USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Comprehensive Deniability Analysis of Signal Handshake Protocols: X3DH, PQXDH to Fully Post-Quantum with Deniable Ring Signatures

Shuichi Katsumata, *PQShield & AIST;* Guilhem Niot, *PQShield & Univ Rennes, CNRS, IRISA;* Ida Tucker and Thom Wiggers, *PQShield*

https://www.usenix.org/conference/usenixsecurity25/presentation/katsumata

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: Comprehensive Deniability Analysis of Signal Handshake Protocols: X3DH, PQXDH to Fully Post-Quantum with Deniable Ring Signatures

Shuichi Katsumata
PQShield & AIST

Guilhem Niot
PQShield, Univ Rennes, CNRS, IRISA

Ida Tucker
PQShield

Thom Wiggers
PQShield

A Artifact Appendix

A.1 Abstract

The Signal protocol relies on a handshake protocol, formerly X3DH and now PQXDH, to set up secure conversations. One of its privacy properties, of value to Signal, is deniability, allowing users to deny participation in communications. Building on Hashimoto et al.'s abstraction for Signal handshake protocols (USENIX'25), our paper proposes a unified framework to analyze the deniability properties of these protocols, including post-quantum variants.

We further analyze post-quantum alternatives like RingXKEM, whose deniability relies on ring signatures (RS). By introducing a novel metric inspired by differential privacy, we provide relaxed, pragmatic guarantees for deniability. We also use this metric to define deniability for RS, a relaxation of anonymity, allowing us to build an efficient RS from NIST-standardized Falcon (and MAYO), which is not anonymous, but is provably deniable.

We implemented in C our proposed ring signature constructions and provided benchmark results demonstrating their concrete efficiency. Our artifact provides these implementations.

A.2 Description & Requirements

This artifact contains the experimental implementations of the ring signatures proposed in our paper, based on the signature schemes Falcon and MAYO. In addition, it includes the implementations of the prior ring signature works Falafl, Calamari and Raptor for performance comparison. The implementations of previous works are extracted from public sources, with minor updates to incorporate our benchmark code.

Our implementations of Falcon-RS and MAYO-RS consist in extending the respective reference C implementation of each base scheme to support signatures for rings of two users, as proposed in our paper. They concretely allow to generate signatures for a ring of two users, that is given one user's secret key and another's public key. They also allow to verify ring signatures given two public keys. We incorporate our benchmark code in each folder to produce the benchmarks of Table 3.

This section lists all the information necessary to recreate the experimental setup used in our paper.

A.2.1 Security, privacy, and ethical concerns

There is no risk introduced by this artifact for evaluators. All the operations are performed locally, in a typical set-up.

A.2.2 How to access

Our artifact can be permanently accessed on Zenodo at DOI 10.5281/zenodo.15571694.

A.2.3 Hardware dependencies

Part of our artifact requires AVX2 instructions support (available on Intel and AMD CPUs). Specifically the ring signatures Calamari and Falafl in the folder *prevworks/Calamari-and-Falafl*.

A.2.4 Software dependencies

For ease of reproducibility, we provide a Dockerfile specification that will compile the code of our artifacts and produce binaries required for our experiments.

The set-up instructions in the following section assume the installation of *podman* to execute our Dockerfile. Note that one may also use their Docker installation (in which case they should replace the *podman* command with *docker* in the instructions provided in the following section).

A.2.5 Benchmarks

Our artifact is used to produce the benchmarks available in Table 3 in our paper of FalconRS and MAYORS. Additionally, the code of previous ring signatures was used to compare the

efficiency of our schemes to prior works in the introduction and at the end of Section 6.

Note that more detailed benchmarks for prior works are available in Table 8 of the full version of our paper[1].

A.3 Set-up

This section includes all the installation steps required to prepare the environment to be used for the evaluation of this artifact.

A.3.1 Installation

1. The evaluator should first install *podman* by following the instructions at https://podman.io/docs/installation for their OS. For instance on Ubuntu, this is done by running `sudo apt-get -y install podman`.

2. Then, the evaluator should download the artifact's code. This can be achieved in two ways:

   (a) By downloading the repository ZIP available at DOI 10.5281/zenodo.15571694, decompressing it, and opening a terminal in the decompressed folder.

   (b) Or using Git (needs to be installed through the OS package manager) and by running in a terminal

   ```
   git clone \
     https://github.com/GuilhemN/ringsign.git
   cd ringsign
   git checkout 7cb6de1
   ```

3. Finally, the evaluator should compile the binaries required for the evaluation using the provided Dockerfile. This is achieved by running the following commands

   ```
   podman build -t rs-c-code .
   podman create --name=rs-c-code-tmp \
     rs-c-code
   podman cp rs-c-code-tmp:/out ./out
   podman rm rs-c-code-tmp
   ```

   Alternatively, instead of copying the binaries out of the container, the evaluator can execute them directly within the container by running:

   ```
   podman build -t rs-c-code .
   ```

   In this case, all benchmark commands in the following sections should be of the form `podman run -rm rs-c-code /out/<binary>` instead of using directly `./out/<binary>`. For example, instead of `./out/calamari-falafl-test_rs_iso`, one would run `podman run -rm rs-c-code /out/calamari-falafl-test_rs_iso`.

A.3.2 Basic Test

The evaluator can perform a simple functionality test by running one of the binaries produced during set-up:

```
./out/calamari-falafl-test_rs_iso
```

After a few seconds, benchmarks should appear for the key generation, signing, and verification. You can simply check for the presence of "keygen cycles", "signing cycles" and "verification cycles".

A.4 Evaluation workflow

A.4.1 Major Claims

Our paper makes the following claim:

(C1): Our ring signatures FalconRS and MAYORS outperform publicly-available implementations of prior ring signatures – Raptor, Calamari and Falafl – by a factor 32 - 66× for signing, and 146 - 1025× for verification. This is proven by the experiments (E1), (E2), (E3), (E4), (E5) described at the end of section 6 of our paper and in section J.3 of our full version[2] whose benchmark results are reported in Table 3 of our paper and Table 8 of our full version.

A.4.2 Experiments

(E1): [Benchmark FalconRS] [10 human-minutes + 1 compute-second]: benchmark the performance of Falcon-512 and FalconRS.
Execution: The benchmark can be executed by running `./out/rs-falcon-speed`.
Results: The evaluator should only collect numbers in the row "512", corresponding to the performance of Falcon-512 and FalconRS. They should compare the cycle counts to the ones provided in Table 3 of our paper (output in cycles by the script, in Megacycles in the paper): Keygen corresponds to the column *kg*, normal Sign corresponds to the column *sd*, 2-ring Sign corresponds to the column *rs_sd*, normal Verify corresponds to the column *vv*, 2-ring Verify corresponds to the column *rs_vv*. It is expected that cycle counts will differ on a different platform, but ratios between two cycle counts should be similar to what we obtained.
For the reference, on our test machine and as reported in Table 3 of our paper, we measured 6.2 Megacycles for

[1] https://eprint.iacr.org/2025/1090.pdf

[2] https://eprint.iacr.org/2025/1090.pdf

the Keygen, 0.26 Megacycles for normal signing, 0.74 Megacycles for 2-ring signing, 0.02 Megacycles for normal verification, 0.04 Megacycles for 2-ring verification.

(E2): [Benchmark MAYORS] [10 human-minutes + 10 compute-second]: benchmark the performance of MAYO 1 and 2 and the corresponding MAYORS.

Execution: The benchmark for MAYO 1 and MAYORS based on MAYO 1 can be executed by running `./out/mayo_bench MAYO_1 500`. The benchmark for MAYO 2 and MAYORS based on MAYO 2 can be executed by running `./out/mayo_bench MAYO_2 500`.

Results: The evaluator should compare the results of the commands with the values in Table 3. The output of the first command is used to fill the row $MAYO_1$ in the Table 3 of our paper, the output of the second command is used to fill the row $MAYO_2$ in Table 3.

The two commands have an output following the same format. They list benchmark results for different functions, with the average cycles count for one execution. We define the correspondence between the name of the functions in the commands output, and the name of the columns in Table 3.

Keygen corresponds to the function *mayo_keypair*, normal Sign corresponds to the function *mayo_sign*, 2-ring Sign corresponds to the function *mayo_rs_sign*, normal Verify corresponds to the function *mayo_verify*, 2-ring Verify corresponds to the function *mayo_rs_verify*. It is expected that cycle counts will differ on a different platform, but ratios between two cycle counts should be similar to what we obtained.

For the reference, on our test machine and as reported in Table 3 of our paper, we measured the following. For MAYO 1, we have 0.24 Megacycles for the Keygen, 0.88 Megacycles for normal signing, 1.1 Megacycles for 2-ring signing, 0.17 Megacycles for normal verification, 0.28 Megacycles for 2-ring verification. For MAYO 2, we have 0.65 Megacycles for the Keygen, 1.1 Megacycles for normal signing, 1.5 Megacycles for 2-ring signing, 0.09 Megacycles for normal verification, 0.16 Megacycles for 2-ring verification.

(E3): [Benchmark Raptor] [10 human-minutes + 5 compute-second]: benchmark the ring-signature Raptor for 2 users and verify claim (C1) that it is outperformed by FalconRS and MAYORS.

Execution: The benchmark can be executed by running `./out/raptor`.

Results: The command outputs **kilo**cycle counts for the key generation (preceeded by "bench keygen"), 2-ring signing (preceeded by "bench sign"), and 2-ring verification (preceeded by "bench verify") of Raptor. The evaluator can compare the observed performance of signing and verification to the 2-ring signing and 2-ring verification of FalconRS and MAYORS obtained in (E1) and (E2) to verify the performance ratios of claim (C1).

For the reference, on our test machine and as reported in the full version of our paper, we measured 27.1 Megacycles for the Keygen, 5 Megacycles for 2-ring signing, 2 Megacycles for 2-ring verification.

(E4): [Benchmark Falafl] [10 human-minutes + 5 compute-second]: benchmark the ring-signature Falafl for 2 users and verify claim (C1) that it is outperformed by FalconRS and MAYORS.

Execution: The benchmark can be executed by running `./out/calamari-falafl-test_rs_lat`.

Results: The command outputs cycle counts for the key generation (preceeded by "keygen cycles"), 2-ring signing (preceeded by "signing cycles"), and 2-ring verification (preceeded by "verify cycles") of Falafl. The evaluator can compare the observed performance of signing and verification to the 2-ring signing and 2-ring verification of FalconRS and MAYORS obtained in (E1) and (E2) to verify the performance ratios of claim (C1).

For the reference, on our test machine and as reported in the full version of our paper, we measured 0.1 Megacycles for the Keygen, 163 Megacycles for 2-ring signing, 76 Megacycles for 2-ring verification.

(E5): [Benchmark Calamari] [10 human-minutes + 30 compute-second]: benchmark the ring-signature Calamari for 2 users and verify claim (C1) that it is outperformed by FalconRS and MAYORS.

Execution: The benchmark can be executed by running `./out/calamari-falafl-test_rs_iso`.

Results: The command outputs follows the same format as for Falafl. It includes the cycle counts for the key generation (preceeded by "keygen cycles"), 2-ring signing (preceeded by "signing cycles"), and 2-ring verification (preceeded by "verify cycles") of Falafl. The evaluator can compare the observed performance of signing and verification to the 2-ring signing and 2-ring verification of FalconRS and MAYORS obtained in (E1) and (E2) to verify the performance ratios of claim (C1).

For the reference, on our test machine and as reported in the full version of our paper, we measured 119.5 Megacycles for the Keygen, 46581 Megacycles for 2-ring signing, 41250 Megacycles for 2-ring verification.

A.5 Notes on Reusability

The code of our ring signatures FalconRS and MAYORS can be reused for research purposes. We included a README in each folder `RS-falcon` and `RS-MAYO` to detail compilation options, and location of code examples to generate keys and ring signatures, as well as verify these signatures in C.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodol-

ogy followed for the evaluation of this artifact can be found at
https://secartifacts.github.io/usenixsec2025/.