

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Leveraging Cryptographic Simulator Synthesis for Formally Verifying the FOO E-Voting Protocol

David Baelde, *Univ Rennes, CNRS, IRISA;* Adrien Koutsos and Justine Sauvage, *Inria*

https://www.usenix.org/conference/usenixsecurity26/presentation/baelde

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: Leveraging Cryptographic Simulator Synthesis for Formally Verifying the FOO E-Voting Protocol

David Baelde
Univ Rennes, CNRS, IRISA

Adrien Koutsos
Inria

Justine Sauvage
Inria

A Artifact Appendix

This appendix describes the artifacts associated to the paper *Leveraging Cryptographic Simulator Synthesis for Formally Verifying the FOO E-Voting Protocol*, and explains how to reproduce the experiments presented in that paper.

A.1 Abstract

The artifacts are:

- the source code of the Squirrel proof assistant with our modified version of the `crypto` tactic;
- the Squirrel proof scripts for the paper's running example and case studies;
- the Squirrel proof scripts for the proof of vote privacy for the FOO e-voting protocol;
- static HTML files allowing to follow Squirrel runs on all proof scripts, step by step, without having to build and run the tool.

A.2 Description & Requirements

All files are available at the following URL:

`https://doi.org/10.5281/zenodo.17880702`

External software dependencies are freely available online, mostly on standard package managers.

Our code should run on any modern machine. It has been tested on standard laptops with Intel i7 and AMD Ryzen 7 CPUs with 30Go RAM. It should build and run on any standard Linux or MacOS distribution; it has been tested on Debian 13, Ubuntu 20.04.6 LTS, MacOS 26.2 and 15.7.3.

The artifacts have been integrated into the main line of development of the Squirrel proof assistant, i.e. the `master` branch of Squirrel's public repository[1]. We expect that our contributions will remain available in that repository, where they will be maintained and kept compatible with future evolutions of both Squirrel and its external dependencies. If one does not need to work with the exact archived version of our artifacts, it might become more practical to access and experiment with our contributions through the Squirrel repository. Note that the present document describes the software dependencies and build instructions for the archived artifact only; one should refer to the `README.md` file in Squirrel's repository for up-to-date information on how to build the current system.

[1] `https://github.com/squirrel-prover/squirrel-prover/`

A.2.1 Security, privacy, and ethical concerns

We do not identify any risk associated to building and running Squirrel on proof scripts. Squirrel itself does not modify or create files, and it does not communicate on the network.

A.2.2 Hardware dependencies

None.

A.2.3 Software dependencies

Our code relies on the following software (the provided versions are known to work, but slightly different versions should also work unless otherwise stated):

- `opam` 2.3.0 (any version above 2.0.0 should work)
- `ocaml` 5.4.0 (any version above 5.0.0 should work)
- `dune` 3.21.0
- `fmt` 0.11.0
- `ocamlgraph` 2.2.0
- `alcotest` 1.9.1
- `sedlex` 3.7
- `ppx_inline_test` v0.17.1
- `menhir` 20250912
- `menhirLib` 20250912

- `conf-which` 1
- `zarith` 1.14
- `why3` 1.8.2 (other versions are likely to break)
- `cvc5` 1.0.8 (other versions may not work)
- `z3` 4.12.2 or 4.13.2 (other versions may not work)

Most of these packages are installed automatically (with appropriate versions) using opam and a provided opam configuration file, as explained below.

Squirrel is generally used with Emacs Proof General to develop proof scripts, but can be used as a standalone tool to verify existing scripts. Therefore, this appendix does not include dependencies and instructions for Proof General. The interested reader may find details on how to setup Proof General with Squirrel in the `squirrel/README.md` file.

A.2.4 Benchmarks

Experimental results from the paper only rely on the proof scripts contained in the artifact.

A.3 Set-up

1. Install `opam` using a standard package manager, e.g. by running `apt-get install opam`.
2. Run `opam init` to set it up.
3. Run `opam switch create squirrel 5.4.0` to create a Squirrel switch based on OCaml 5.4.0.
4. Run `eval $(opam env --switch squirrel)` to sync your environment with the switch.

A.3.1 Installation

First install SMT solvers and Why3:

1. Run `opam install why3` to obtain the optional why3 package.
2. Get the binaries for CVC5 and Z3 from the official websites https://github.com/cvc5/cvc5/releases/tag/cvc5-1.0.8 and https://github.com/Z3Prover/z3/releases/tag/z3-4.12.2.

   Make sure the binaries are executable and available in your PATH under the names `cvc5` and `z3` — they may simply be copied to any directory in the path, as they do not rely on installed files.
3. Configure Why3 by running `why3 config detect`. Make sure CVC5 and Z3 are found.

Then, install dependencies and build Squirrel by running the next commands from Squirrel's source code directory, e.g. the `squirrel/` subdirectory of the artifact:

1. Run `opam install . -y --deps-only` to install opam packages required for Squirrel;
2. Run `make` to build Squirrel.

A.3.2 Basic Test

From the toplevel directory of the artifact, running `squirrel/squirrel` should display a short help message.

Running `squirrel/squirrel proofs/motivating.sp` should display a long output ending with "Goodbye!" before exiting with error code 0 in a few seconds.

To test that Squirrel is fully functional with SMT support, run `squirrel/squirrel proofs/kdf.sp`. It should display a long output ending with "Goodbye!" as before, in less than 30 seconds.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The improved crypto tactic included in our version of Squirrel runs as intended, implicitly synthesizing simulators with memoization. This is demonstrated in the Squirrel proof scripts `proofs/*sp`.

(C2): Our modified version of Squirrel allows to prove vote privacy for FOO. This is demonstrated by the Squirrel proof script `proofs/foo/main.sp` which relies on other scripts in the same subdirectory.

(C3): The execution times for these files are reasonable, as shown in Table 1 of our paper.

A.4.2 Experiments

(E1): Run Squirrel on all files for **(C1)** and **(C2)** to check that the proof scripts work as intended, establishing the claimed results.

This experiment takes less than 1 human-minute to launch and verify. It should take less than 5 computer-minutes.

How to: A Makefile is available to run Squirrel on all files. Alternatively, the testers may run Squirrel manually as shown above on individual files.

Preparation: Switch to the `proofs` subdirectory.

Execution: Run `make`.

Results: The output should show OK for each file.

(E2): Execution times for individual files may be recomputed by running Squirrel manually and timing it using the `time` utility.

This can be done easily, as described below, for files in `proofs/*.sp`. For files in `proofs/foo/*.sp`, one first needs to manually change the first lines

by replacing each `include <module>` directive by `include[admit] <module>`. This is to avoid that Squirrel checks the proofs in the included modules, to avoid counting their execution times multiple times.
Preparation: Switch to the `proofs` subdirectory.
Execution: Run the following command for each file: `time ../squirrel/squirrel <file.sp>`.
Results: The execution time is displayed.

(E3): Execution times of individual calls to the `crypto` tactic may be obtained by using Squirrel's internal `time` utility. To do so, occurrences of the `crypto` tactic in the files should be prefixed by `time`. For instance, in `ccapk1.sp` there are two calls to `crypto` as reported in Table 1, which should be modified as shown in Figure 1.
The execution time of each instrumented tactic will be shown at the end of the tactic's output. That output can be obtained by running Squirrel in interactive mode from Emacs' Proof General, and jumping to the relevant position in the proof script. Timing outputs can also be collected in batch using only the standalone tool as shown in Figure 1 for our example file.

A.5 Notes on Reusability

As mentioned above, our artifacts have been integrated in Squirrel's official repository, to ensure that they are maintained in the future and to facilitate their use e.g. as benchmarks for future works.

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/.

```
$ grep -n -C 3 "time crypto" foo/ccapk1.sp
194-  [happens(t)] -> equiv(frame@t).
195-Proof.
196- intro *.
197: time crypto ~no_subgoal_on_failure ~time_sensitive CCA2 (key:sk_mix1).
198- - smt ~no_macros.
199- - smt ~no_macros.
200- - smt ~no_macros.
--
351-[happens(t)] -> equiv(frame@t).
352-Proof.
353- intro *.
354: time crypto ~no_subgoal_on_failure ~time_sensitive CCA2 (key:sk_mix1).
355- - smt ~no_macros.
356- - smt ~no_macros.
357- - smt ~no_macros.

$ ../squirrel/squirrel foo/ccapk1.sp 2> /dev/null | grep time:
[dbg>time: 3.917882 <][goal> Focused goal (1/26):
[dbg>time: 4.021414 <][goal> Focused goal (1/26):
```

Figure 1: Instrumentation and measure of individual tactics