Send and Pretend: Exploiting Transcript Consistency Issues in End-to-End Encrypted Group Chats

Gabriel K. Gegenhuber[1], Moritz Grefner[2], Maximilian Günther[3], Matthäus Wininger[2], David Schmidt[2,4,5], and Aljosha Judmayer[2]

[1]Interdisciplinary Transformation University (IT:U), [2]University of Vienna, Faculty of Computer Science, [3]SBA Research, [4]UniVie Doctoral School Computer Science, [5]CDL AsTra

Abstract

End-to-end encrypted (E2EE) messaging apps are widely praised for their security and thus also used for sensitive coordination in group chats (e.g., by political decision makers). After Threema and WhatsApp, also Signal and iMessage have recently introduced polls to aid agreement processes in groups. This implicitly sets the expectation that all participants see the same outcome and thus have the same view of the conversation. This property is commonly referred to as *transcript consistency* (TC). In this work, we demonstrate that today's major E2EE messengers do not guarantee *any* form of TC for group chats, allowing a malicious group member to selectively omit, reorder, or present altered content to different recipients without triggering warnings in their user interface. We systematically investigate the extent of the problem under a *malicious-participant* threat model that targets the integrity of the shared transcript, or inconsistent delivery across a user's linked devices. We identify multiple equivocation vectors that range from protocol fallback paths to deliberate use of pairwise delivery channels within groups. We demonstrate concrete exploitation scenarios such as social engineering, evading moderation, and, in particular, *rigging polls*. Beyond these cross-service design issues, we also uncover implementation-specific behaviors with privacy implications (e.g., device OS fingerprinting). Finally, we contextualize our findings within prior transcript-consistency research and outline practical low-overhead mitigations and UI signaling strategies that can be integrated into state-of-the-art E2EE group protocols.

1 Introduction

End-to-end encrypted (E2EE) instant messaging applications such as Signal, WhatsApp, iMessage, and Threema are used daily by a substantial fraction of the global population [6, 8, 9, 26]. A precise assessment of the security guarantees provided by these apps is therefore critical for private individuals, as well as government officials, particularly in the United States [33, 37] and the European Union [34]. In these regions, encrypted messaging apps are reportedly used for sensitive governmental communication and decision-making, as illustrated by a recent incident in which a journalist was erroneously added to a group chat of government officials, including the US Secretary of War, Pete Hegseth [33].

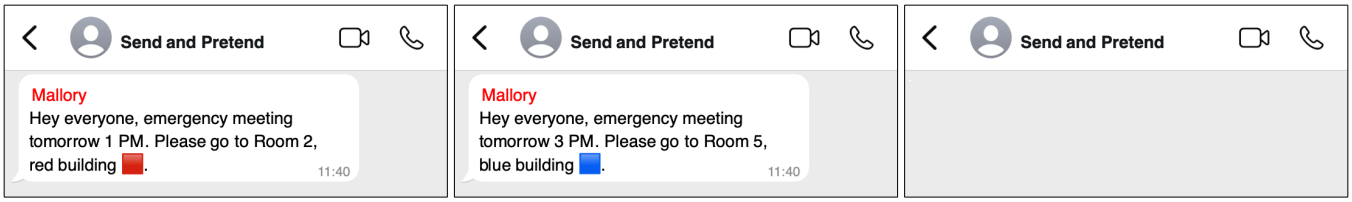
A key security property in this context is that all members of a group conversation eventually converge to the same view of the message transcript. This property is commonly referred to as *transcript consistency (TC)*[1]. In the evolution of E2EE instant messaging, TC has been proposed as a desirable property for group chats early on [32] and was also discussed in the design of the TextSecure protocol [39], which later evolved into Signal and shaped the design of later messengers such as WhatsApp.

Prior work on secure group messaging primarily considered adversaries that control the server and/or the network, and addresses threats such as message dropping, reordering, or traffic analysis by outsiders [19, 48, 53]. Early proposals already discussed retrospective detection of transcript inconsistencies, for example, in Off-the-Record Messaging (OTR) and its first group extensions [32, 39, 67].

Despite these early discussions of TC as a requirement for E2EE messaging systems, the extent to which modern messengers actually provide TC remains underexplored. This gap becomes more pronounced as messengers increasingly integrate interactive group features such as polls [15, 31, 38, 41, 63]. Unlike ordinary messages, polls implicitly assume not only a shared transcript but also a common interpretation of the shared state, thereby amplifying the impact of even subtle inconsistencies. These features, therefore, require a consistent view of the outcome and motivate us to study: *RQ: What is the current state of TC in E2EE messaging?*

In contrast to prior work, we explicitly consider a malicious client, already participating in the group, that aims to manipulate the view on the conversation of selected partici-

[1]Comparable security properties were discussed in academia under different names, e.g., *consensus* [32], *speaker consistency* and *global transcript* [67], *causality preservation* [20, 67], *transcript agreement* [21], *transcript consistency* [36, 39].



(a) Alice (original message) (b) Bob (victim) (c) Carol (victim)

Figure 1: Side-by-side views of a group chat from each participant's perspective. Each panel represents the local transcript on a participant's device. While Alice sees the original message, Bob receives a message with an exchanged meeting time, room, and building, and Carol was skipped entirely. At present, this attack is possible across all prevalent end-to-end encrypted (E2EE) messengers without being detected by the server or triggering warnings on the receiving client.

pants. This assumption is especially relevant in large groups, where honest behavior from all participants cannot be taken for granted[2]. In such settings, an attacker could, for example, distribute malicious links to a subset of participants while sending benign links to others, thereby evading moderation. More aggressively, attackers could steer opinions or inflict social harm by selectively excluding participants or by providing them with incorrect information, for example, about scheduled meetings, as demonstrated in Figure 1. The widespread use of instant messengers and group chats by government officials and world leaders further amplifies the severity of this threat.

In this paper, we show that, contrary to common assumptions, current prevalent instant messengers provide no form of TC for group chats. We demonstrate that WhatsApp [68], Signal [52], iMessage [10], and Threema [61] allow malicious group members to selectively omit, reorder, or tailor messages for individual participants without detection by the messaging server or the recipient. These weaknesses extend to higher-level features, enabling attackers to manipulate polls by presenting different outcomes to different participants, biasing results, or overwriting polls entirely. Such attacks directly target the integrity of the shared group transcript.

To the best of our knowledge, we are the first to expose the full extent of this problem, in particular for targeted messages and interactive features such as polls, and to show that it represents a fundamental design flaw affecting all prevalent E2EE instant messenger designs. Given this negative result, we analyze the underlying causes, survey prior TC research, map the attack surface, and propose practical changes that can improve TC in prevalent secure messaging applications.

In detail, we make the following contributions:

- We show that WhatsApp, Signal, iMessage, and Threema, in their current versions, do not provide *any* form of TC in group chats, allowing malicious participants to omit, reorder, or modify messages for individual group members without detection.
- We demonstrate that the poll feature of all analyzed messengers is also affected, allowing poll creators and voters to manipulate the outcome. This ranges from honest group participants seeing different outcomes, to manipulating the outcome towards a desired decision or overwriting the poll result with a desired outcome.
- We uncover implementation-specific issues with implications for TC and user privacy.
- We discuss mitigations that require minimal changes to the Sender Key protocol used by Signal and WhatsApp, enabling the detection of potential attacks and the ability to warn users.

[2]The Sender Key protocol, as used by WhatsApp and Signal, supports groups of around 1,000 participants [54, 60].

2 Background

For group messaging, E2EE messengers (in particular, those based on the Signal protocol) propose two different paradigms for sending and distributing messages to group participants.

2.1 Pairwise Messaging (Client Fan-Out)

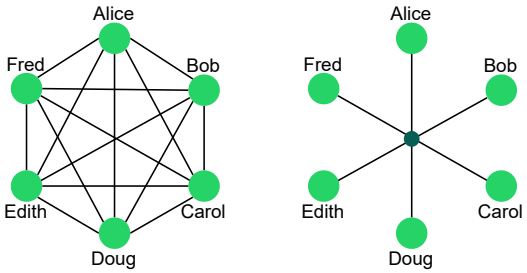
In the client fan-out model, group messages are handled similarly to one-to-one (1:1) conversations. Figure 2a illustrates that the sending client encrypts the same plaintext message separately for each group participant using the respective session keys. Each encrypted message, or *envelope*, is uploaded to the server, which relays it to the intended recipients without knowing its content. This approach preserves strict E2EE semantics but incurs significant computational and bandwidth overhead for the sender in large groups, as the client must perform multiple encryptions and uploads per message.

2.2 Sender Key Protocol (Server Fan-Out)

The *Sender Key Protocol* [11, 12, 69] optimizes the message complexity of group messaging by introducing a single (ephemeral) shared symmetric key for all group members. Each participant independently chooses such a key and distributes it to all other members once, typically when sending

(a) Client Fan-Out (b) Server Fan-Out

Figure 2: Comparison of client-side and server-side fan-out message delivery models.



(a) Pairwise E2EE (b) Sender Key

Figure 3: Costs of different group messaging schemes.

their first message, using the same pairwise E2EE channels as in client fan-out. Once the shared key has been distributed, the sender encrypts each message once using their individual Sender Key and uploads a single (signed) ciphertext to the server, rather than performing multiple encryptions. The server then distributes this ciphertext to all group members via a broadcast mechanism (i.e., *server fan-out*), as illustrated in Figure 2b. In large groups, this design reduces the sender's workload and message complexity, as well as network usage, shifting scalability concerns to the server, while maintaining confidentiality and authenticity through the key distribution mechanism built atop the Signal Protocol.

Current State. Both WhatsApp and Signal are based on the Signal protocol. According to the WhatsApp Security Whitepaper [69], WhatsApp has employed the Sender Key protocol for group messaging since the rollout of E2EE in 2016. In contrast, Signal relied on the client fan-out approach until August 2021, when it also transitioned to the more efficient Sender Key variant [56]. In late 2020, Signal additionally introduced the new private group system (also referred to as *groups V2* or *zkgroups*) [19, 35, 53], which improves privacy by concealing group membership and other identifying information from the service provider.

In contrast, the other services investigated in this work (iMessage, Threema) currently use a pairwise client fan-out approach for E2EE group communication.

Cost Comparison. In client fan-out, each message is encrypted separately for every recipient, resulting in $O(n)$ encryption operations and uploads per message. Server fan-out reduces this to $O(1)$ by encrypting the message once and relying on the server for distribution. This improvement applies to all ongoing group messages. While the initial distribution of the Sender Key relies on pairwise E2EE channels and incurs a one-time cost, this overhead is outweighed by the efficiency gains of subsequent group messages.

Note that the examples in Figure 3 assume that each participant owns only a single device. In practice, users often have multiple linked companion devices (e.g., tablets, laptops, or desktops). Under the client fan-out model with per-device sessions, the sender must encrypt and upload a distinct ciphertext for every device belonging to every participant, further amplifying computational and bandwidth costs and – as we show in this work – potential TC issues. An exception is Threema, which relies on a single account-wide key that it shares across devices in multi-device settings [64].

2.3 Transcript Consistency

Properties offering some form of *transcript consistency* (TC) were discussed under many names, e.g., *consensus* [32], *speaker consistency* and *global transcript* [67], *causality preservation* [20, 67], *transcript agreement* [21], *transcript consistency* [36, 39]. We group different terms and flavors into four TC variants. These variants range from the weakest to the strongest consistency guarantees for a message transcript. Note that we intentionally do not specify when or how often TC has to be checked. As none of the analyzed E2EE messaging services provides any form of TC, we defer a detailed contextualization and discussion to the Appendix A.1:

- *set transcript consistency* (STC): The participants of the group eventually agree on the same (unordered) set of messages belonging to the conversation.
- *participant transcript consistency* (PTC): The participants of the group eventually agree on the ordered sequence of messages coming from each participant individually. Messages from different participants are not relatable to each other, except that they belong to the same conversation.

Service	Group Messaging	Max. Size	User Estimate	Open-Source Implementation
WhatsApp	Sender Key	1,024	3.5 B	whatsmeow [5]
iMessage	Pairwise E2EE	32	1.3 B	rustpush [2]
Signal	Sender Key	1,001	70 M	Signal-Desktop [3]
Threema	Pairwise E2EE	256	12 M	threema-android [4]

Table 1: Overview of the analyzed messaging services, their group messaging protocols, maximum group sizes [1, 42, 54, 60, 62, 65], estimated user bases [6, 8, 9, 26], and the open-source implementations we used as bases.

- *causal transcript consistency* (CTC): The participants of the group eventually agree on the same (unordered) set of messages belonging to the conversation, as well as a strict partial order of messages, i.e., not every pair of messages must be comparable (e.g., messages can be at the same height).
- *total transcript consistency* (TTC): The participants of the group eventually agree on the same ordered set of messages belonging to the conversation, s.t. for any two messages m_1, m_2 that appear in the transcript of an honest participant, if m_1 precedes m_2 for some honest participant, then m_1 precedes m_2 for every honest participant.

3 Threat Model

We assume an asynchronous communication model, which is common in instant messaging environments [39, 55] and distributed systems literature [18], i.e., there is no guarantee that messages are delivered in order, or within a known upper time bound. In all analyzed instant messengers, all messages are delivered via a central server infrastructure operated by the respective instant messaging service. In other words, all messages must pass through a central server. The trust that is put into the server varies across the analyzed services. While Signal aims to minimize the information available at the server [19, 35], WhatsApp servers hold plain text information regarding group memberships[3].

Adversary. The adversary is a registered user of the messaging system and one participant of the targeted group chat[4]. The attacker fully controls their own devices and local protocol state, and thus may arbitrarily deviate from the prescribed protocol rules. The adversary is not capable of violating the security assumptions of cryptographic primitives, does not control or collude with the server, and does not have access to other users' long-term secrets or devices. In practice, especially in large groups, participants can be added via invites or links and may not be uniformly trusted, which makes malicious or compromised members a realistic threat.

Adversarial Goals. The adversary aims to violate TC within a group chat while preserving E2EE semantics. In particular, the adversary seeks to achieve the following without being detected by the server or through alerts or visible clues in the standard user interfaces offered for the respective chat software:

- (G1) **Inconsistent delivery to participants:** Cause different participants in a group conversation to receive different messages, or selectively prevent some participants from receiving a message, so that honest users observe inconsistent views of the group transcript.
- (G2) **Inconsistent delivery to devices:** Deliver different messages to different devices belonging to the same honest user in a direct or group conversation, or drop messages for specific devices, causing the user's devices to diverge in their view of the conversation history.

The motivation for such attacks ranges from low-impact pranks to high-impact scenarios such as targeted device exploitation ((G2)), as well as the deliberate disruption of group coordination by inducing disagreement and confusion ((G1)), which is particularly consequential in large groups and high-stakes decision-making contexts.

4 Methodology

In this section, we describe our messenger selection, experimental setup, and evaluation methodology. In addition to analyzing the message transmission paths used for group communication (e.g., pairwise E2EE and the Sender Key protocol), we examine the robustness of each service against anomalies during key distribution and analyze the recovery and retransmission behavior triggered by encryption failures. While this section investigates the general equivocation vectors and protocol weaknesses, Section 5 evaluates exploitation in practice and demonstrates that an attacker can exploit these vectors not only for standard text messages but also for rich content such as location sharing or polls.

4.1 Messenger Selection

We aim to include relevant instant messaging services that support E2EE in different variants (e.g., group messaging model, their target user bases, and free vs. paid usage). The Signal protocol is widely regarded as the de facto gold standard for secure E2EE messaging, and the Signal application is among the most popular messengers within the security community. In addition, WhatsApp has approximately 3.5

[3] We revisit this aspect in more detail when discussing possible mitigation strategies in Section 6.

[4] All described attacks work with at least one adversarial participant within a group.

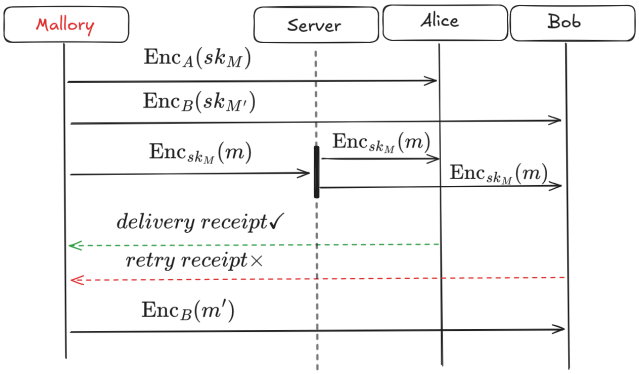


Figure 4: Group communication with sender key messages (WhatsApp, Signal), where a decryption failure triggers retransmission of a message via direct E2EE communication.

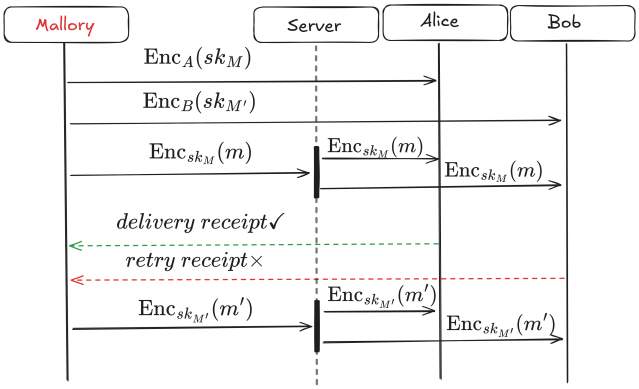


Figure 5: Message equivocation using sender key sends only (WhatsApp, Signal). We test whether multiple broadcasts that result in contradictory or repeated message deliveries constitute a practical attack vector, or whether they trigger UI warnings or suspicion at receiving clients.

billion active users [26] and adopted the Signal protocol in 2016. We thus start our evaluation with these two applications and Signal's Sender Key protocol.

Beyond the Signal protocol, other instant messengers have adopted E2EE as well. For example, iMessage is preinstalled on iPhones and other Apple devices, is among the most popular instant messengers in the US, and accounts for more than one billion users globally. Finally, we include Threema as an additional messenger with a strong focus on security and privacy; it is the only paid app among the services we consider and is particularly popular in Europe.

In Table 1, we provide an overview of the messenger applications we study, along with their estimated user bases and current maximum group sizes.

4.2 Experimental Setup and Evaluation

To reflect the adversarial capabilities described in Section 3 (i.e., control over device state, their encryption keys, and the ability to deviate from the protocol, while still sending syntactically valid messages), we use both official reference clients (Signal, Threema) and unofficial client implementations (WhatsApp, iMessage). We apply targeted patches that enable controlled deviations from the protocol behavior. While open or unofficial clients simplify testing, an attacker could also reverse engineer the official clients and perform the experiments via instrumentation, e.g., using Frida [45]. We assume the attacker to be an existing member of the group, while one or more other group participants act as victims. On the victim side, we evaluate clients across major platforms, including Android, iOS, Windows, macOS, Linux, and web-based clients.

To assess the robustness of the target applications with respect to message delivery and TC, we evaluate their behavior under a range of error conditions and atypical delivery paths, including cases where protocol assumptions are violated. For each messenger, we first analyze the default group message flow. For WhatsApp and Signal, we start with Sender Key-based group messaging and then evaluate fallback behavior triggered by decryption errors. Next, we examine pairwise E2EE transmission, which is the default for iMessage and Threema, and test whether similar pairwise delivery can also be used in WhatsApp and Signal to send group messages. While the previous evaluations focus on equivocation via pairwise E2EE channels, we finally assess how robust Sender Key-based message broadcast (i.e., on WhatsApp and Signal) is against equivocation and whether contradictory or repeated messages via the broadcast channel trigger user-visible alerts.

4.2.1 Sender Key with Pairwise E2EE Fallback

Both WhatsApp and Signal employ the Sender Key protocol for group messaging. Under this protocol, each group message is encrypted using a symmetric *sender key* selected by the message sender. To enable other participants to decrypt future Sender Key messages, the sender first distributes the sender key to all group members via pairwise E2EE channels.

Figure 4 demonstrates that this design permits a malicious sender to distribute different sender keys to participants, which would result in decryption failures for one or more recipients. We investigate the resulting failure and recovery behavior and show that, in practice, failures of the Sender Key protocol are often handled by falling back to pairwise E2EE communication. While this behavior is the intended recovery path, the resulting pairwise sessions can subsequently be used to transmit different messages to different participants.

4.2.2 Pairwise E2EE Messages

For iMessage and Threema, this is the regular (and only) path for group communication, being highly vulnerable to message equivocation by design, as demonstrated in Figure 6.

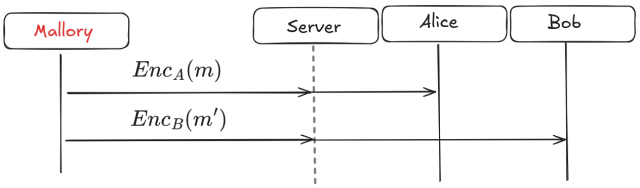


Figure 6: Group communication based on pairwise E2EE is inherently prone to equivocation attacks. While some messengers still use it as the default transmission path in groups (iMessage, Threema), we evaluate whether it can nevertheless be leveraged for group communication on the messengers supporting the Sender Key protocol (WhatsApp, Signal).

Signal adopted the Sender Key protocol for group messaging in 2021 and relied on pairwise E2EE group messages for much of its lifetime. Both Signal and WhatsApp employ pairwise E2EE transmission as a recovery mechanism for group messages when Sender Key sends fail. Because encryption failures are sometimes visible to users in the client UI, we further investigate whether a malicious client can intentionally bypass the Sender Key protocol and instead transmit group messages exclusively via the pairwise E2EE path as the primary delivery mechanism.

4.2.3 Sender Key Messages Only

The previous two techniques demonstrate how a malicious sender can create divergent views of the message transcript by deviating from the Sender Key protocol (if applicable) and using directed, pairwise E2EE channels to transmit equivocated messages. In this final scenario, shown in Figure 5, we examine whether an attacker can violate TC while using only Sender Key messages, without relying on transmitting the equivocated messages via pairwise E2EE sessions. Specifically, we test how clients respond when an attacker broadcasts multiple versions of the same message, each encrypted for different clients, using the server fan-out mechanism.

5 Results and Exploitation

In this section, we first present transcript inconsistencies under the analyzed protocols. We then discuss practical attack scenarios and highlight implementation-specific issues.

5.1 Equivocation Vectors

Table 2 shows that all services are vulnerable to message equivocation in the group setting (G1). Threema is not vulnerable to the device-specific equivocation case (G2) because of its distinctive multi-device architecture: only a single device of the receiving account fetches the message from the server, then re-encrypts and reflects it to all remaining devices via a mediator server. This is a consequence of

	One-to-One (1:1)				Group Chats			
G1	–	–	–	–	✓	✓	✓	✓
G2	✓	✓	✓	✗	✓	✓	✓	✗

G1 Participant inconsistency G2 Device inconsistency
– not applicable ✓ vulnerable ✗ not vulnerable

Table 2: Overview of messengers vulnerable to G1, which enables participant inconsistency, and G2, which enables device inconsistency, in 1:1 communication and group chats.

Threema's multi-device implementation rather than stronger TC defenses, and it comes with a serious trade-off, since Threema's multi-device protocol implementation does not support forward secrecy [64].

5.1.1 Sender Key with Pairwise E2EE Fallback

Both WhatsApp and Signal implement a recovery mechanism for Sender Key group messages in which recipients can request a retransmission after a decryption failure. In response, the sender retransmits the message to the requesting recipient via the pairwise 1:1 E2EE session, which provides an opportunity for equivocation by delivering recipient-specific content (cf. Figure 4). In Signal, unsuccessful message decryptions fail silently, thus creating no barriers for the attacker. In WhatsApp, decryption errors are shown to the user, as demonstrated in Figure 7. While this could potentially raise suspicion by the user and thereby limit exploitability, the error bubble disappears immediately after the attacker retransmits the equivocated message, thus only being visible for a fraction of a second. More severely, WhatsApp allows the message sender to set a *decrypt-fail* attribute to *hide*, effectively suppressing this error bubble entirely on the receiver side, yielding the same behavior and exploitability as in Signal. To summarize, this vector enables sending different messages to arbitrary participants and their devices, achieving both goals G1 and G2 in WhatsApp and Signal.

5.1.2 Pairwise E2EE Messages

While Sender Key messages aim to construct a shared broadcast channel, messaging based on pairwise E2EE sessions is inherently vulnerable to equivocation. We confirm this vector for iMessage and Threema, both of which exclusively rely on pairwise channels for group communication.

Although WhatsApp and Signal could, in principle, enforce Sender Key delivery for all group messages except for sender key distributions, both messengers still accept group messages sent via pairwise channels. In Signal, this requires no special effort, as messages can simply be sent via the

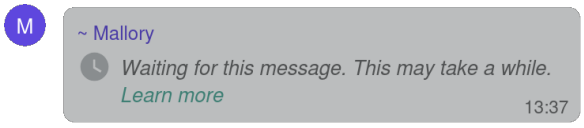


Figure 7: By default, WhatsApp shows a decryption error message bubble upon unsuccessful message decryption. However, the error persists only until the sender retransmits the message. Moreover, WhatsApp lets the sender suppress these warnings on the receiver side, again leading to silent failure and leaving no trace of the attack in the victim's UI.

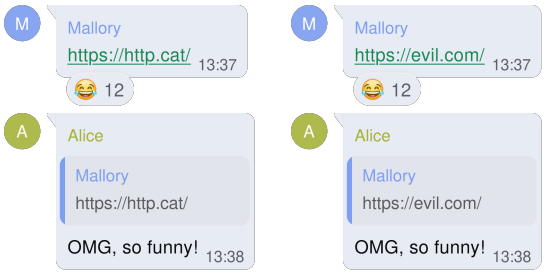
legacy group messaging path used prior to 2021 (cf. Section 2). In WhatsApp, pairwise group messages are rejected by default, but we found that they are accepted when disguised as retransmissions. This holds even if the receiving client never requested a retransmission, effectively enabling arbitrary equivocation via pairwise E2EE channels. In summary, this equivocation vector is viable across all studied messengers and does not rely on decryption failures that could otherwise serve as a detection signal. In addition to (G1), if the protocol uses per-device sessions, which is the case for all evaluated services except Threema, targeting specific devices ((G2)) is also possible.

5.1.3 Sender Key Messages Only

Instead of responding to a victim's retry request with a pairwise retransmission, both WhatsApp and Signal allow a malicious sender to ignore the retry receipt and simply broadcast the equivocated message via the Sender Key approach. Clients who have already successfully received the corresponding message (matched by its unique message ID) on the first attempt simply ignore the subsequent transmission, even if it would cause a decryption failure or contain contradicting content. Again, no warnings were displayed to the user.

On WhatsApp, the server tracks all members of a group and always broadcasts a message to all participants. To improve privacy, Signal does not store this kind of data in plaintext and thus allows the message sender to define who is part of the group and should be included in the message broadcast. This creates an additional vector for creating individual transmissions for different group participants, effectively allowing the attacker to establish individual channels to each participant using Sender Key group messages[5]. To summarize, via multiple Sender Key message broadcasts, the attacker is able to achieve both goals ((G1), (G2)) for all evaluated and applicable applications.

[5]This differs from pairwise delivery because 1:1 messages and Sender Key group messages are generally handled via different server endpoints. Additionally, the Signal server even allows Sender Key messages (which are multi-recipient sealed sender V2 messages) through the 1:1 endpoint [58].



(a) Normal member view. (b) Victim's view.

Figure 8: Split-view attack in which the attacker (Mallory) sends a benign link to most group members and exploits their reactions to increase trust in a spoofed link sent to the victim.

We show that:
- ✓✓ pairwise E2EE messaging is vulnerable to transcript inconsistency attacks per design;
- ✓✓ current Sender Key implementations are vulnerable due to fallback and retransmission paths;
- ✓✓ thereby, all evaluated messenger services are vulnerable to message equivocation, and none provide server-side or UI detection mechanisms, enabling practical, stealthy exploitation.

Takeaway ✓✓

5.2 Exploitation Examples

After demonstrating the possibility of sending inconsistent messages within a group, we show the impact of the previously described equivocation vectors by providing examples that could occur in real-world scenarios. While we earlier showed that all studied messengers are vulnerable to transcript inconsistencies, polls are particularly critical because their results are rendered as authoritative application state. In addition to design-level vectors, we therefore analyze implementation-specific behaviors and bugs. Although not required to violate TC, these issues are practically relevant because they amplify TC failures by reducing attacker effort and suppressing UI warnings. Alongside this paper, we release videos demonstrating these attacks against unmodified, real-world client applications and devices for all investigated messengers[6].

5.2.1 Social Engineering and Manipulation Tactics

Attackers can exploit inconsistencies in group chats to manipulate group members through social engineering and deliberate deception. Figure 1 illustrates how a malicious group member can sabotage group coordination in the context of a scheduled meeting. Victims may receive no message at all, or receive entirely different meeting details. The technique is not

[6]Attacks were executed exclusively against our own accounts and devices.

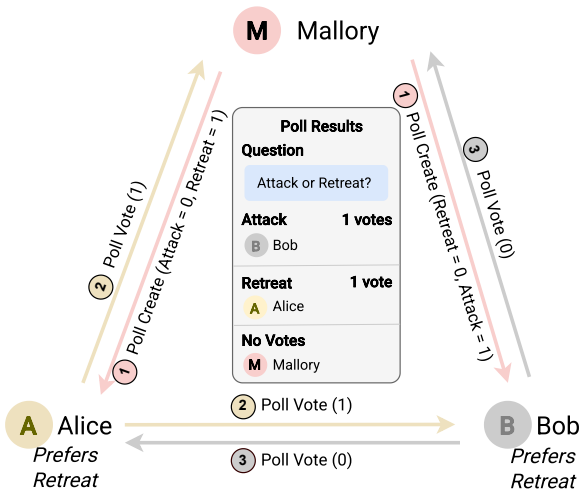
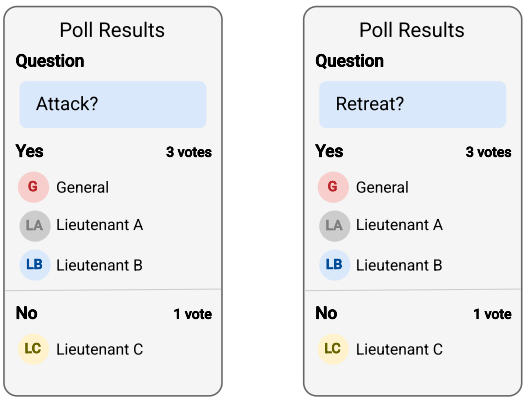


Figure 9: Mallory sends a manipulated poll to Bob (❶). For Mallory and Alice, the option *Attack* has index 0 and *Retreat* has index 1. For Bob, the indices are swapped. Although Bob, like Alice (❷), prefers *Retreat* (❸), his vote appears as *Attack* to the other participants. Due to the tie in votes, Mallory can now decide whether to attack or retreat.



(a) Members were lured into voting for the attack option.

(b) View of victims receiving an inverted question.

Figure 10: A malicious poll creator can send different questions to parts of the group, tricking them into voting for unfavored options and leading to manipulated results. Attacking polls by sending out inverted questions is also possible on WhatsApp, since voting hashes only consider the voting option text. We provided videos of this attack in our artifact.

limited to text-based messages; for example, on WhatsApp, attackers can apply it to the location-sharing functionality. To further increase deniability, attackers can leverage short message timers to automatically delete messages, removing evidence and enabling plausible deniability after the incident. In addition, such manipulation can undermine the victim's trust in their own perception and memory, which may cause psychological distress and negatively affect mental well-being.

5.2.2 Evading Moderation and Split View Attacks

Attackers could also exploit transcript inconsistencies for phishing campaigns and to evade moderation. Similar to Figure 1, an attacker can send messages with different content to group participants, for example, benign-looking messages to group administrators and moderators while simultaneously delivering scam messages to other group members.

A related scenario combines phishing links with legitimate URLs. Since replies and reactions (e.g., emoji) are linked to messages by their message ID, an attacker can induce part of the group to react positively (e.g., with 👍) to a benign URL, such as a cat picture. The same reaction can then appear attached to the attacker's spoofed message for members who instead received the malicious link, thereby increasing its perceived trustworthiness. An example of this attack is demonstrated in Figure 8.

5.2.3 Rigging Polls

While all investigated messaging services support polls today, the feature was introduced at different times. Threema was the first to introduce them in January 2015 [63]. WhatsApp introduced polls for group chats in November 2022 [31] and refined the feature in May 2023 [41]. More recently, iMessage added polls in June 2025 with iOS 26 [38], and Signal introduced the feature in November 2025 [15].

We begin with an analysis of how polls are implemented across different messaging services. We identified three distinct poll action types:

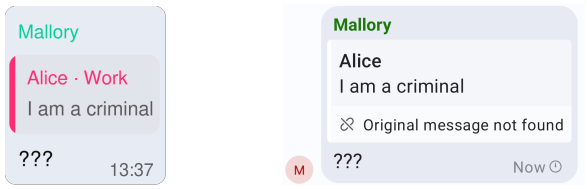
- **Poll Create:** A group member creates the poll by specifying the question and available options.
- **Poll Vote:** Each participant casts a vote for one or more options.
- **Poll Close:** In Signal and Threema, the poll creator can finalize the votes and close the poll.

Across all services, subsequent poll actions (i.e., *voting* and *closing*) reference the original *poll creation* message via its message ID. The *voting* action is implemented differently across services: WhatsApp encodes votes using a hash of the selected option, whereas Signal and Threema use an index, and iMessage relies on a UUID-based identifier.

In the following, we consider two types of attackers: malicious poll creators and voters.

Malicious Poll Creator. A malicious poll creator can send different poll creation messages to different participants, leading to divergent local interpretations.

Juggling Poll Options. For index- or ID-based schemes (Signal, Threema, iMessage), the attacker can reorder poll options



(a) WhatsApp (b) Signal

Figure 11: Screenshots of fake quote references on WhatsApp and Signal. A message references a nonexisting message, causing the client to display the attacker-provided fallback text. Signal shows that it does not find the referenced message.

or provide entirely different questions or option sets across recipients as demonstrated in Figure 9.

In hash-based schemes (i.e., WhatsApp), the attacker can manipulate poll options by introducing insignificant changes (e.g., whitespaces or encoding variations), thereby producing unrecognized hashes. Votes cast for these manipulated options are silently discarded without error for other participants.

Inverting Poll Questions. Across all services, the poll question itself is not protected by cryptography. Thus, an attacker can change it arbitrarily. For binary questions, this can be abused, even when hash-based voting is used (i.e., on WhatsApp), by inverting the question as shown in Figure 10. If hashing covered not only the selected voting option but also the corresponding poll question, such attacks would not be possible.

Dictating and Orchestrating Poll Results. In Threema, the poll creator rebroadcasts all poll votes previously collected by other participants for intermediate results. Thus, a malicious poll creator can add, remove, or rewrite votes for any group participant upon closing the poll, thereby altering the final result. The creator can close the poll multiple times in Threema, overwriting the entire result repeatedly (including the title/description and the options).

Signal does not require the creator to broadcast the final poll state upon closing. Instead, it sends a message to participating clients to close the poll on their end, referencing the original poll message by timestamp ID. The clients ignore future votes referencing an already closed poll. Thus, selectively sent poll close messages could make some members think a poll is still ongoing, when other members already see a final result.

Malicious Poll Voter. A malicious poll voter can equivocate by sending different voting messages to different participants. As a result, intermediate poll results displayed on recipients' devices may diverge, leading to inconsistent local views of the poll outcome.

For Threema, the main attack vector consists of sending manipulated votes to the poll creator, as the poll creator's final message overwrites intermediate results on other recipients' devices. Specifically, using a positive value different from 1 in a vote causes the vote to be counted toward the option's total number of votes, while the UI does not indicate that the user selected the option. As a result, a malicious voter can inject additional *hidden votes*, even in single-choice polls, and send them to the creator to invalidate the final poll outcome.

We show that transcript inconsistencies enable:
- ✓✓ conducting social engineering attacks, e.g., by sending different message content to different participants or by dropping messages;
- ✓✓ evading moderation, e.g., by sending benign URLs to moderators and most participants, while delivering phishing links to selected victims;
- ✓✓ manipulating polls, both as poll creator and as voter, due to transcript inconsistency, amplified by vulnerable design decisions, e.g., index-based voting.

Takeaway ✓✓

5.3 Implementation-Specific Issues

In addition to protocol-level observations, we identified several implementation-specific behaviors that introduce side channels and security risks. We found potentially problematic behavior related to TC in message quoting (WhatsApp, Signal) and implementation flaws that allow an adversary to submit votes on behalf of other participants (iMessage). Moreover, we observed notable differences in the client behavior across operating systems and device types (especially in WhatsApp and Signal), which can leak metadata or enable device fingerprinting in adversarial settings.

5.3.1 Quoted Messages

We identified potential issues related to quoted messages. While Threema and iMessage require the quoted message (referenced by its message ID) to exist and do not include fallback information in a sent reply, Signal and WhatsApp embed fallback text that is rendered if the referenced message cannot be resolved (e.g., for recently joined group participants). As a result, an adversary can craft messages that appear to quote content allegedly written by another participant, even though such a message was never sent, by providing a broken reference with attacker-controlled fallback text.

Figure 11 illustrates this behavior in the official WhatsApp and Signal clients. Although Signal's UI indicates that it cannot find the original message, an attacker can potentially exploit this behavior in practice, e.g., when a newly added group member sees a fabricated quote and lacks the context required to assess its authenticity. In contrast, WhatsApp does not display any warning alongside the quoted message, which directly facilitates social engineering attacks.

For Signal's desktop client [3], we additionally observed that the UI allows references to messages received after the quoting message, thereby violating causal ordering. We provide a Proof of Concept (PoC) video in our artifact [29].

Additionally, we noticed implementation/OS-specific differences in resolving collisions within the message ID on Signal, see Appendix A.2.

5.3.2 iMessage: Voting for Other Participants

Apple implements polls as a protocol extension named `MSMessageExtensionBalloonPlugin`. Internally, the client represents a poll as `JSON`. Also, for voting, the voter sends a `JSON` that includes a list of votes, each having a `voteOptionIdentifier`, the UUID of the option, and the `participantHandle`.

The voter can manipulate the `participantHandle` field in the vote message, e.g., by changing it to another group member or by inserting additional votes attributed to different participants. In this case, the poll overview suggests that another member voted for an option or that multiple participants voted for it. However, the detailed poll view still correctly attributes the vote, since it counts it towards the sender and ignores the `participantHandle`.

5.3.3 WhatsApp: OS-Specific Parsing of Polls

We observed differences in how polls are parsed and interpreted between Android and iOS clients. WhatsApp supports multiple user addressing formats, namely the *JID* (Jabber ID) and the *LID* (Logical ID), which decouples the user identifier from the phone number. Official implementations use the newer LID format when hashing and encrypting votes. However, we found that an attacker can craft votes using the legacy JID format, which are correctly parsed and displayed on Android but discarded and not shown on iOS devices.

Moreover, Android clients are more permissive when processing malformed *poll creation* messages. In particular, the Android client correctly parses and accepts polls containing repeated options, whereas iOS clients discard such polls entirely.

Conversely, during the voting phase, iOS correctly parses and accepts voting messages containing repeated (i.e., identical) hashes[7], whereas Android clients discard such messages.

In all three cases, an attacker can exploit these inconsistencies to induce divergent voting outcomes (e.g., by only letting Android users participate in the vote) without relying on the equivocation vectors described earlier.

Finally, we observed that the metadata appended to vote messages differs between Android and iOS, allowing an attacker to infer the sender's operating system by inspecting message metadata.

5.3.4 WhatsApp: OS-Specific Message IDs

During our analysis across platforms, operating systems, and client configurations, we found that WhatsApp message IDs for arbitrary message types are not generated uniformly and can leak the sender's OS. Table 3 summarizes the observed prefix and length values for messages on different OSs and device types. In an adversarial setting, such leakage can facilitate targeted social engineering and aid reconnaissance by enabling an attacker to fingerprint a victim's devices and tailor subsequent attacks or exploits accordingly. This privacy leak may also be abused in private relationships, for example, for stalking, by inferring that a message was sent from a specific device (e.g., a desktop computer), since the ID of a received message can be readily inspected in WhatsApp Web using standard browser developer tools by moderately skilled users.

OS	Client	Prefix	Len
Android	Main Device	`A5`	32
Android	Companion (e.g. Tablet)	`AC`	32
Android	Wear OS (e.g., SmartWatch)	`A3`	32
iOS	Main, Companion, watchOS	`3A`	20
iPadOS	Tablet iPadOS	`3C`	20
macOS	macOS Desktop	`3B`	20
Windows	Windows Desktop UWP[a]	`3F`	20
Windows	Windows Desktop Electron[b]	`3EB0`	22
Web	WhatsApp Web	`3EB0`	22

[a] Native UWP App was discontinued in Nov 2025.
[b] Current version uses Electron WebView [7].

Table 3: WhatsApp leaks a user's OS and device type through the prefix and length of generated message IDs.

We show that the current implementations allow:
- ✓✓ putting words into a victim's mouth by fabricating quoted messages in WhatsApp and Signal;
- ✓✓ spoofing votes for other participants for iMessage;
- ✓✓ poll tampering by exploiting OS-specific implementation differences between WhatsApp clients;
- ✓✓ fingerprinting the OS type of a message's sending client via the message ID in WhatsApp.

Takeaway ✓✓

6 Countermeasures and Mitigation Paths

Early academic papers already discussed solutions to check for TC in retrospect [32], or proposed entirely new messaging protocol designs to address the issue of TC [49, 51]. Apparently, none of those proposals were ever adopted for any of the major messaging services analyzed in this paper.

We therefore take a different approach. Instead of proposing another complete protocol (re-)design to address TC, we analyze which form of TC can be achieved within the currently deployed group messaging designs with as few modifications to protocol and threat model as possible. Specifically, we propose minimal modifications to the Sender Key protocol

[7] The vote is nevertheless counted only once.

used by two of the four analyzed prevalent instant messengers: WhatsApp and Signal. Our changes enable messengers to issue user-facing warnings when TC cannot be guaranteed, thereby making potential attacks transparent. In the best case, our modifications can ensure that all participants either know that STC might be violated, or that STC can eventually be achieved. This means that all group participants can arrive at a consistent transcript, given that they are online at the same time and new messages are submitted to the group slower than the network and processing delay of the slowest participant[8].

More generally, we argue that improving TC efficiently, i.e., without increasing communication complexity significantly, seems most realistic for group chats that operate in a common broadcast domain with shared group keys, where recipients can more readily detect inconsistencies and systems can provide meaningful user-facing warnings. This is a strong argument for group chat applications with TC to be built on top of MLS [13, 16]. Nevertheless, our findings highlight that explicitly considering TC at the design level remains crucial for current and future group messaging systems.

6.1 Sender Key with TC Improvements

To enhance the Sender Key protocol, we leverage the fact that all analyzed E2EE messaging designs already rely on a centralized server infrastructure to relay messages. If the server is assumed to be honest with respect to message delivery, similar to the trusted delivery service in MLS, addressing TC becomes significantly easier. We do not consider this assumption to constitute a substantial change to the trust or threat model, since the server already has the technical ability to arbitrarily drop messages and therefore must be trusted with respect to message delivery in any case.

Overall, our assumptions and rules for our improvements to Sender Key TC are:

- **Unsolicited Retransmissions:** Client implementations must drop unsolicited 1:1 group retransmissions.
- **Enforced Group Broadcast:** It can be enforced that a message to a group conversation is sent to all its members. In other words, clients must only accept messages from the server transmitted via the Sender Key protocol to the entire group, 1:1 group messages are only allowed to distribute a sender key.
- **Broadcast Retry Requests:** If a retry for a message in a group chat is requested by any participant, this retry request must be announced via server fan-out to all group participants, so that other honest members are informed about a message being dropped or retransmitted for another participant.
- **Static Group:** If a member joins or leaves, we treat this as a new group. Thus, the set of participants for a particular group is fixed and defined at group creation.
- **Server Timestamps:** The server is trusted to add the current timestamp to every message.

Under the assumption that the server faithfully delivers encrypted messages to all group participants using the Sender Key protocol via server fan-out, we can implicitly ensure that every group participant receives the same (signed) ciphertexts. There remain two cases that can still be problematic:

Different Key with Valid Decryption. Since each participant distributes their symmetric sender key to every other participant individually, a malicious sender can provide different recipients with different sender keys. In principle, a malicious sender could attempt to find a ciphertext and key combination that meaningfully decrypts into two different plaintexts. This attack was discussed in [49] and its success probability can be considered negligible with adequate parameter settings and domain separation.

Retransmission Due to Decryption Failure. Because group membership changes and device state resets can occur at any time, a participant may be unable to decrypt a received message using its current sender key for a given sender. This is an inherent property of the protocol and the asynchronous communication model. A malicious participant can abuse this by distributing different sender keys to trigger retry requests with the fallback to pairwise 1:1 communication, providing the original sender with the possibility to equivocate.

The first, albeit insufficient, countermeasure that suggests itself is to require that any retransmitted message also be redistributed to the entire group via server fan-out; we explain below why this approach is inadequate. Participants who already received and decrypted the message would then need to verify that the retransmitted ciphertext decrypts successfully and that its plaintext matches their local copy. If decryption fails again for a participant, that participant would need to issue a retry request for the retransmission itself. In effect, recipients would request retries for messages that already appear in their transcript solely to validate that the retransmitted content matches, which introduces a complex error resolution loop with no clear abort cutoff point.

To avoid this complexity, we propose treating *any* decryption error or retransmission as a potential equivocation signal. Rather than attempting to fix retransmissions within the current protocol, we suggest surfacing decryption errors and retransmissions in the UI, together with an explicit warning that the retransmitted content might differ across participants, and providing decryption error and retransmission statistics on a per-participant basis. Such statistics help distinguish false positives from attacks and support identifying the attacker.

6.1.1 Guarantees and Limitations

If no decryption failures occur, all participants are guaranteed to eventually see the same messages (STC). In the presence

[8] See Appendix A.1 for details.

of decryption failures and associated warnings, no guarantees can be provided for the content of the affected messages. However, highlighting such messages in the UI as potentially different across participants can already act as a sufficient deterrent against targeted equivocation.

Currently, the ordering of messages works by appending messages as received, or based on when a message was sent from the local device. If messages would be ordered by server timestamp, this could lead to cases where messages "hop around" in the chat window: For example, a message created locally would require a roundtrip to the server to get a server timestamp according to which it can be ordered in the UI. The alternative would be to only reorder messages totally (TTC) in retrospect upon explicit user intent.

While a malicious participant could deliberately trigger warning messages, our countermeasure still flags the anomaly and exposes the involved sender and recipient, thus working as intended. This improves upon the status quo, where attackers can operate completely undetected. A more pressing concern is false positives caused by legitimate retransmissions due to technical failures. A short self-study did not reveal any decryption failures during normal operation, but accurately quantifying their real-world likelihood would require a large-scale experiment and a complementary user study that are difficult to deploy without platform-operator support. Thus, this is left for future work. More fundamentally, a general, protocol-agnostic fix is a hard problem, as prior approaches rely on full protocol redesigns and often neglect performance. Rather than attempting such a redesign in this paper, which would exceed typical conference length constraints, we deliberately present the least intrusive mitigation that integrates into deployed systems, fully aware of its imperfections.

6.1.2 Implementation Considerations

We highlight the cornerstones of the proposed changes for WhatsApp and Signal to account for the different protocol architectures. The two designs differ in the server's view of group membership. In WhatsApp, the server is aware of the group participants, whereas Signal supports private groups where the server does not store the group composition in plaintext. To account for our previous *Enforced Group Broadcast* assumption, the following outlines how this may be achievable with the respective messenger services.

WhatsApp. In the case of WhatsApp, the server can check and ensure delivery to the entire group by directly invoking the appropriate group endpoint and relying on the group state information already available on the WhatsApp server.

Signal. Signal aims to minimize the plaintext state and metadata available on its servers, for example by supporting private groups [19, 39] and sealed sender [40, 55]. While the chat server's validation process for multi-recipient group sends checks the supplied group send token for valid endorsements against the list of recipients [59], the current zero-knowledge architecture does not seem to allow for direct associations with group member state [57]. As a consequence, the chat server cannot directly verify that a multi-recipient message targets all group participants, which necessitates shifting delivery checks to the clients.

In this setting, the server needs to attach some form of a participant list[9] to each multi-recipient message[10]. In return, this approach enables validation of group messages against the authoritative group state maintained via *zkgroups*. Concretely, clients could validate the server-provided recipient list against the group state referenced in the message plaintext through its *group revision number*. This requires that the correct revision number is added by the sending client, and that the receiving client is able to reconstruct the according participant list. Since group membership changes could occur at any time, these client-side checks may conflict with concurrent changes to group state. In the simplest case, group membership is static and defined at creation[11]. If the reconstructed participant list does not match the information provided by the server, receiving clients should drop the respective message.

7 Related Work and TC History

One of the earliest works on transcript agreement in group messaging is by Goldberg et al. [32], who proposed Multi-party Off-the-Record Messaging (mpOTR). They described a property closely related to TC, which they termed *consensus*. Their approach relies on retrospective consistency checks after session termination, allowing participants to detect violations but offers no remediation during an ongoing conversation. Marlinspike later adopted this idea when informally describing a desired consistency property for TextSecure [39], which later evolved into Signal. As we demonstrate, current Signal group chats do not provide this property.

Unger et al. [67] systematized transcript-related guarantees and decomposed TC into three conceptually related desiderata, namely *speaker consistency*, *causality preserving*, and a *global transcript*. They emphasized that a global transcript implies speaker consistency and further observed that pairwise OTR connections in groups inherently violate both speaker consistency and causality.

Subsequent work shifted focus toward attacks by a malicious messaging server, while explicitly or implicitly assuming that all group participants behave honestly. Schliep et al. [50] analyzed attacks caused by a malicious Signal server, with a focus on message reordering and message dropping.

[9] For example, a hash of the participant list.

[10] Which must not be a story, i.e., have the *isStory* flag set.

[11] Alternatively, a sending client providing an incorrect revision number could be detected by relying on server-provided timestamps for messages and recorded group state changes. The sender would then verify the revision using the server timestamp.

They did not consider inconsistencies caused by malicious group members. A follow-up work in 2019 [49] proposed conversation integrity guarantees by introducing order-enforcing service providers that act similarly to trusted timestamping services. From a practical perspective, these results do not directly transfer to modern Signal group chats, since the implementation at the time differed substantially from today's design and did not yet support the Sender Key protocol [69] or private groups [35].

Research on the Sender Key protocol itself focused on formal modeling and security analysis [11] and on protocol improvements [12, 19]. These works do not address TC violations caused by equivocation or selective message delivery by malicious participants.

Rösler et al. [48] formalized security goals for group messaging and introduced *traceable delivery*, which allows detecting whether a sent message was received. Their threat model assumes group members to be honest and to follow the protocol. At the time of their study, Signal did not provide this property, not even for 1:1 chats. In contrast, we explicitly consider malicious group participants and focus on inconsistencies that arise even when the server behaves correctly. Several works studied properties closely related to TC, such as causality and history integrity. Eugster et al. [23] explored goals for secure group communication while considering causality, but they neither aimed to tolerate Byzantine participants nor required total order delivery. Barooti et al. [14] introduced *history integrity* and studied active attacks in the presence of a compromised device state. Their analysis focused on 1:1 conversations and did not consider group messaging. Transcript franking [20, 43] represents another related line of work. It aims to enable selective disclosure of conversation transcripts to moderators or authorities in cases of abuse. Correct transcript franking requires preserving message causality. Chen et al. [20] showed that existing instant messengers fail to preserve causality even in 1:1 conversations, and that TLS 1.3 itself does not guarantee this property. They further identified causality-preserving group messaging as an open problem.

MLS [13, 16, 46] defines a notion termed *transcript consistency*, but this property applies exclusively to the handshake protocol and thus to group state rather than application messages [47]. MLS aims to ensure that all honest members share a consistent view of group state, including group membership, proposal processing, and key schedule evolution. However, MLS is a group key establishment and state agreement protocol, not a messaging protocol. It deliberately leaves TC for application messages to the application layer.

Recent work has shown that legitimate users can exploit protocol mechanisms in deployed E2EE messengers to undermine security and privacy guarantees [24, 25, 30]. Complementing these lines of work, we study TC violations caused by malicious group participants in deployed E2EE messaging applications, demonstrate practical attacks, and discuss mitigations that integrate with the Sender Key protocol.

8 Discussion

While confidentiality and authenticity against external adversaries are relatively well-understood and carefully engineered, the integrity of the shared group transcript in the presence of malicious participants has largely remained unaddressed.

This gap is particularly visible from the user's perspective. Group chats (and especially interactive features such as polls) create the expectation that all participants observe the same conversation and derive the same outcomes. In practice, none of the analyzed systems enforce such guarantees. As a result, TC failures violate basic user expectations about how group communication functions.

A key reason for this mismatch is the prevailing threat model in protocol design, which does not account for participants targeting the consistency of the conversation. Over the last decade, encrypted messaging applications have shifted from services for communication among trusted parties (e.g., friends and family) to platforms that support large, open groups where participants join via links or QR codes, representing much weaker trust relationships. Additionally, they are used as channels for coordination and decision-making by organizations, officials, and other high-value groups in adversarial settings (e.g., the current geopolitical situation). In this context, the threat does not necessarily stem from malicious intent on the part of an account holder but may instead arise from a threat actor who has compromised a user's device participating in the respective group.

Beyond impact for normal messages, additional message types (e.g., location sharing, polls) create even more attack vectors and further amplify the problem. We observed numerous behaviors (e.g., naive implementations of polls, platform-dependent message handling) that reduce attacker effort and increase the practical impact of transcript inconsistencies. The prevalence of such issues stands in contrast to the expectation of highly hardened messengers deployed at a global scale.

A natural direction for future work is to better understand how transcript inconsistencies (or future UI warnings) manifest at the user level and whether they are noticed in practice. Extending our analysis to additional E2EE messaging platforms and emerging group messaging designs (e.g., MLS) could help assess whether TC can be readily improved in newer systems or whether similar design trade-offs persist. Finally, we argue that TC should be addressed at the design level with the same rigor as other well-established security and privacy properties of modern E2EE instant messengers.

9 Conclusion

In this paper, we answered *What is the current state of TC in E2EE messaging?* by showing that all analyzed E2EE messengers, namely WhatsApp, Signal, iMessage, and Threema, failed to provide TC. As a result, malicious group participants could selectively omit, reorder, or tailor messages to present

different content to different participants. These attacks directly undermine the integrity of the shared group transcript.

Our analysis showed that these weaknesses arose from fundamental design choices rather than isolated implementation bugs. Despite long-standing awareness of transcript consistency issues in academia and expert communities, current systems offer neither preventive mechanisms nor effective means to detect them (neither on the server nor on the receiving client). The risk posed by transcript inconsistency is further amplified by growing group sizes, newly introduced features such as polls, and the sensitive contexts in which these messengers are used in current times. To improve the existing situation, we proposed practical protocol changes that require minimal adaptations to the Sender Key protocol, which Signal and WhatsApp use, to detect inconsistencies.

Ethical Considerations

Overall, we conducted all experiments exclusively using our own accounts to avoid any harm to real users. We further identify two stakeholder groups affected by our findings: (1) instant messaging users, and (2) platform operators. In the following, we discuss the potential impact of our findings on each stakeholder group and describe the mitigation measures we apply to reduce harm.

Instant Messaging Users. Publishing our research could facilitate the exploitation of TC inconsistencies. Attackers could leverage our insights for sophisticated social engineering attacks against instant messaging users.

However, withholding our findings would amount to security by obscurity, which practical experience has repeatedly shown to be ineffective as a security strategy. We include our PoC code for all messenger applications in the paper submission and publish it to support reproducibility and future work after a 90 days grace period.

The vulnerabilities affect all analyzed E2EE messaging apps, which makes raising public awareness essential. Increased awareness can help users to better understand the threat model and security guarantees of group chats, or even recognize and detect such attacks in certain cases. In addition, disseminating our findings within the research community fosters the development and evaluation of countermeasures.

Additionally, we aimed to minimize potential harm to users and platforms by responsibly disclosing our findings to the affected platform operators before publishing.

Platform Operators. Highlighting existing issues is essential to enable long-term improvements. Our proposed countermeasures require only minimal design changes to the existing Sender Key protocol, which WhatsApp and Signal currently use. Therefore, we consider our countermeasures a practical route to improve upon the current situation with relatively low overhead. In addition, it is particularly important to incorporate our findings into future protocol designs, as mitigating these risks without understanding them is challenging.

As mentioned earlier, we responsibly disclosed all findings to the platform operators of the analyzed messaging apps, which gives them the opportunity to address the issues before public disclosure.

Responsible Disclosure

On February 6th, 2026 we responsibly disclosed the identified issues to the affected platform operators by sharing a preprint of this paper together with individual issue reports. For WhatsApp, Apple, and Threema, we used the respective bug bounty or vulnerability disclosure platforms. Signal was contacted via its (publicly recommended) security contact email address.

At the time of writing (2026-07-29), all messengers (WhatsApp, Apple, Threema, and Signal) acknowledged receipt of the reports, and provided feedback on the identified issues. Meta awarded a USD 1,000 bug bounty and indicated plans to address implementation-specific issues (e.g., adding security hashes protecting the poll question), but no protocol-level transcript consistency changes were communicated. Similarly, Apple expressed appreciation for the disclosure, but indicated that it does not currently plan changes related to transcript consistency. Threema plans to address implementation-specific issues (i.e., inflated votes by a single participant), while they consider fabricated results of a poll creator less relevant to their current threat model and also do not plan to implement protocol-level changes regarding transcript consistency in the near future. Signal highlighted that transcript consistency does not currently constitute an explicit security goal. However, Signal also noted that its ongoing redesign of group chats may bring transcript consistency closer to its scope of future security goals.

Response Time. Apple responded approximately one month after the disclosure, while WhatsApp and Threema each responded after roughly three months, and Signal took about five months to respond.

Paper and Artifact Release. We made sure to give all platform operators sufficient time ($>$ 90 days) to process our reports, communicate their mitigation plans and request an embargo. To support comprehensive evaluation and reproducibility of our work and to increase awareness within the community, we publicly release the artifacts at https://github.com/sbaresearch/transcript-consistency.

Open Science

Since more than 90 days have passed and vendors do not consider TC as part of their threat model, the artifacts are made available to foster reproducibility and future work: on GitHub at https://github.com/sbaresearch/transcript-consistency and on Zenodo for long-term archiving at https://doi.org/10.5281/zenodo.20323807.

The provided repository contains the following:

- `code/`: This directory contains the code of our custom clients for all targeted messengers.
- `code/imessage/`: This directory contains a modified version of rustpush [2] that includes a command-line client to send inconsistent messages into group chats and to manipulate polls and votes.
- `code/signal/poc-skdm/`: This directory contains a modified Signal Desktop [3] client demonstrating the Sender Key with E2EE fallback case for a specific recipient selected in the GUI.
- `code/signal/poc-groups/`: This directory contains a modified Signal Desktop [3] client supporting server and client fan-out-only modes and providing a more advanced GUI for exclusions and overrides for all considered message types (e.g., polls, regular messages).
- `code/threema/threema-android/`: This directory contains a modified Threema Android [4] client with support for commands in relevant input fields (for regular messages and polls) to control excluded recipients and overrides, as well as poll voting and closing.
- `code/whatsapp/poc-client/`: This directory contains our modified WhatsApp client that allows sending inconsistent messages while supporting server and client fan-out-only modes, as well as Sender Key with E2EE fallback, which builds on whatsmeow [5].
- `code/whatsapp/quote-client/`: This directory contains our modified WhatsApp client to manipulate quotes, building on whatsmeow [5].
- `videos/`: This directory contains PoC videos showing the behavior of official clients in a group with a malicious client.

Throughout the repository, `README.md` files provide the information required to reproduce the results.

Environment. For the Signal Desktop client, in particular, we provide containers for improved reproducibility since there are many dependencies and the build steps would otherwise be prone to failure. The entire GUI can be run inside a container.

Acknowledgments

The financial support by the Austrian Federal Ministry of Economy, Energy and Tourism, the National Foundation for Research, Technology and Development, the Christian Doppler Research Association, the FFG Bridge project 46322124 SecKey, the FFG KIRAS/K-PASS project 59103683 TelCrit, and the University of Vienna, Faculty of Computer Science, Security & Privacy Group, SBA Research (SBA-K1 NGC) a COMET Center within the COMET – Competence Centers for Excellent Technologies Program funded by BMIMI, BMWET, and the federal state of Vienna, is gratefully acknowledged. We would also like to thank our anonymous reviewers for their valuable feedback and suggestions.

References

[1] Apple Community – Sending group messages to 50+ recipients. Accessed: 2026-01-27. URL: https://discussions.apple.com/thread/255951019.

[2] GitHub – Rustpush. Accessed: 2026-01-27. URL: https://github.com/OpenBubbles/rustpush.

[3] GitHub – Signal Desktop. Accessed: 2026-01-27. URL: https://github.com/signalapp/Signal-Desktop.

[4] GitHub – Threema for Android. Accessed: 2026-01-27. URL: https://github.com/threema-ch/threema-android.

[5] GitHub – whatsmeow. Accessed: 2026-01-27. URL: https://github.com/tulir/whatsmeow.

[6] iMessage Statistics. Accessed: 2026-01-23. URL: https://usesignhouse.com/blog/imessage-stats/.

[7] Meta just killed native WhatsApp on Windows 11, now it opens WebView. Accessed 2026-01-23. URL: https://www.windowslatest.com/2025/11/12/meta-just-killed-native-whatsapp-on-windows-11-now-it-opens-webview-uses-1gb-ram-all-the-time/.

[8] Signal Statistics. Accessed: 2026-01-23. URL: https://www.businessofapps.com/data/signal-statistics/.

[9] Threema Press Information. Accessed: 2026-01-23. URL: https://threema.com/press-files/1_press_info/press_threema_portrait_en.pdf.

[10] Apple. Messages for iPhone, iPad, Apple Watch, and Mac. Accessed: 2026-01-27. URL: https://support.apple.com/messages.

[11] David Balbás, Daniel Collins, and Phillip Gajland. Analysis and Improvements of the Sender Keys Protocol for Group Messaging. *CoRR*, abs/2301.07045, 2023. arXiv: 2301.07045. URL: https://doi.org/10.48550/arXiv.2301.07045, doi:10.48550/ARXIV.2301.07045.

[12] David Balbás, Daniel Collins, and Phillip Gajland. WhatsUpp with Sender Keys? Analysis, Improvements and Security Proofs. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part V*, volume 14442 of *Lecture Notes in Computer Science*, pages 307–341. Springer, 2023. URL: https://doi.org/10.1007/978-981-99-8733-7_10, doi:10.1007/978-981-99-8733-7_10.

[13] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol, July 2023. Issue: 9420 Num Pages: 132 Series: Request for Comments Published: RFC 9420. URL: https://www.rfc-editor.org/info/rfc9420, doi:10.17487/RFC9420.

[14] Khashayar Barooti, Daniel Collins, Simone Colombo, Loïs Huguenin-Dumittan, and Serge Vaudenay. On Active Attack Detection in Messaging with Immediate Decryption. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 362–395. Springer, 2023. URL: https://doi.org/10.1007/978-3-031-38551-3_12, doi:10.1007/978-3-031-38551-3_12.

[15] Nina Berman. Signal Polls: Yes, no, maybe (yes!). Accessed: 2026-01-27. URL: https://signal.org/blog/polls.

[16] Benjamin Beurdouche, Eric Rescorla, Emad Omara, Srinivas Inguva, and Alan Duric. The Messaging Layer Security (MLS) Architecture, April 2025. Issue: 9750 Num Pages: 41 Series: Request for Comments Published: RFC 9750. URL: https://www.rfc-editor.org/info/rfc9750, doi:10.17487/RFC9750.

[17] Gabriel Bracha. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing (PODC '84)*, pages 154–162, Vancouver, British Columbia, Canada, 1984. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/800222.806743, doi:10.1145/800222.806743.

[18] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011. doi:10.1007/978-3-642-15260-3.

[19] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1445–1459. ACM, 2020. URL: https://eprint.iacr.org/2019/1416.pdf, doi:10.1145/3372297.3417887.

[20] Shan Chen and Marc Fischlin. Integrating Causality in Messaging Channels. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part III*, volume 14653 of *Lecture Notes in Computer Science*, pages 251–282. Springer, 2024. URL: https://eprint.iacr.org/2024/362.pdf, doi:10.1007/978-3-031-58734-4_9.

[21] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1802–1819. ACM, 2018. doi:10.1145/3243734.3243747.

[22] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the Presence of Partial Synchrony. *Journal of the ACM*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

[23] Patrick Eugster, Giorgia Azzurra Marson, and Bertram Poettering. A Cryptographic Look at Multi-party Channels. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 31–45. IEEE Computer Society, 2018. doi:10.1109/CSF.2018.00010.

[24] Viktor E. Garske, Swantje Lange, Gabriel K. Gegenhuber, David Schmidt, Andreas Noack, and Jiska Classen. Blue Bubbles, Red Flags: Investigating Privacy Leakage in Apple iMessage. In *Proceedings of the 2026 ACM*

SIGSAC Conference on Computer and Communications Security, 2026.

[25] Gabriel K. Gegenhuber, Philipp É. Frenzel, Maximilian Günther, and Aljosha Jud ayer. Prekey Pogo: Investigating Security and Privacy Issues in WhatsApp's Handshake Mechan sm. In *19th USENIX WOOT Conference on Offensive Technologies (WOOT)*, 2025.

[26] Gabriel K. Gegenhuber, Philipp É. Frenzel, Maximilian Günther, Johanna Ullrich, and Aljosha Judmayer. Hey there! You are using WhatsApp: Enumerating Three Billion Accounts for Security and Privacy. In *33rd Annual Network and Distributed System Security Symposium (NDSS)*, 2026.

[27] Gabriel K. Gegenhuber, Moritz Grefner, Maximilian Günther, Matthäus Wininger, David Schmidt, and Aljosha Judmayer. Duplicate Message Edit ID in Signal. URL: https://github.com/sbaresearch/transcript-consistency/blob/main/videos/signal/16/desktop-duplicate-handling-2.mkv.

[28] Gabriel K. Gegenhuber, Moritz Grefner, Maximilian Günther, Matthäus Wininger, David Schmidt, and Aljosha Judmayer. Duplicate Message ID in Signal. URL: https://github.com/sbaresearch/transcript-consistency/blob/main/videos/signal/15/desktop-duplicate-handling-1.mkv.

[29] Gabriel K. Gegenhuber, Moritz Grefner, Maximilian Günther, Matthäus Wininger, David Schmidt, and Aljosha Judmayer. Malicious Signal Message Referencing Future Message. URL: https://github.com/sbaresearch/transcript-consistency/blob/main/videos/signal/14/fake-quote-2-wrong-causality.mkv.

[30] Gabriel K. Gegenhuber, Maximilian Günther, Markus Maier, Aljosha Judmayer, Florian Holzbauer, Philipp É. Frenzel, and Johanna Ullrich. Careless Whisper: Exploiting Silent Delivery Receipts to Monitor Users on Mobile Instant Messengers, 2024. URL: https://arxiv.org/abs/2411.11194, arXiv:2411.11194.

[31] Thomas Germain. WhatsApp Revamps Group Chats with New Communities Feature. Accessed: 2026-01-27. URL: https://gizmodo.com/whatsapp-communities-new-group-chat-feature-1849737258.

[32] Ian Goldberg, Berkant Ustaoglu, Matthew Van Gundy, and Hao Chen. Multi-party off-the-record messaging. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*, pages 358–368. ACM, 2009. doi:10.1145/1653662.1653705.

[33] Jeffrey Goldberg. The Trump Administration Accidentally Texted Me Its War Plans, March 2025. URL: https://www.theatlantic.com/politics/archive/2025/03/trump-administration-accidentally-texted-me-its-war-plans/682151.

[34] Martin Holland. EU Signal Group: Too sensitive for release, not sensitive enough for archiving, November 2025. URL: https://www.heise.de/en/news/EU-Signal-Group-Too-sensitive-for-release-not-sensitive-enough-for-archiving-11080698.html.

[35] jimio. Technology Preview: Signal Private Group System, December 2019. URL: https://signal.org/blog/signal-private-group-system/.

[36] Martin Kleppmann, Stephan A. Kollmann, Diana A. Vasile, and Alastair R. Beresford. From Secure Messaging to Secure Collaboration. In Vashek Matyás, Petr Svenda, Frank Stajano, Bruce Christianson, and Jonathan Anderson, editors, *Security Protocols XXVI - 26th International Workshop, Cambridge, UK, March 19-21, 2018, Revised Selected Papers*, volume 11286 of *Lecture Notes in Computer Science*, pages 179–185. Springer, 2018. URL: https://martin.kleppmann.com/papers/secure-collaboration-spw18.pdf, doi:10.1007/978-3-030-03251-7_21.

[37] Micah Lee. Despite misleading marketing, Israeli company TeleMessage, used by Trump officials, can access plaintext chat logs, May 2025. URL: https://micahflee.com/despite-misleading-marketing-israeli-company-telemessage-used-by-trump-officials-can-access-plaintext-chat-logs/.

[38] Aisha Malik. Apple is bringing polls to Messages in iOS 26. Accessed: 2026-01-27. URL: https://techcrunch.com/2025/06/09/apple-is-bringing-polls-to-imessage-in-ios-26.

[39] Moxie Marlinspike. Private Group Messaging, May 2014. URL: https://signal.org/blog/private-groups/.

[40] Ian Martiny, Gabriel Kaptchuk, Adam J. Aviv, Daniel S. Roche, and Eric Wustrow. Improving Signal's Sealed Sender. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021. URL: https://www.cs.umd.edu/~kaptchuk/publications/ndss21.pdf.

[41] Meta. New Updates to Polls and Sharing With Captions on WhatsApp. Accessed: 2026-01-27. URL: https://about.fb.com/news/2023/05/whatsapp-polls-updates-sharing-with-captions.

[42] Meta. WhatsApp – How to join a group in a community. Accessed: 2026-01-27. URL: https://faq.whatsapp.com/967457667545238/.

[43] Armin Namavari and Thomas Ristenpart. Transcript Franking for Encrypted Messaging. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part II*, volume 16246 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2025. URL: https://doi.org/10.1007/978-981-95-5096-8_1, doi:10.1007/978-981-95-5096-8_1.

[44] Rafael Pass and Elaine Shi. The Sleepy Model of Consensus. In *Advances in Cryptology – ASIACRYPT 2017*, volume 10624 of *Lecture Notes in Computer Science*, pages 380–409. Springer, 2017. doi:10.1007/978-3-319-70697-9_14.

[45] Ole André V. Ravnås. Frida – A world-class dynamic instrumentation toolkit. URL: https://frida.re/.

[46] Raphael Robert. The Messaging Layer Security (MLS) Extensions. Internet-Draft draft-ietf-mls-extensions-08, Internet Engineering Task Force, July 2025. Backup Publisher: Internet Engineering Task Force Num Pages: 37. URL: https://datatracker.ietf.org/doc/draft-ietf-mls-extensions/08/.

[47] Raphael Robert. Messaging Layer Security (MLS): Towards More End-to-End Encryption, July 2025. Published: Presentation, Pass the SALT 2025.

[48] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 415–429. IEEE, 2018. URL: https://doi.org/10.1109/EuroSP.2018.00036, doi:10.1109/EUROSP.2018.00036.

[49] Michael Schliep and Nicholas Hopper. End-to-End Secure Mobile Group Messaging with Conversation Integrity and Deniability. In Lorenzo Cavallaro, Johannes Kinder, and Josep Domingo-Ferrer, editors, *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society, WPES@CCS 2019, London, UK, November 11, 2019*, pages 55–73. ACM, 2019. URL: https://www-users.cse.umn.edu/~hoppernj/cowpi_wpes19.pdf, doi:10.1145/3338498.3358644.

[50] Michael Schliep, Ian Kariniemi, and Nicholas Hopper. Is Bob Sending Mixed Signals? In Bhavani Thuraisingham and Adam J. Lee, editors, *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society, Dallas, TX, USA, October 30 - November 3, 2017*, pages 31–40. ACM, 2017. URL: https://www-users.cse.umn.edu/~hoppernj/mixed_signals_wpes17.pdf, doi:10.1145/3139550.3139568.

[51] Michael Schliep, Eugene Y. Vasserman, and Nicholas Hopper. Consistent Synchronous Group Off-The-Record Messaging with SYM-GOTR. *Proc. Priv. Enhancing Technol.*, 2018(3):181–202, 2018. URL: https://doi.org/10.1515/popets-2018-0027, doi:10.1515/POPETS-2018-0027.

[52] Signal. Signal » Home. Accessed: 2026-01-27. URL: https://signal.org/.

[53] Signal Messenger. New Features Coming to Signal Groups, October 2020. URL: https://signal.org/blog/new-groups/.

[54] Signal Support. Group chats. Accessed: 2026-01-27. URL: https://support.signal.org/hc/en-us/articles/360007319331-Group-chats.

[55] Signal Technology Foundation / signalapp. libsignal-protocol-java: A Java implementation of the Signal Protocol, 2022. URL: https://github.com/signalapp/libsignal-protocol-java.

[56] signalapp. Signal-Android: Commit 0459d118a397e5405c9742e3f6238189e1fe0c9d. Accessed: 2026-02-05. URL: https://github.com/signalapp/Signal-Android/commit/0459d118a397e5405c9742e3f6238189e1fe0c9d.

[57] signalapp. GroupSendEndorsement.ts at commit 85686ca in *libsignal*, 2025. Accessed: 2026-02-05. URL: https://github.com/signalapp/libsignal/blob/85686caa01465eacba6fddcdc19a22d2d62d8c7f/node/ts/zkgroup/groupsend/GroupSendEndorsement.ts.

[58] signalapp. IncomingMessage.java at commit dc3920a in *Signal-Server*, 2025. Accessed: 2026-02-04. URL: https://github.com/signalapp/Signal-Server/blob/dc3920a99cb25cae66ca7439004545b69ae55ca4/service/src/main/java/org/whispersystems/textsecuregcm/entities/IncomingMessage.java#L79-L89.

[59] signalapp. MessageController.java at commit ad21f00 in *Signal-Server*, 2026. Accessed: 2026-02-05. URL: https://gi

thub.com/signalapp/Signal-Server/blob/ad21f002ab837f931c28a5ea020d82eb0b1f43aa/service/src/main/java/org/whispersystems/textsecuregcm/controllers/MessageController.java#L589.

[60] signalapp. RemoteConfig.kt at commit 1ddde6a in *Signal-Android*, 2026. Accessed: 2026-02-05. URL: https://github.com/signalapp/Signal-Android/blob/1ddde6ab92f39dff0987f60a08460412a0879a55/app/src/main/java/org/thoughtcrime/securesms/util/RemoteConfig.kt#L547-L563.

[61] Threema. Secure Communication For Individuals and Companies. Accessed: 2026-01-27. URL: https://threema.com/en.

[62] Threema. Threema for iOS: Larger Groups and More. Accessed: 2026-01-27. URL: https://threema.com/en/blog/threema-464-for-ios.

[63] Threema. Threema Poll Feature. Accessed: 2026-01-27. URL: https://threema.com/en/blog/threema-poll-feature.

[64] Threema. Cryptography Whitepaper. Technical report, March 2025. Accessed: 2026-01-30. URL: https://threema.com/press-files/2_documentation/cryptography_whitepaper.pdf.

[65] threema-ch. Threema-Info.plist at commit 7b1636e in *threema-ios*, 2025. Accessed: 2026-01-29; iOS app source file from the Threema open-source repository. URL: https://github.com/threema-ch/threema-ios/blob/7b1636e9a1e765f6aa1db7a71420c4d3874b0ca5/Threema/SupportingFiles/Threema/Threema-Info.plist#L118-L119.

[66] Sam Toueg. Randomized Byzantine Agreements. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing (PODC '84)*, pages 163–178, Vancouver, British Columbia, Canada, 1984. Association for Computing Machinery. URL: https://dl.acm.org/doi/10.1145/800222.806744, doi:10.1145/800222.806744.

[67] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. SoK: Secure Messaging. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 232–249. IEEE Computer Society, 2015. doi:10.1109/SP.2015.22.

[68] WhatsApp. Secure and Reliable Free Private Messaging and Calling. Accessed: 2026-01-27. URL: https://www.whatsapp.com/.

[69] Whatsapp. WhatsApp Encryption Overview: Technical white paper. Technical report, September 2023. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf.

A Appendix

A.1 Defining Transcript Consistency

The ambiguous use of the term *transcript consistency*, e.g. in the context of MLS where it is only referring to group state [13, 16, 47], as well as the various informal descriptions of its desired properties [32, 36, 39] necessitate more fine-grained definitions of the different flavors of this security property.

We define four versions of this security property in the context of secure group messaging, allowing for different design and implementation approaches. Our definitions range from the weakest form of TC to its strongest form. For all versions, we consider a group chat with a set of participants $\mathcal{P} = P_1, P_2, P_3, \ldots, P_n$ that are already added to the group[12]. The attacker is a member of the respective group conversation and controls at most f participants. The total group size is fixed and denoted by n, where $f \leq \lfloor \frac{n-1}{3} \rfloor$ holds. This bound is necessary to ensure that agreement on a common message transcript is theoretically possible under asynchrony and malicious participants [17, 66].

Communication in the context of instant messaging is assumed to be asynchronous. Moreover, honest participants can go offline at any time, akin to the sleepy model in consensus [44]. For TC this is challenging, as a consistent transcript cannot be ensured for all honest participants in times where honest participants are offline. We account for this in our definitions of TC by omitting when exactly, or how often, TC should be achieved. This leaves room for different approaches as to when TC should be verified or enforced. In a basic form, it could be sufficient to *eventually* arrive at a consistent transcript: When all participants are online and no additional message is sent within the group for at least Δ time, where Δ is the propagation delay (network as well as processing) of the slowest participant in a group[13]. Therefore, TC can eventually be achieved if such situations arise. This deliberately leaves unspecified whether participants are required to know when TC is achieved, or whether it is sufficient to know that TC remains achievable and has not been violated.

Definition 1 (Set Transcript Consistency)**.** *Consider a group of participants $\mathcal{P}$ executing a secure group messaging protocol in the presence of an adversary that controls a subset of Byzantine participants. Each participant $i \in \mathcal{P}$ maintains a*

[12] Users may operate multiple linked devices, each maintaining independent cryptographic state, as supported by the messaging systems under investigation. At a high level, this behavior can be modeled as separate participants.

[13] This assumption is comparable to GST (Global Stabilization Time) [22].

local transcript Tr_i*, defined as the collection of messages that* i *accepts as belonging to the conversation after performing all protocol-defined verification checks.*

The protocol provides Set Transcript Consistency (STC) *if, for any execution and for any two honest participants* $i, j \in P$*, the following eventually holds:*

$$\text{set}(\mathsf{Tr}_i) = \text{set}(\mathsf{Tr}_j).$$

That is, all honest participants eventually accept exactly the same set of messages, independently of the order in which those messages are delivered or displayed locally.

STC should reflect the first description of such a property by Goldberg et al. [32] which suggested to perform this consistency check (termed *consensus* in [32]) in retrospect after shutting down a group chat by lexicographically ordering all messages and comparing the digests.

Definition 2 (Participant Transcript Consistency)**.** *Consider a group of participants* $\mathcal{P}$ *executing a secure group messaging protocol in the presence of an adversary that controls a subset of Byzantine participants. Each participant* $i \in \mathcal{P}$ *maintains a local* transcript Tr_i*, defined as the set of messages that* i *accepts as belonging to the conversation after performing all protocol-defined verification checks.*

Each message m *is associated with a unique sender* $\mathsf{snd}(m) \in \mathcal{P}$ *and a sender-local sequence number* $\mathsf{seq}(m) \in \mathbb{N}$*.*

The protocol provides Participant Transcript Consistency (PTC) *if, for any execution and for any two honest participants* $i, j \in \mathcal{P}$*, the following properties eventually hold:*

1. ***Set Agreement:***

$$\text{set}(\mathsf{Tr}_i) = \text{set}(\mathsf{Tr}_j).$$

2. ***Per-Speaker FIFO Order:*** *For any two messages* m_1, m_2 *such that* $\mathsf{snd}(m_1) = \mathsf{snd}(m_2)$ *and* $\mathsf{seq}(m_1) < \mathsf{seq}(m_2)$*, if* m_1 *precedes* m_2 *in* Tr_i *for some honest participant* i*, then* m_1 *precedes* m_2 *in* Tr_j *for every honest participant* j*.*

No ordering constraint is imposed on messages originating from different senders.

This definition of PTC is comparable to the property termed *speaker consistency* by Unger et al. [67].

Definition 3 (Causal Transcript Consistency)**.** *Consider a group of participants* $\mathcal{P}$ *executing a secure group messaging protocol in the presence of an adversary that controls a subset of Byzantine participants. Each participant* $i \in \mathcal{P}$ *maintains a local* transcript Tr_i*, defined as the set of messages that* i *accepts as belonging to the conversation after performing all protocol-defined verification checks, together with a local delivery order.*

Let $\rightarrow$ *denote a protocol-defined* causal precedence *relation over messages (e.g., induced by send-receive dependencies or explicit causal metadata).*

The protocol provides Causal Transcript Consistency (CTC) *if, for any execution and for any two honest participants* $i, j \in \mathcal{P}$*, the following properties eventually hold:*

1. ***Set Agreement:***

$$\text{set}(\mathsf{Tr}_i) = \text{set}(\mathsf{Tr}_j).$$

2. ***Causal Order Agreement:*** *For any two messages* m_1, m_2 *such that* $m_1 \rightarrow m_2$*, if* $m_1 \rightarrow m_2$ *in* Tr_i *for some honest participant* i*, then* $m_1 \rightarrow m_2$ *in* Tr_j *for every honest participant* j*.*

This definition of TC captures a form of causal ordering of messages: If a message m_1 was received before a message m_2 was sent, then m_2 causally depends on m_1, i.e., $m_1 \rightarrow m_2$, and must be delivered after m_1. Causal ordering only relates any two messages, but does not allow putting every message in relation to any other message, s.t. messages can also be at the same position.

Definition 4 (Total Transcript Consistency)**.** *Consider a group of participants* $\mathcal{P}$ *executing a secure group messaging protocol in the presence of an adversary that controls a subset* f *of Byzantine participants. Each participant* $i \in \mathcal{P}$ *maintains a local* transcript Tr_i*, defined as the sequence of messages that* i *accepts as belonging to the conversation after performing all protocol-defined verification checks.*

The protocol provides Total Transcript Consistency (TTC) *if, for any execution and for any two honest participants* $i, j \in \mathcal{P}$*, the following eventually holds:*

$$\mathsf{Tr}_i = \mathsf{Tr}_j.$$

Equivalently, for any two messages m_1 *and* m_2 *that appear in the transcripts of honest participants, if* m_1 *precedes* m_2 *in* Tr_i *for some honest participant* i*, then* m_1 *precedes* m_2 *in* Tr_j *for every honest participant* j*.*

TTC should capture the informal notion of *global transcript* [36, 67], *transcript agreement* [21] as well as the intuitive meaning described by Marlinspike [39].

A.2 Signal: Inconsistent Duplicate Handling

During our experiments, we noticed additional implementation differences between Signal's mobile app and the desktop client. Duplicates in Signal are usually detected if a message with the same author service ID and client timestamp has been received before. The desktop client, however, also takes the sender's device ID into account, making it possible to receive two messages from different devices with the same sender

service ID and client timestamp pair. This is demonstrated in the artifact video [28].

Another possibility to receive a duplicate message with Signal's desktop client is if the edit of a message has the same timestamp as a message sent after this edit. The message with the reused timestamp is dropped by the official Android client, for example, but accepted by the desktop client, as shown by the artifact video [27].