USENIX Security '26 Artifact Appendix: 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

A Artifact Appendix

A.1 Abstract

This artifact contains the proof-of-concept code used to demonstrate transcript consistency issues in group chats of WhatsApp, Signal, iMessage, and Threema. More specifically, current end-to-end-encryption designs in group chats allow a malicious group participant to send different messages to different group members. The codebase includes modified clients that send inconsistent group messages, and manipulate poll creation, voting, and closing messages. Further, it contains code to demonstrate implementation-specific issues such as fabricated quoted messages and inconsistent duplicate handling.

A.2 Description & Requirements

Since our PoCs have to interact with four different messaging services, we have to use four different client approaches and hence four different software stacks. The most streamlined and easiest to reproduce is probably the Signal-related workflow, as it is completely dockerized. Therefore, we would recommend the evaluator to start there.

A.2.1 Security, Privacy, and Ethical Concerns

Unfortunately we cannot provide test accounts for all respective instant messaging services. We encourage the evaluators to use dedicated test accounts if available instead of their own private accounts. This should prevent accidental exposure of private (and potentially sensitive) inbound instant messages to less secure environments such as the evaluators' test setups configured as linked devices. During testing, we never experienced any countermeasures by the vendors that would result in a block or ban of the involved accounts. This might be due to the fact that (on the server side) most of the TC attacks are indistinguishable from normal messaging behavior.

A.2.2 How to Access

We provide our code on GitHub at `https://github.com/sbaresearch/transcript-consistency` and on Zenodo for long-term archiving at `https://doi.org/10.5281/zenodo.20323807`.

A.2.3 Hardware Dependencies

A working Android and/or iOS device running the according messaging app (primary device) is required to test the described transcript consistency and implementation issues for the respective messaging service.

A.2.4 Software Dependencies

The exact software dependencies are listed in the dependency files in the corresponding artifact subdirectories. To provide a high-level overview, the artifacts require:

- Git and standard build tools for the host operating system.
- Go tooling for the WhatsApp clients based on `whatsmeow`.
- Rust tooling for the iMessage client based on `rustpush`.
- Python and Docker for the Signal client containers based on `Signal-Desktop`.
- Android Studio, the Android SDK, the Android NDK, `bash`, the Protobuf compiler, the Rust toolchain, and a Java/Kotlin build environment for the modified Threema Android debug client, or only Docker for building a release APK using the provided build script.
- Official messenger apps for the receiver devices used in the experiments.

The artifacts do not include credentials, phone numbers, Apple IDs, or Threema licenses. Therefore, reproducing the results requires creating the respective accounts.

A.2.5 Benchmarks

None

A.3 Setup

A.3.1 Installation

Download the archive using the provided link and extract the content to disk. The extracted content should look like this:

- `videos/`: This directory contains PoC videos showing the behavior of official clients in a group with a malicious client.
- `code/`: This directory contains the code of our custom clients for all targeted messengers.
- `code/signal/poc-skdm/`: This directory contains a modified Signal Desktop [2] 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 [2] 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/imessage/`: This directory contains a modified version of rustpush [1] that includes a command-line client to send inconsistent messages into group chats and to manipulate polls and votes.
- `code/threema/threema-android/`: This directory contains a modified Threema Android [3] 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 [4].
- `code/whatsapp/quote-client/`: This directory contains our modified WhatsApp client to manipulate quotes, building on whatsmeow [4].

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

A.3.2 Basic Test

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.

Therefore, switch into either of the `code/signal/{poc-skdm,poc-groups}` directories and execute `./build-container.py` followed by `./start-container.py`. This fire up a container with VNC installed together with a browser from which it can be accessed via `http://localhost:6080/`. In this container the modified Signal desktop can be built and executed with `just run`. Do note that Signal user data will be stored in a Docker volume that has to be manually removed afterwards.

A.4 Evaluation Workflow

A.4.1 Major Claims

(C1): WhatsApp, Signal, iMessage, and Threema do not have transcript consistency in group chats. A malicious group participant can send malicious messages to selected participants without triggering server-side detection or clear recipient-facing UI warnings. This result corresponds to Sections 4 and 5.1, Table 2, and Experiments E1–E3.

(C2): Higher-level interactive group features, in particular polls, inherit and amplify the transcript consistency problem. A malicious poll creator or voter can make honest participants observe divergent poll questions, options, intermediate results, or final results. This corresponds to Section 5.2.3 and Figures 9 and 10, and is evaluated by Experiment E4.

(C3): Implementation-specific behavior further facilitates practical exploitation. This includes fabricated quoted messages in WhatsApp and Signal, spoofed or misleading vote attribution in iMessage, platform-specific WhatsApp poll parsing differences, Signal duplicate handling differences, and WhatsApp OS/device fingerprinting through message IDs. This corresponds to Section 5.3, Table 3, and Appendix A.2 of the paper, and is evaluated by Experiment E5.

A.4.2 Experiments

(E1): Pairwise group message equivocation [45 human-minutes + 20 compute-minutes per messenger]: This experiment reproduces the group transcript inconsistency vector for pairwise E2EE delivery. It applies directly to iMessage and Threema.

Preparation: Create a test group with one attacker account and at least two honest recipients. Build and authenticate the modified client for the selected messenger.

Execution: Use the modified client to send message `M1` to one recipient and message `M2` to another recipient under the same group conversation context. For omission tests, exclude one recipient entirely.

Results: The receiving official clients should show different local transcripts: one recipient sees `M1`, another

sees `M2`, and excluded recipients do not see the message. The clients should not display a clear warning that the group transcript diverged.

(E2): Sender Key fallback and retransmission [45 human-minutes + 20 compute-minutes per messenger]: This experiment reproduces the WhatsApp and Signal vector in which Sender Key delivery failure or retry behavior leads to pairwise retransmission and enables recipient-specific content. For Signal, this experiment requires using the `poc-skdm` client in contrast to all other experiments, which can be built and started in the same way as the `poc-groups` version.

Preparation: Create WhatsApp or Signal test groups and build the corresponding modified client. Ensure that at least two receiving devices are online.

Execution: Use the artifact's Sender Key distribution or fallback controls to cause one recipient to require a retry or retransmission. Respond with a recipient-specific replacement message. For WhatsApp, also evaluate the configuration that hides the transient decrypt-fail UI. For Signal, use the provided recipient selection UI to determine the recipient which will receive the modified message. The client will append hardcoded text.

Results: The targeted recipient should receive the replacement message while other participants retain the original message or no corresponding message. Signal should fail silently; WhatsApp's transient decrypt-fail indicator should disappear after retransmission or be suppressed where the artifact exercises the corresponding attribute.

(E3): Sender Key-only equivocation [45 human-minutes + 20 compute-minutes per messenger]: This experiment reproduces equivocation using Sender Key group messages without delivering the final equivocated content through pairwise sessions.

Preparation: Use the WhatsApp or Signal modified client in server fan-out-only mode and create a test group with at least two recipients.

Execution: Broadcast different versions of a group message so that clients that already accepted one message version ignore later transmissions with the same message identity, while selected recipients receive a different version.

Results: Honest participants should end with divergent transcripts even though the attack uses Sender Key group-message delivery. No participant should receive a clear warning that a contradictory message version was sent to the group.

(E4): Poll manipulation [45 human-minutes + 20 compute-minutes per messenger]: This experiment reproduces the poll attacks from Section 5.2.3.

Preparation: Create a group whose clients support polls. Build the corresponding modified client and create a benign poll to verify normal poll rendering.

Execution: As a malicious poll creator, send different poll questions, option orders, option identifiers, or option text variants to different recipients. As a malicious voter, send different vote messages to different recipients or, for Threema and iMessage, use the service-specific vote manipulation supported by the artifact.

Results: Recipients should observe different poll questions, options, intermediate results, or final results. For index- or identifier-based schemes, option reordering should make a victim's intended vote count for a different option elsewhere. For WhatsApp, insignificant option changes or inverted questions should produce divergent interpretations. For Threema, manipulated close messages can overwrite final results.

(E5): Implementation-specific issues [45 human-minutes + 20 compute-minutes]: This experiment reproduces the additional implementation-specific findings.

Preparation: Select the relevant sub-artifact from the WhatsApp quote client, the Signal modified client, or the iMessage client. Prepare the official receiving clients.

Execution: For quoted-message tests, send a reply that references a missing or future message and supplies attacker-controlled fallback text. For iMessage, send a poll vote JSON with manipulated `participantHandle` values. For Signal, run the duplicate-handling examples and compare them with the supplied videos. If possible, additionally compare WhatsApp poll parsing and message IDs across Android, iOS, Web, Windows, and macOS clients.

Results: WhatsApp and Signal should display attacker-controlled quote fallback text for missing references, with WhatsApp providing no clear warning. iMessage should show misleading vote attribution in the poll preview while the detailed view remains more restrictive. WhatsApp clients should differ in poll parsing behavior across platforms, and WhatsApp message IDs should reveal OS/device-specific prefixes or lengths as described in Table 3.

A.5 Notes on Reusability

Reproducing the exploits in and end-to-end manner requires both multiple accounts (i.e., phone numbers or licenses) on different messaging platforms and different smartphones (i.e., Android, iOS). Unfortunately, we cannot provide this in a scalable way (i.e., for multiple reviewers). As a replacement, we provide pre-recorded videos of the exploits in practice in addition to the PoC source code.

Disclaimer: We are not aware of platform operators detecting and taking action against the provided exploits. However, we strongly recommend only executing the PoC with specific test accounts.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.

References

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

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

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

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