# USENIX'23 Artifact Appendix: Cheesecloth: Zero-Knowledge Proofs of Real-World Vulnerabilities

Santiago Cuéllar* Galois, Inc.
Bill Harris Galois, Inc.
James Parker Galois, Inc.
Stuart Pernsteiner Galois, Inc.
Eran Tromer Columbia University

## A Artifact Appendix

### A.1 Abstract

This artifact accompanies the paper, *Cheesecloth: Zero-Knowledge Proofs of Real-World Vulnerabilities*. It contains pointers to the software repository for the work described in the paper, instructions for compiling the software and its dependencies, and instructions for running the benchmarks in the paper.

### A.2 Description & Requirements

#### A.2.1 Security, privacy, and ethical concerns

There is no risk for evaluators.

**Ethical concerns** CHEESECLOTH aids in responsible disclosure by producing zero-knowledge proofs of the existence of vulnerabilities while keeping the vulnerabilities and exploits secret. All vulnerabilities used in our evaluation have been previously disclosed publicly, and fixes are widely deployed. Thus, the work presented in the paper does not constitute an unethical disclosure of potentially harmful information. A black hat researcher could use CHEESECLOTH as part of the process to sell a vulnerability, however CHEESECLOTH's involvement is unlikely to change the fact that the vulnerability will still be sold and abused.

#### A.2.2 How to access

The source code accompanying this paper is available at https://github.com/GaloisInc/cheesecloth/tree/usenix-2023-artifact.

#### A.2.3 Hardware dependencies

All measurements reported in the paper were performed on a 128 core Intel Xeon E7-8867 CPU with 2 TB of RAM, although our implementation uses considerably less memory. 592 GB of disk space is required to store the outputs of all benchmarks. The OpenSSL benchmark takes up most of this disk space. The GRIT and FFmpeg benchmark outputs combined require less than 35 GB.

#### A.2.4 Software dependencies

Benchmarks were run on Debian 11. LLVM version 9 is a required dependency. The Haskell (`stack`) and Rust (`cargo`) development tools are also required to build CHEESECLOTH. All other Haskell and Rust dependencies are listed in project configuration files and are automatically retrieved using `stack` and `cargo`.

#### A.2.5 Benchmarks

Experiments require the vulnerable versions of GRIT, FFmpeg, and OpenSSL. All vulnerable versions are provided as git submodules of the CHEESECLOTH repository and point to the appropriate commit.

### A.3 Set-up

#### A.3.1 Installation

LLVM version 9, `stack`, and `cargo` are dependencies that must be installed. The two main components of the CHEESECLOTH compilation chain are `MicroRAM` and `witness-checker`, which both live in git submodules. For convenience, the scripts `./scripts/build_microram` and `./scripts/build_witness_checker` will compile each tool.

A `Dockerfile` is provided for reproducible builds, however Docker may add too much overhead in practice. You can build and run the Docker container with:

```
docker build --platform linux/x86_64 -f
    cheesecloth/Dockerfile -t
    cheesecloth-image .
docker run --platform linux/x86_64 -it
    cheesecloth-image:latest /bin/bash
```

#### A.3.2 Basic Test

Correctness tests for `MicroRAM` and `witness-checker` can be run with `stack test` in the `MicroRAM` directory and `cargo test` in the `witness-checker` directory.

*Authors listed alphabetically.

## A.4 Evaluation workflow

### A.4.1 Major Claims

**(C1):** CHEESECLOTH produces Zero-Knowledge proofs of vulnerabilities for GRIT, FFmpeg, and OpenSSL with multiplication gate counts as described in Table 1 of the paper.

**(C2):** Turning off the sparsity and public-pc segment optimizations increases circuit size in terms of multiplication gate counts as described in Table 2 of the paper.

### A.4.2 Experiments

For each of the experiments, running GRIT and FFmpeg takes hours, while OpenSSL takes days. You may wish to skip the OpenSSL runs.

**(E1):** Generate the ZK proofs of vulnerabilities for GRIT, FFmpeg, and OpenSSL.

**How to:** Once all the dependencies are installed, run the following scripts to generate the ZK proofs:

```
./scripts/run_grit && mv out/grit
    out/grit_baseline
./scripts/run_ffmpeg && mv out/
    ffmpeg out/ffmpeg_baseline
./scripts/run_openssl && mv out/
    openssl out/openssl_baseline
```

**Results:** All of the scripts should finish successfully and the corresponding output directories will contain a `witness-checker.log` file. This file contains the multiplication gate counts at the `"mul_gates"` key (under `"gate_stats"`) which correspond to Table 1.

**(E1.5):** Run the ZK proofs from E1 through the Diet Mac'n'Cheese ZK proof backend.

**How to:** While our contribution is agnostic to the ZK backend, you can run the Diet Mac'n'Cheese backend with the following scripts (you will need to update `swanky_dir` in the script to point to the location of Diet Mac'n'Cheese on your file system):

```
./scripts/dmc.sh verifier out/
    grit_baseline &
./scripts/dmc.sh prover out/
    grit_baseline
```

You may want to invoke the prover and verifier in separate terminals.

**Results:** The scripts will report the backend protocol time for the given ZK proof (replace `prover` with `prover-count` to report protocol communication).

**(E2):** Regenerate the ZK circuits with the sparsity and public-pc segment optimizations turned off.

**How to:** Run the following scripts to regenerate the ZK proofs:

```
./scripts/run_grit_no_sparsity && mv
    out/grit out/grit_no_sparsity
./scripts/run_grit_no_publicpc && mv
    out/grit out/grit_no_publicpc
./scripts/run_ffmpeg_no_sparsity &&
    mv out/ffmpeg out/
    ffmpeg_no_sparsity
./scripts/run_ffmpeg_no_publicpc &&
    mv out/ffmpeg out/
    ffmpeg_no_publicpc
./scripts/run_openssl_no_sparsity &&
    mv out/openssl out/
    openssl_no_sparsity
./scripts/run_openssl_no_publicpc &&
    mv out/openssl out/
    openssl_no_publicpc
```

**Results:** All of the output directories will contain a `witness-checker.log` file again with the multiplication gate counts which correspond to Table 2.

## A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.