rolling out the E2EE feature for RCS in the Android Messages app [3]. The Google Messages app uses the Signal protocol to implement the E2EE feature for RCS messages [19]. Google is only offering the E2EE feature on one-to-one chats by default if both participants in the conversation are using the Google Messages app. However, to utilize the E2EE feature in the Google Messages app, both the sender and the receiver must use the Google Messages app on their phone devices, have chat features enabled, and use data or Wi-Fi for RCS messages.

A.1.3 Skype It is an IM and VoIP app [54]. Using the Signal protocol, it implements the E2EE feature as an optional property [41]. Therefore, all Skype messages and calls are not E2EE by default. Users can protect their audio calls or messages by turning on an option called *Private Conversation*, which supports an E2EE scheme based on the Signal protocol. This option only supports the E2EE feature in chats and audio calls between two users. There is no E2EE protection for either a video call or a group scenario. In one-to-one audio calls, the Skype app uses an existing *Private Conversation* session between two users to generate an encryption key and initiate an E2EE audio call. After the E2EE audio call is set up, media packets are encrypted with SRTP using the previously generated encryption key.

A.2 Additional E2EE Applications Using Proprietary Protocols

A.2.1 Element It is an IM app and an independent communication system connected via Matrix [11]. The Element app is built on top of the Matrix protocol and uses the encryption implemented within the Matrix open standard [38]. In all one-to-one and group chats and calls, the Element app uses the Olm encryption library, which is based on the Double Ratchet protocol popularized by Signal, to implement the E2EE feature by default.

A.2.2 FaceTime It is a video and audio calling service developed by the Apple company [15]. The Apple company claims that the audio and video content of FaceTime calls is encrypted E2EE by default in all one-to-one and group scenarios. FaceTime uses the Apple Push Notification service (APNs) to establish the first connection point to the user's registered devices [25]. This first connection point is made via an Apple server infrastructure that transmits data packets between the users' registered devices. Users' registered devices verify their identity certificates and establish a shared secret for each session by using APNs and Session Traversal Utilities for NAT (STUN) messages through the relayed connection. By using SRTP, the shared secret is used to obtain session keys for the streamed media channels.

A.2.3 KakaoTalk It is an IM app created by the Kakao company in South Korea [27]. It allows users to implement E2EE functionality as an opt-in feature. Therefore, the KakaoTalk app does not enable the E2EE feature by default, and users must select an option called *Secret Chat* to chat in an E2EE manner. The E2EE feature was added to the KakaoTalk app on top of its LOCO Messaging Protocol [46]. The LOCO E2EE messaging protocol uses Transport Layer Security (TLS), a central public-key directory server, the AES encryption algorithm, and the RSA key-pair. When using the *Secret Chat* feature, all messages are E2EE in one-to-one and group chat rooms. However, audio and video calls are not available when using the *Secret Chat* feature.

A.2.4 LINE It is an IM app that is popular in East Asia [32]. It implements E2EE functionality by default under a security feature called *Letter Sealing* [8]. Therefore, the *Letter Sealing* feature is turned on by default for all text messages, location information, voice calls, and video calls between two users in one-to-one scenarios. However, only text messages and location information are E2EE by default on group chats. The LINE app does not support E2EE voice or video calls in group scenarios. To implement the E2EE feature, the LINE app uses the elliptic curve Diffie-Hellman (ECDH) protocol over Curve25519 and AES-256 in GCM mode. However, in one-to-one voice and video calls, the LINE app utilizes the curve secp256r1 for the VoIP encryption protocol, AES for symmetric encryption, and HKDF for deriving symmetric keys.

A.2.5 Linphone It is an audio and video calling app that supports IM [33]. It implements E2EE functionality as an opt-in feature for one-to-one and group messages, as well as for audio and video calls. In order to implement E2EE in one-to-one and group IM features, the Linphone app uses its own E2EE protocol called Linphone instant message encryption (LIME) [34]. This LIME protocol is inspired by the Signal protocol, allowing users to send and receive messages privately and asynchronously. On the other hand, the Linphone app implements the E2EE feature for one-to-one audio and video calls using ZRTP and SRTP-DTLS, which are compatible with WebRTC. However, the E2EE feature is not available for voice or video calls in group scenarios.

A.2.6 Messages by Apple It is an IM app developed by the Apple company to send messages with iMessage and SMS/MMS [39]. The Apple Messages app utilizes the iMessage protocol to implement the E2EE feature by default in all one-to-one and group scenarios [24]. After switching on iMessage on a device, the device generates encryption and signing pairs of keys for use with the service. The Apple iMessage protocol uses an encryption RSA 1,280-bit key and an encryption EC 256-bit key on the NIST P-256 curve for the encryption, whereas with the elliptic curve digital signature algorithm (ECDSA), 256-bit signing keys are used for the signatures. It also uses Apple Identity Service (IDS) to store public keys and maintain the mapping between them and the user's phone number or email address, along with the device's APNs address, whereas private keys are saved in the device's keychain. The APNs are then used to deliver the encrypted message text, the encrypted message key, and the sender's digital signature.

A.2.7 Silent Phone It is an IM and VoIP app developed by Silent Circle [53]. It claims that all messages and calls are E2EE by default in all one-to-one and group scenarios. It uses its own protocol, based on the Signal protocol, to implement E2EE in IM features [65]. On the other hand, ZRTP is used to implement the E2EE feature in audio and video calls [73]. ZRTP uses Diffie-Hellman key exchange and SRTP to establish a shared session key and encrypt data.

A.2.8 Threema It is an IM app that also allows users to make voice and video calls [60]. It claims that all messages and calls are E2EE by default in all one-to-one and group scenarios. It uses its own protocol to implement the E2EE feature in messages and calls [61]. When the Threema application is installed on a user's phone device, it generates, for each user, a unique asymmetric key pair consisting of a public key and a private key based on elliptic curve cryptography. The Threema app uses the ECDH protocol to establish a shared secret. It then uses the XSalsa20 stream cipher to encrypt the plaintext, whereas a message authentication code (MAC) is computed by Poly1305-AES. In Threema calls, WebRTC is used to establish a secure peer-to-peer (P2P) connection. The audio stream is encrypted with the SRTP protocol, and the key exchange is done with the DTLS-SRTP protocol.

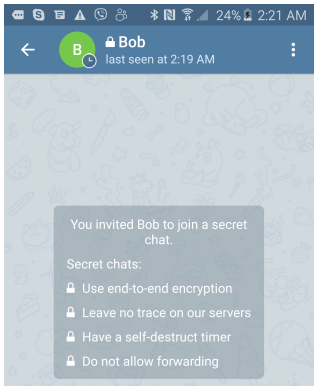
A.2.9 Wickr Wickr [70] has developed the Wickr Me app and the Wickr Pro app for individual and business uses, respectively. To provide the E2EE feature, Wickr uses its own protocol called the Wickr secure messaging protocol, which is based on standard cryptographic primitives [23]. It uses ECDH key exchange with P521 key pairs, ECDSA with P521 key pairs, AES 256 in GCM mode, and KDF. All messages and audio/video calls are E2EE by default in all one-to-one and group scenarios. Once Wickr generates keys for users and their first devices, it stores public keys and root identifiers on Wickr servers.

A.2.10 Wire It is an IM and VoIP app created by the Wire Swiss GmbH company [71]. It claims that all messages and calls are E2EE by default in all one-to-one and group scenarios. Wire uses the Proteus protocol to implement the E2EE feature, which copies some features of the Signal protocol [18]. However, the Proteus protocol has been customized as an independent implementation of the Signal protocol. The Proteus protocol uses the following cryptographic primitives: the ChaCha20 stream cipher, HMAC-SHA256 as MAC, ECDH key exchange, and HKDF for key derivation. In Wire calls, the call media session is encrypted by the SRTP protocol, whereas the DTLS handshake is used to negotiate the SRTP encryption algorithm, keys, and parameters. Once a client generates the key material, the client uploads pre-keys bundled with its public identity key to a Wire server, which can be used by other clients to asynchronously initiate an E2EE conversation.

A.3 Additional Tables and Figures

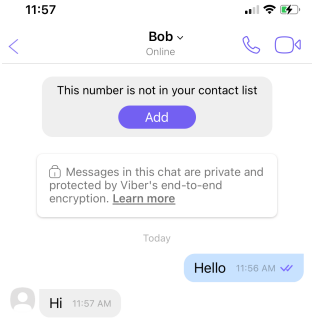
Table 3: End-to-End Encrypted Applications Rating and Reviews on Google Play Store

Application	Installs on Google Play	Rating	Reviews
WhatsApp	5,000,000,000+	4.3	172,000,000
Facebook Messenger	5,000,000,000+	4.1	85,900,000
Google Meet	5,000,000,000+	4.6	9,810,000
Viber	1,000,000,000+	4.5	16,200,000
Telegram	1,000,000,000+	4.3	11,800,000
Skype	1,000,000,000+	4.1	11,500,000
Messages by Google	1,000,000,000+	4.2	9,150,000
LINE	500,000,000+	4.1	13,700,000
Zoom	500,000,000+	4.2	3,980,000
KakaoTalk	100,000,000+	4.3	3,160,000
Signal	100,000,000+	4.4	2,190,000
Wickr Me	10,000,000+	4.8	89,000
Threema	1,000,000+	4.1	70,800
Wire	1,000,000+	2.9	35,100
Linphone	500,000+	3.8	5,350
Element	500,000+	4.1	4,570
Silent Phone	500,000+	3.8	1,830

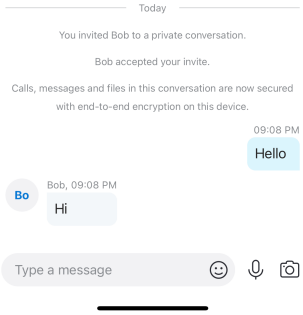


(a) Telegram

(b) WhatsApp

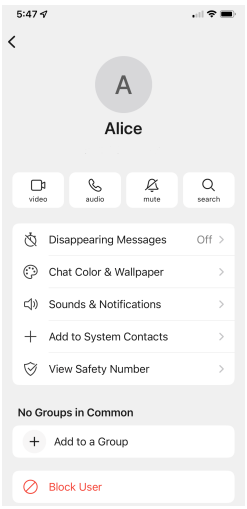


(c) Viber

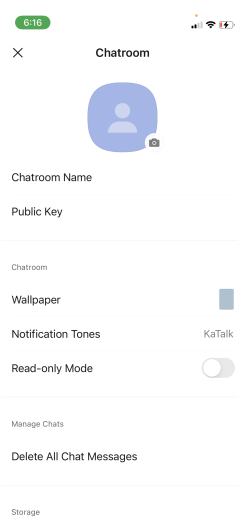


(d) Skype

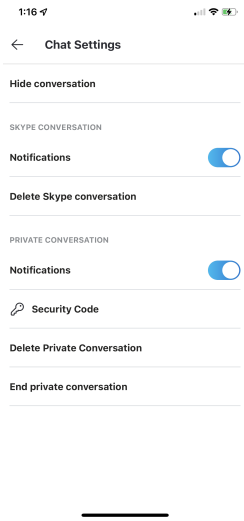
Figure 3: Alerting users that the opportunistic E2EE mode is turned on and their messages are end-to-end encrypted by using different indicators, such as special notification messages and lock icons.



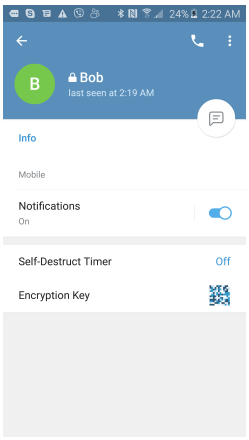
(a) Signal



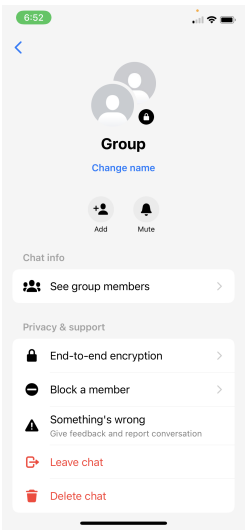
(b) KakaoTalk



(c) Skype



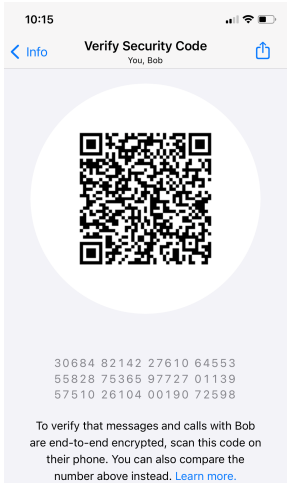
(d) Telegram



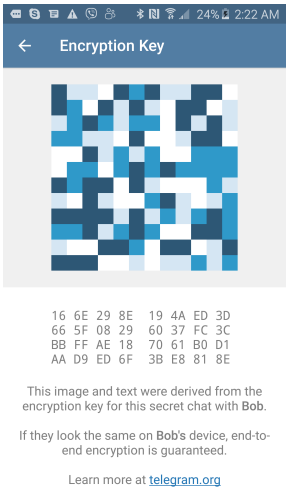
(e) Facebook Messenger

Figure 4: Some E2EE applications refer to the authentication ceremony using the terminology shown above.

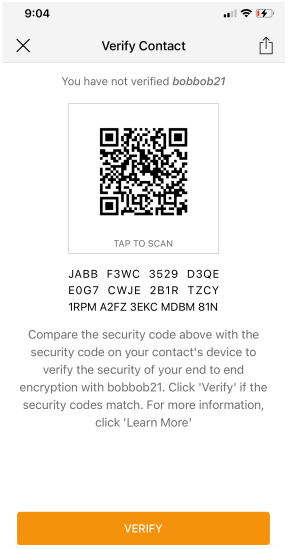
(a) Signal



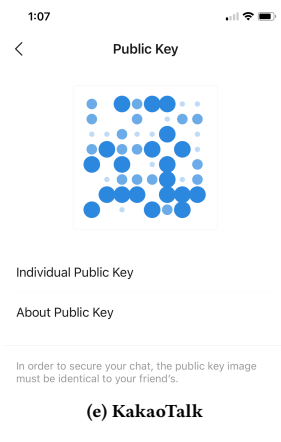
(b) WhatsApp



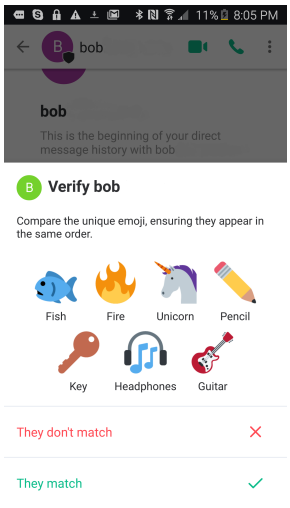
(c) Telegram



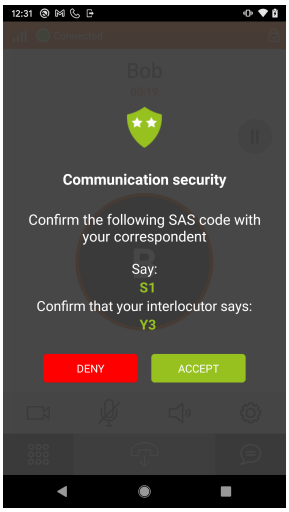
(d) Wickr



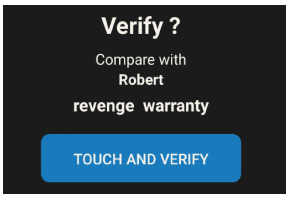
(e) KakaoTalk



(f) Element



(g) Linphone



(h) Silent Phone

Figure 5: Fingerprint representations in E2EE applications