USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Oblivious Digital Tokens

Mihael Liskij, *ETH Zurich;* Xuhua Ding, *Singapore Management University;* Gene Tsudik, *UC Irvine;* David Basin, *ETH Zurich*

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

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: Oblivious Digital Tokens

Mihael Liskij, ETH Zurich
Xuhua Ding, Singapore Management University
Gene Tsudik, UC Irvine
David Basin, ETH Zurich

A Artifact Appendix

A.1 Abstract

Our artifact consists of two parts:

(A1): A Tamarin model of the protocol we use to prove that our protocol satisfies the binding integrity property. It consists of an ODT protocol model and a proof that the model satisfies this property.

(A2): The ODT protocol prototype implementation. It consists of an ODT client implementation that uses Intel SGX with OpenSSL, an ODT server implementation based on OpenSSL, various auxiliary scripts, and the raw measurement results. We also provide a Dockerfile and Docker image for easy setup and testing.

Each component comes with a `README.md` document that describes how to setup and use the component.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

One of our scripts for artifact (A2) temporarily disables address space layout randomization (ASLR) until it is completed. However, in case the script is interrupted, ASLR might remain disabled until the computer is restarted. To remedy this, we provide a script `enable_aslr.sh` that the evaluators can use at the end of their evaluation of artifact (A2) to ensure that ASLR is enabled. In case you use the Docker deployment of artefact (A2), this is no longer a potential problem.

A.2.2 How to access

The latest artifact version can be found on GitHub[1] and on Zenodo[2].

A.2.3 Hardware dependencies

Artifact (A1) does not require special hardware and the complete proof can be computed and inspected on commodity hardware with 8 GB of RAM and 10 GB of swap space. It takes up to 30 minutes to compute the proof and up to 30 minutes to open it for inspection, with an `Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz` CPU. The model and the proof itself need only 1 MB of storage space.

Artifact (A2) requires an Intel processor with at least SGX1 capabilities; SGX2 capabilities are not required. We test and develop artifact (A2) on the same device as artifact (A1). After installation, the artifact needs between 5 and 6 GB of space on the filesystem.

A.2.4 Software dependencies

Artifact (A1) requires the Tamarin prover, which is available for MacOS, Linux, and Windows.

Artifact (A2) is developed for Linux. It requires the Intel SGX SDK driver, Intel SGX SDK library, Intel SGX PSW library, the Intel SGX SSL library, and an OpenSSL installation. We provide instructions on how to setup the build tools and the Intel SGX SDK driver. For the rest of the dependencies, we provide a script that allows a user to interactively install them, and a Docker image (and corresponding Dockerfile) where the dependencies are installed automatically. For both options, the dependencies are fixed to specific versions and should continue to install and work in the future. Note that Docker containers use the SGX driver from the host kernel. Therefore, it is mandatory to install the Intel SGX SDK driver on the host system.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

The artifacts can be downloaded from Github and Zotero, as specified in Section A.2.2. Artifact (A1) is stored in the `tamarin_proof` directory and artifact (A2) is stored in the `ODT_implementation` directory.

Artifact (A1) does not need to be installed, but requires the Tamarin prover to inspect it. Installation instructions can be found on the following URL: `https://tamarin-prover.com/manual/master/book/002_installation.html`.

Artifact (A2) must be installed, either manually or through Docker, and requires the Intel SGX driver. The Intel SGX driver is included in the newer Linux kernels, but it must be explicitly installed for older ones. Therefore, we recommend

[1] https://github.com/Anonymous-Usenix-25/Oblivious-Digital-Tokens/releases/tag/usenix

[2] https://zenodo.org/records/14765576

that evaluators update their kernel if possible. For hardware that supports only SGX1, the kernel driver seems to not work, so one must manually install the older driver according to the instructions found at: https://github.com/intel/linux-sgx-driver. For the remaining installation steps, we provide a `setup.sh` script in the `scripts` directory of artifact (A2) that performs the installation process. The installation process takes around 90 minutes on our hardware and asks for superuser privileges for certain installation steps. If an error occurs, the script will be interrupted and you should investigate the cause of the error. As aforementioned, we provide a Docker image as an alternative to running the `setup.sh` script. Once the Docker image is downloaded, or regenerated from the Dockerfile, one can start an interactive session in the image to obtain a test-ready environment.

Note that we provide extended installation and usage instructions for both artifacts in the `README.md` files found in the respective artifacts' directories.

A.3.2 Basic Test

For artifact (A1), calling `tamarin-prover --version` should show no errors. If no errors are shown, then Tamarin prover is installed correctly.

For artifact (A2), the script `heap_verification.sh`, found in the `scripts` directory, performs an ODT verification based on heap measurements. It configures and recompiles the ODT client and server for heap verification and performs an ODT verification. This process takes around ten minutes to complete. If the heap verification was performed successfully, the expected last line of the output is: "ODT verification success".

We also provide a verification test based on stack measurements. To perform it, we provide the script `stack_verification.sh` that configures and recompiles the ODT client and server for stack verification, disables ASLR, and performs an ODT verification. This process takes around ten minutes to complete. If the stack verification was performed successfully, the expected last line of the output is: "ODT verification success" (and an additional message that ASLR was enabled again). The script asks for superuser privileges to disable and enable ASLR.

A.4 Evaluation workflow

A.4.1 Major Claims

We make the following major claims in the paper:

(C1): Our ODT protocol, shown in Figure 5, satisfies binding integrity as described in Section 7.3. This is proven by experiment (E1).

(C2): Our ODT server prototype incurs a slowdown of less than 1 millisecond compared to a normal OpenSSL server. The experiment (E2) is described in Section 6.3.2 and the results are reported in Table 2.

(C3): Our O-TEE implementation incurs a slowdown of around 144 millisecond compared to a normal OpenSSL client. The experiment (E3) is described in Section 6.3.2 and the results are reported in Table 3.

A.4.2 Experiments

(E1): [Tamarin proof] [15 human-minutes + 60 compute-minutes + 1MB disk]:

Preparation: Install Tamarin prover as described in the installation instructions found at https://tamarin-prover.com/manual/master/book/002_installation.html.

Execution: To explore the finished proof, launch Tamarin in interactive mode by calling it with `tamarin-prover interactive .` (the dot is part of the command) in the `tamarin_proof` directory and open the Tamarin Web interface at http://127.0.0.1:3001.

To redo the whole proof, we provide a partially proven ODT model in the `partial-proof.spthy` file, an oracle file `myoracle.py`, and a script `prove.sh`. The `partial-proof.spthy` file contains a proof of the last lemma that we construct by choosing the proof steps ourselves. We do this manually because Tamarin's autoprover has a tendency to get stuck in an infinite loop. Tamarin can prove all other lemmas with the help of the `myoracle.py` file. You can use the `prove.sh` script to invoke Tamarin on the `partial-proof.spthy` file and have it construct the complete proof `proof2.spthy`.

Upon successful completion of the script, Tamarin should say that all lemmas are verified. You can find the script output in `proof2.log`. The last 43 lines of `proof2.log` should roughly match the content of the `expected_log_output.txt` file. Note that `partial-proof.spthy` and `proof2.spthy` are the output of Tamarin's proof process and are not designed for direct inspection. You must use Tamarin's web interface to inspect them.

Results: In the Tamarin Web interface, you can explore any completed proof, e.g., `proof.spthy` or `proof2.spthy`. After clicking on a proof, and after around 30 minutes, the interface will show the names of all successfully proven lemmas in green on the left side. If all the lemmas are green, then this serves as proof for claim (C1). Some lines of the last proof should be red since the last lemma tests for the existence of a solution, so a single complete trace is enough to prove the lemma.

(E2) and (E3): [Prototype performance evaluation] [10 human-minutes + 30 compute-minutes + 1 GB disk]:

While experiments E2 and E3 are described separately in the paper, we bundle them together for convenience.

Preparation: If not yet complete, install the Intel SGX SDK driver and run the `setup.sh` script found in the

`scripts/` directory. The script installs all the necessary Intel SGX libraries and the ODT client and server. You can also use the provided Docker image where the `setup.sh` script is already complete. Also, make sure that you do not have any CPU intensive processes running while the measurements are performed.

Execution: To perform the measurements, run the `runtime_measurement.sh` script. This will configure and recompile the ODT client and server for runtime measurement. Afterwards, it will start various combinations of ODT and unmodified OpenSSL clients and servers, and perform six timing measurements in total with 1000 samples each. During each measurement, the script outputs the index of the current sample.

Results: To read the results, we provide the `analyze_results.py` script. It formats the results in the form of Table 2 and Table 3 from the paper to help evaluate claims (C2) and (C3). The raw results are stored in the `scripts/results/` directory.

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/.