ZKSMT: A VM for Proving SMT Theorems in Zero Knowledge

Daniel Luick, John C. Kolesar, and Timos Antonopoulos, *Yale University;*
William R. Harris and James Parker, *Galois, Inc.;* Ruzica Piskac, *Yale University;*
Eran Tromer, *Boston University;* Xiao Wang and Ning Luo, *Northwestern University*

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

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: ZKSMT: A VM for Proving SMT Theorems in Zero Knowledge

Daniel Luick
Yale University
daniel.luick@yale.edu

John C. Kolesar
Yale University
john.kolesar@yale.edu

Timos Antonopoulos
Yale University
timos.antonopoulos@yale.edu

William R. Harris
Galois, Inc.
bll.hrris@gmail.com

James Parker
Galois, Inc.
james@galois.com

Ruzica Piskac
Yale University
ruzica.piskac@yale.edu

Eran Tromer
Boston University
tromer@bu.edu

Xiao Wang
Northwestern University
wangxiao@northwestern.edu

Ning Luo
Northwestern University
ning.luo@northwestern.edu

A Artifact Appendix

A.1 Abstract

This artifact contains an implementation of ZKSMT, a zero-knowledge protocol for validating proofs of SMT theorems. It also contains benchmarks and scripts for reproducing our experimental results. ZKSMT is implemented primarily in C++, but our artifact also includes scripts written in OCaml and Python 3.

In addition to the main zero knowledge checker, we also provide a compiler for converting proofs from SMTInterpol into our format. We do not include SMT-Interpol itself in our artifact, but it is publicly available at https://ultimate.informatik.uni-freiburg.de/smtinterpol/online/proof.html.

The artifact also includes a plaintext version of ZKSMT that validates proofs in the same format but does not perform any cryptographic operations. In the evaluation for the paper, we used the plaintext version of ZKSMT only within Cheesecloth. The main test scripts that we provide in the artifact do not use the plaintext version of ZKSMT.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

All of the benchmarks in our evaluation are public and open-source. The proofs that we use for the benchmarks come from SMTInterpol, which is also open-source.

A.2.2 How to access

Our source code is publicly available as a GitHub repository at the commit https://github.com/PP-FM/ZKSMT-pub/tree/Usenix2024.

A.2.3 Hardware dependencies

For the evaluation results reported in the paper, we ran ZKSMT on AWS instances of type `r5b.4xlarge` with 128 GB of memory, 16 vCPUs, and a 10 Gbps network connection between the prover and verifier. However, the underlying ZK protocols that we use only consume about 100 Mbps bandwidth. We also configured ZKSMT to use 8 threads. Our artifact does not need to be run on an AWS instance, but it does require a very large amount of RAM.

For our reported running times, we ran the prover and verifier on distinct machines, but it is also possible to run both ends on a single machine. It would need to be a machine with at least 256 GB of memory, such as an AWS instance of type `r5b.8xlarge`, to run the full benchmark suite on its own, but 128 GB is sufficient for all benchmarks except the final one. In testing, we did not notice any significant difference in performance, either positively or negatively, between running on one machine or two. We recommend reproducing our results on one machine if possible, as this eliminates the difficulties of establishing a connection between two machines.

Because the WiSA benchmarks are much more time-consuming than everything else in our evaluation, we also provide a script `limited.py` that runs everything except the WiSA benchmarks. The script requires only 4 GB of memory if run on a single machine.

A.2.4 Software dependencies

ZKSMT was developed on Ubuntu 22.04, though later versions of Ubuntu will likely work as well. We use Docker to provide a reproducible build environment.

The zero knowledge checker itself depends on EMP-zk, a library for efficient and interactive zero-knowledge proofs. EMP-zk in turn depends on libntl, a high-performance number theory library. Other dependencies include Python 3, g++, CMake, and matplotlib for creating plots. See the provided Dockerfile for all dependencies.

The process of generating the proofs that ZKSMT validates is not part of our main research contributions, but we include the code for proof generation in our artifact. Our proofs are generated using SMTInterpol and compiled using a compiler provided in the ZKSMT repository. The compiler depends on OCaml, its Dune build system, and several packages installed through the opam package manager.

More detailed installation instructions are in Section A.3.

A.2.5 Benchmarks

Our main benchmarks come from the tests for the Boogie verification language. We stress test ZKSMT with benchmarks derived from WiSA, the Wisconsin Safety Analyzer, as provided in the SMT-LIB benchmark suite. We also run tests on each individual rule that is part of ZKSMT's rule set, using stub proof files generated by our compiler.

For our baseline comparison, we ran a subset of the Boogie benchmarks on a plaintext version of our checker, located in the `checkers` directory in our repository, with the Cheesecloth tool. We do not include the code for reproducing this part of our results in the artifact. The results from running Cheesecloth are stored in `zkchecker/cycle_count.txt` in our repository.

A.3 Set-up

A.3.1 Installation

We have two scripts for the initial setup. The script `install_docker.sh` installs Docker. After Docker has been installed, the script `setup_container.sh` can be run to create a Docker image with all of the required dependencies based on a provided Dockerfile. See steps 1 and 2 in the README.

A.3.2 Basic Test

For a basic test, we provide the script `simple.py`, described in step 3 of the README. The script runs only a single Boogie benchmark. If the script succeeds, it should output timing information for various sections of the algorithm and for specific rules, and it should print `check complete` at the end.

Just like the other test scripts, `simple.py` can be run on either one machine or two. For the multi-machine approach, the script can be used to ensure that the two machines can communicate successfully.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): ZKSMT can validate SMT proofs involving Boolean logic, equality with uninterpreted functions (EUF), and linear integer arithmetic (LIA) efficiently in zero knowledge (§7.1, Figure 2).

(C2): The running times of our individual proof rules scale linearly relative to the maximum list length in a proof. Other size parameters, such as the number of expressions in the expression table, are irrelevant in comparison (§7.2, Figure 5).

(C3): Our compiler does not enlarge proofs excessively when it translates them into our ZK format (§7.2, Figure 4). At most, the number of proof steps in the ZK version of a proof is 7 times larger than the number of proof steps in the original proof from SMTInterpol.

(C4): ZKSMT can validate extra-large SMT proofs (§7.2, Figure 6a).

(C5): ZKSMT is significantly faster than Cheesecloth, a general-purpose tool for executing C-like programs in zero knowledge (§7.3, Figure 6c). Though the exact runtime ratio may differ due to variations in startup time, ZKSMT is consistently faster by multiple orders of magnitude.

A.4.2 Experiments

(E1): Full results (7 compute-hours, 256 GB RAM or 128 GB RAM each on two machines):
After setting up a Docker container as described in Section A.3, see `full.py` in step 3 of the README. This reproduces all major claims. After running and extracting output from the Docker container, the figures mentioned in A.4.1 will be available in a new directory named `out` in the repository.

(E2): Limited results (1.3 compute-hours, about 4 GB RAM):
Same as E1, but run the `limited.py` script instead. This reproduces all major claims except C4.

A.5 Notes on Reusability

Though the most important feature of our artifact is the zero knowledge checker itself, our repository contains other programs. One of them is a compiler for generating ZKSMT proofs from the RESOLUTE format generated by SMTInterpol, with a few tweaks. The compiler allows ZKSMT to be run on proofs not included in the repository. Note that the compiler only supports the theories that the implementation of our ZK checker supports: for instance, it cannot translate proofs about quantifiers or non-linear arithmetic.

Additionally, there is a plaintext version of the zero knowledge checker, and the zero knowledge checker can be run independently of the benchmarking scripts. See the READMEs in their respective directories for more information.

Though the memory requirements for the WiSA benchmarks are quite high, the script `limited.py`, which runs everything except the WiSA benchmarks, works on machines with at least 4 GB of memory. Furthermore, most WiSA benchmarks other than the final largest benchmark require significantly less than 256 GB of memory. If running the full benchmark suite is prohibitively costly, the test scripts can be modified to skip some of the benchmarks. The names of specific WiSA benchmarks can be added to the variable `wisa_skip_list` in `zkchecker/benchmark.py`, and `full.py` will skip them when run afterward; the plotting in `zkchecker/plot.py` is designed to work even if certain WiSA benchmarks are missing.

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/usenixsec2024/.