USENIX Security '26 Artifact Appendix: SHADOWFAX: Hybrid Security and Deniability for AKEMs

Phillip Gajland
IBM Research Europe – Zurich

Vincent Hwang
MPI-SP
Radboud University

Jonas Janneck
Ruhr University Bochum

A Artifact Appendix

A.1 Abstract

This artifact accompanies the paper **SHADOWFAX: Hybrid Security and Deniability for AKEMs** for hybrid deniable AKEMs. We provide several portable implementations of the hybrid AKEM SHADOWFAX. When instantiated with standardised components (ML–KEM and FALCON), SHADOWFAX yields ciphertexts of 1,728 bytes and public keys of 2,036 bytes, with encapsulation and decapsulation costs of 1.8M and 0.7M cycles on an Apple M1 Pro.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

This artifact simply runs the instantiations of cryptographic primitives (non-destructive). See our paper for ethical concerns.

A.2.2 How to access

This artifact is available from the following two sources:

- Zenodo:
  - Available: https://doi.org/10.5281/zenodo.17975204.
  - Final version: https://doi.org/10.5281/zenodo.18939418.
- Github: https://github.com/vincentvbh/shadowfax.

A.2.3 Hardware dependencies

We do not require any particular hardware for the functionality of the artifact. As for the reproducibility of the performance, numbers in our paper were reported on an Apple M1 Pro.

A.2.4 Software dependencies

For the software, we require the following for functionality:

- `gcc`
- `make`
- `bash`
- `cc`

We benchmark the performance of our instantiations with the following software.

- `gcc (Homebrew GCC 13.3.0) 13.3.0`
- `GNU Make 3.81`
- `GNU bash, version 3.2.57(1)-release (arm64-apple-darwin23)`

`cc` is the same as `gcc`.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

For functionality, install the required software with package managers. For reproducing the performance and the same hardware, make sure the GCC version is the same.

A.3.2 Basic Test

Run the commands in Section A.4.2.

A.4 Evaluation workflow

A.4.1 Major Claims

Our major claims are the performance results in Table 1.

Table 1: Cycle counts (in thousands) of different authenticated key encapsulation mechanisms AKEM and ring signature schemes RSig run on a Firestorm core of an Apple M1 Pro running at 3GHz.

RSig	Unit	Gen	Sgn	Ver
Raptor [2, 3]	kcc	71 420	7 980	505
	ms	23.81	2.66	0.17
GANDALF [FALCON, FALCON]* [1]	kcc	-	-	-
	ms	5.20	0.49	0.02
GANDALF [FALCON, FALCON] (this work)	kcc	12 298	832	97
	ms	4.10	0.28	0.03
GANDALF [ANTRAG, MITAKA]	kcc	12 965	1 021	97
	ms	4.32	0.34	0.03
AKEM	**Unit**	**Gen**	**Enc**	**Dec**
DH-AKEM [X25519]	kcc	227	679	457
	ms	0.08	0.23	0.15
PQ-AKEM [BAT, GANDALF [ANTRAG, MITAKA]]	kcc	24 608	1 150	246
	ms	8.20	0.38	0.08
PQ-AKEM [ML–KEM, GANDALF [FALCON, FALCON]]	kcc	13 130	1 198	225
	ms	4.38	0.40	0.07
SHADOWFAX [X25519, BAT, GANDALF [ANTRAG, MITAKA]]	kcc	24 739	1 837	692
	ms	8.25	0.61	0.23
SHADOWFAX [X25519, ML–KEM, GANDALF [FALCON, FALCON]]	kcc	12 785	1 681	659
	ms	4.26	0.56	0.22

* Our benchmark. In [1], the authors accessed the timing counters through the standard C library on our platform. We benchmark their implementation on our platform. For other implementations, we access the cycle counters through the macOS API with a fallback to assembly for cycle counters on other operating systems. As mentioned before, the implementation of [1] leads to slightly larger signatures (1 288 bytes).

A.4.2 Experiments

For the correctness of the implementations, run the following command:

```
bash ./test_everything.sh
```

The script will build and test the correctness of the implementations. The log will be written to `./log/test_log.txt`.

For benchmarking the implementations, run the following command:

```
bash ./bench_everything.sh
```

The script will build and benchmark the implementations. Root access is required on macOS. The results are written to `./log/bench_log.txt` and converted into LaTeXcommands in `./latex/bench_latex.tex`. There are not much human-time involved. As for the compute-time, it requies few minutes or so on a standard personal computer.

A.5 Notes on Reusability

Our artifact is about portable implementations. For functionality, we also tested in the following environment:

- Apple M1 Pro, Sonoma 14.6.1 (same hardware).
  - `Apple clang version 15.0.0 (clang-1500.3.9.4)`.
  - Same `bash` and `make`.
- Dell Inc. XPS 9320, Ubuntu 22.04.5 LTS.
  - `gcc (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0`.
  - `GNU Make 4.3`.
  - `GNU bash, version 5.1.16(1)-release (x86_64-pc-linux-gnu`.
- Dell Inc. XPS 9320, Ubuntu 22.04.5 LTS (same hardware as above).
  - `Ubuntu clang version 14.0.0-1ubuntu1.1`.
  - Same `bash` and `make` as above.
- MacBook Pro 2020 (Intel(R) Core(TM) i7-1068NG7 CPU @ 2.30GHz).
  - `Apple clang version 12.0.0 (clang-1200.0.32.28)`.
  - `GNU Make 3.81`.
  - `GNU bash, version 5.2.37(1)-release (x86_64-apple-darwin22.6.0)`

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] Shuichi Katsumata, Guilhem Niot, Ida Tucker, and Thom Wiggers. Comprehensive deniability analysis of signal handshake protocols: X3DH, PQXDH to fully post-quantum with deniable ring signatures. Cryptology ePrint Archive, Paper 2025/1090, 2025. To appear in USENIX Security 2025: 34th USENIX Security Symposium.

[2] Xingye Lu, Man Ho Au, and Zhenfei Zhang. Raptor: A practical lattice-based (linkable) ring signature. pages 110–130, 2019.

[3] Zhenfei Zhang. Raptor. https://github.com/zhenfeizhang/raptor, 2020.