Abuse Reporting for Metadata-Hiding Communication Based on Secret Sharing

Saba Eskandarian, *University of North Carolina at Chapel Hill*

https://www.usenix.org/conference/usenixsecurity24/presentation/eskandarian

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

USENIX Security ’24 Artifact Appendix: Abuse Reporting for Metadata-Hiding Communication Based on Secret Sharing

Saba Eskandarian
University of North Carolina at Chapel Hill
saba@cs.unc.edu

A Artifact Appendix

A.1 Abstract

This paper demonstrates that, for broad classes of metadata-hiding schemes, lightweight abuse reporting can be deployed with minimal changes to the overall architecture of the system. Our insight is that much of the structure needed to support abuse reporting already exists in these schemes. By taking a non-generic approach, we can reuse this structure to achieve abuse reporting with minimal overhead. In particular, we show how to modify schemes based on secret sharing user inputs to support a message franking-style protocol. Compared to prior work, our *shared franking* technique more than halves the time to prepare a franked message and gives order of magnitude reductions in server-side message processing times, as well as in the time to decrypt a message and verify a report.

The artifacts accompanying the paper consist of the code for the paper’s evaluation. This artifact appendix contains directions needed to access and run the code for the evaluation in the body of the paper. This evaluation measures the performance of the various algorithms involved when run locally on a single machine.

The provided GitHub repository also contains the source code and evaluation data for the supplemental evaluation contained in the paper’s appendices, as well as the data resulting from our comparison with the results of re-running the code from prior works. The supplemental evaluation includes code to run the algorithms and communicate the results over a network. We do not discuss these materials here.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact does not pose any particular risk to the evaluators’ machines, data privacy, etc., or raise any other notable ethical concerns.

A.2.2 How to access

The artifact can be accessed at the following stable GitHub URL: https://github.com/SabaEskandarian/Shared_Franking/tree/ba9e81644ba9879e4fcfe57d39842b2aa5076f45.

A.2.3 Hardware dependencies

None.

A.2.4 Software dependencies

The only dependencies for running the software are the C standard libraries and OpenSSL (confirmed working on version 3.0.2).

A.2.5 Benchmarks

None.

A.3 Set-up

The code was run on Ubuntu 20.04, but newer versions should work just as well. The evaluator needs to set up the machine for C development and install OpenSSL. Git should be installed for the purpose of accessing the code.

```
sudo apt install git build-essential libssl-dev
```

A.3.1 Installation

Installation simply requires downloading the contents of the provided git repository.

```
git clone [repo_address]
```

A.3.2 Basic Test

Run the following commands for the basic tests.

```
make test
./test
```

The expected result is a message saying that the tests passed.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *Compared to prior work, our shared franking technique more than halves the time to prepare a franked message and gives order of magnitude reductions in server-side message processing times, as well as in the time to decrypt a message and verify a report. This is demonstrated in Figure 6 of our paper, where the running time of shared franking algorithms is displayed next to those of prior work.*

(C2): *Server-side operations on the critical path for message delivery run the fastest and do not noticeably increase in cost as message lengths or number of servers increase. This is demonstrated by Figures 3 and 4 of our paper, where we graph the performance of the different shared franking algorithms with increasing message sizes and numbers of servers, respectively.*

A.4.2 Experiments

(E1): *[Shared Franking Evaluation] [A few minutes]: run the shared franking evaluation, showing compute costs for different message sizes and numbers of servers.*
How to: Run the code described in the execution section below.
Preparation: No additional preparation is needed beyond the Basic Test described above.
Execution: run the following code. The results will be printed and can be sent to a file for later use.

```
make shared_franking_eval
./shared_franking_eval
```

Results: The running time for the various components of shared franking will be run for different message sizes and different numbers of servers, and the results displayed. This contributes to proving claim C1 when combined with repeating the evaluations of the prior works to which we compare. This also proves claim C2 because the increase in the server-side processing algorithms' running time (`p1_mean`, `pis_mean`) will be very small.
Note that since the times involved are so small (on the order of microseconds), systems that have a lot of other programs running will return fairly noisy results. We find the results are more consistent when run on a headless server or a cloud compute instance.
We access and modify the data by putting the output in a csv file for easy viewing and editing in spreadsheet software. The numbers for Figure 3 of the paper come from any row where the number of servers is 2. The numbers for Figure 4 come from the last 9 rows of the file, which reports performance for a 1KB message with a number of servers varying from 2 to 10. The numbers used in Figure 6 come from taking the row with 1KB message sizes and 2 servers from near the end of the output.

(E2): *[Plain Franking Evaluation] [Under a minute]: run the plain franking evaluation, showing compute costs for different message sizes.*
How to: Run the code described in the execution section below.
Preparation: No additional preparation is needed beyond the Basic Test described above.
Execution: run the following code. The results will be printed and can be sent to a file for later use.

```
make plain_franking_eval
./plain_franking_eval
```

Results: The running time for the various components of message franking will be run for different message sizes, and the results displayed. The numbers will be lower than those for the shared franking scheme, as this is a baseline for the scheme used in practice that cannot be applied to the metadata-hiding setting we consider. The numbers used in Figure 6 of the paper come from taking the row with 1KB message sizes.
Note that since the times involved are so small (on the order of microseconds), systems that have a lot of other programs running will return fairly noisy results. We find the results are more consistent when run on a headless server or a cloud compute instance.

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/usenixsec2024/.