USENIX Security '25 Artifact Appendix: Bots can Snoop: Uncovering and Mitigating Privacy Risks of Bots in Group Chats

Kai-Hsiang Chou*
National Taiwan University

Yi-Min Lin*
National Taiwan University

Yi-An Wang
National Taiwan University

Jonathan Weiping Li
National Taiwan University

Tiffany Hyun-Jin Kim
HRL Laboratories

Hsu-Chun Hsiao†
National Taiwan University
Academia Sinica

A Artifact Appendix

A.1 Abstract

This artifact appendix includes the following components: (1) analysis scripts for the Pushshift Telegram dataset; (2) basic chatbot implementations for various messaging services; and (3) the prototype implementation of SnoopGuard, our proposed secure group messaging protocol. First, we provide the analysis scripts for the Pushshift Telegram dataset, as used in Section 3.2 of our paper. These scripts demonstrate the prevalence of repeatedly encountering chatbots across different groups. Our analysis reveals that when a user joins a new group containing chatbots, there is a 3.6% chance that at least one of the chatbots can recognize and associate the user with their previous interactions in other groups. Second, we include basic chatbot implementations for Discord, Keybase, LINE, Slack, and Telegram, along with tutorials on intercepting packets to and from these chatbots. These implementations help examine the information that chatbots can access and verify whether they support end-to-end encryption, as discussed in Section 4 of our paper. Finally, we provide a prototype implementation of SnoopGuard, our proposed secure group messaging protocol. This implementation is based on existing open-source libraries for the Signal Protocol and Message Layer Security (MLS). Additionally, we include benchmark scripts and the performance evaluation results referenced in Section 6.2.2 and Appendix D of our paper. Our prototype implementation shows that sending a message in a group of 50 users and 10 chatbots takes about 10 milliseconds when integrated with MLS.

A.2 Description & Requirements

Our artifacts include three components: (1) analysis scripts for the Pushshift Telegram dataset; (2) basic chatbot implementations for various messaging services; and (3) the prototype implementation of SnoopGuard, our proposed secure group messaging protocol.

A.2.1 Security, privacy, and ethical concerns

The risks of this artifact align with general expectations and do not pose significant threats to system security, data privacy, or ethical concerns.

A.2.2 How to access

The latest version of the source code is available on GitHub (https://github.com/csienslab/snoopguard-artifact). The specific version used in the main paper can be accessed on GitHub (https://github.com/csienslab/snoopguard-artifact/tree/5649ba9) or on Zenodo (https://zenodo.org/records/14729613).

A.2.3 Hardware dependencies

Pushshift Telegram Dataset Analysis. The data analysis process consists of two parts: (1) data preprocessing, and (2) data analysis. The first step involves downloading the Pushshift Telegram dataset and importing it into PostgreSQL, which requires approximately 1 TB of storage. To simplify reproduction, we have included the processed and extracted data in our source code as several pickle files (Python's object serialization format). These pickle files can be used directly without downloading the full dataset. The second step has no specific hardware requirements.

Basic Chatbot Implementations. No specific hardware requirements.

SnoopGuard Reference Implementation. The implementation requires at least 16GB of RAM to complete the benchmark. The benchmark tests were executed on a Mac mini with an Apple M2 processor and 16 GB of RAM. The provided

*Both authors contributed equally to this research.
†Hsu-Chun Hsiao (hchsiao@csie.ntu.edu.tw) is the corresponding author.

Geekbench script assumes an AArch64 (ARM64) architecture. Running the script on other architectures may require modifications to download compatible versions.

A.2.4 Software dependencies

Pushshift Telegram Dataset Analysis. We use PostgreSQL for data storage and extraction, and Python for writing the analysis scripts. As mentioned earlier, the processed and extracted data are included in our source code as several pickle files to facilitate reproducibility. These pickle files can be used directly, eliminating the need for PostgreSQL.

Basic Chatbot Implementations. Our chatbots are implemented on multiple platforms, including Discord, Keybase, LINE, Slack, and Telegram, using Node.js (v20.18.0) in our experiments. We use Wireshark to monitor HTTP and WebSocket connections and ngrok to create reverse proxies.

SnoopGuard Reference Implementation. Running the implementation requires a Go compiler (v1.21).[1] For resource-constrained benchmarks, Docker is also required.

A.2.5 Benchmarks

None.

A.3 Set-up

Pushshift Telegram Dataset Analysis. The analysis code can be found in the `analyze_pushshift` directory. To reproduce the data preprocessing process, download the Pushshift Telegram Dataset [2]. Set up a PostgreSQL database using the schema provided in `import_data/schema.sql`. Then, import the data using the scripts located in `import_data/*.py` from our repository.

Basic Chatbot Implementations. To execute the reference chatbot implementations, install Node.js, Wireshark, and ngrok. One also needs to apply for chatbot tokens from each platform's website. Detailed setup instructions are available in the README file of each implementation.

SnoopGuard Reference Implementation. Install the Go compiler.

A.3.1 Installation

Pushshift Telegram Dataset Analysis. To reproduce the data export process, run `analysis/dump_data_to_pickle.py` to generate the pickle files. Alternatively, one can use the pre-exported data provided in our repository to skip this step.

Basic Chatbot Implementations. The setup process for each chatbot platform varies. Detailed instructions can be found in the README file for each implementation.

SnoopGuard Reference Implementation. No additional steps are required beyond cloning the repository.

A.3.2 Basic Test

Pushshift Telegram Dataset Analysis. No functionality test is needed.

Basic Chatbot Implementations. Each chatbot is designed to echo any message it receives. To test this, send a message to the group. If the chatbot is set up correctly and receives the message, it will echo the input message.

SnoopGuard Reference Implementation. To test that the environment is set up correctly, run the test scripts using the following commands.:

```
go test ./pkg/user -v -timeout 0
go test ./pkg/chatbot -v -timeout 0
```

If the test case passes, the last line of the output would start with `ok`. All test cases should pass.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Our analysis of the Pushshift Telegram dataset shows that 3.6% of users interact with the same chatbots across multiple groups. This result is demonstrated by experiment (E1) in Section 3.2.

(C2): Our analysis of chatbots on Discord, Keybase, LINE, Slack, and Telegram reveals that platform support for message access control and end-to-end encryption varies. Table 1 provides a detailed comparison. This finding is demonstrated by experiment (E2) in Section 4.1.

(C3): Our reference implementation of SnoopGuard achieves the benchmark results shown in Figures 4 and 5. Specifically, adding a chatbot to a group of 50 members takes about 2 milliseconds, regardless of the protocol. Sending a message to 50 members and 10 chatbots takes about 10 milliseconds when using MLS and about 5 milliseconds with the Sender Keys Protocol. These results are validated by experiment (E3) in Section 6.2.2 and Appendix D.

[1] At the time of writing, some of the packages we used do not support Go v.1.24 or higher.

[2] https://zenodo.org/records/3607497

A.4.2 Experiments

(E1): [Cross-Group Identification Analysis] *[30 human-minutes + 20 compute-hour + 1TB disk]*:
How to: Follow the instructions in Appendix A.3 and the README file to set up the Pushshift Telegram Dataset Analysis.
Preparation: As described in Appendix A.3 and the README file, data preprocessing involves downloading the Pushshift Telegram Dataset, importing it into PostgreSQL, and using the provided script to generate pickle files. Alternatively, one can use the preprocessed pickle files included in the source code to save time and reduce storage requirements.
Execution: Run `analysis/chatbot_analysis.py` to reproduce the results discussed in *Prevalence of Cross-Group Chatbots* (Section 3.2). Run `analysis/bot_user_encounter_ana.py` to reproduce the results from *User-Chatbot Encounters* (Section 3.2).
Results:

- **Prevalence of Cross-Group Chatbots:** The script prints the number of chatbots appearing in multiple groups. The results indicate that 253 chatbots (35.2%) were present in more than one group.
- **User-Chatbot Encounters:** The script `analysis/bot_user_encounter_ana.py` displays the results summarized in the main article at the end of the output. The analysis shows that 42,508 users (3.6%) encountered the same chatbot in multiple groups. Out of 4,155,927 user-chatbot pairs that interacted at least once, 97,813 pairs (2.4%) had multiple encounters.

(E2): [Chatbot Design Analysis] *[3 human-hour]*:
How to: Follow the instructions in Appendix A.3 and the accompanying README file to set up the basic chatbots on various platforms.
Preparation: The setup process for chatbots varies across platforms. Detailed instructions for configuring the chatbots and using Wireshark to intercept HTTP / WebSocket connections can be found in the corresponding README files.
Execution: Once the chatbots are set up, create a group on each platform and add the chatbot to the group. The steps for testing the chatbots' message access control are provided in Section 4.1.
Results: The expected results are shown in the rows *Message access control*, *Group E2EE (w/ bots)*, and *Hide sender* in Table 1. Respective evaluation criteria are listed below:

- Receiving chatbots' echo indicates that it has access to message.
- Plain text intercepted by Wireshark indicates absence of E2EE.
- User identifiers intercepted by Wireshark indicates not hiding sender.

(E3): [SnoopGuard Benchmark] *[6 compute-hour + 16GB RAM]*:
How to: Follow the instructions in the README file.
Preparation: Ensure that the Go compiler is installed.
Execution: Run the script `benchmark_roundtrip.sh` to perform a full benchmark. The execution times reported in the paper are the median values (`p50`).
Results: The output results will be stored in the `benchmark_results/native_roundtrip/`. In the result files, the keyword `IGA` indicates the use of SnoopGuard to achieve selective internal group anonymity (IGA). The keyword `Pseudo` indicates the use of SnoopGuard to achieve pseudonymity, as described in Section 5.5. If no keyword is present, the original protocol (Sender Keys Protocol or MLS) was used. The expected results are shown in Figure 5 and Figure 6.

A.5 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/usenixsec2025/.