USENIX'23 Artifact Appendix
Curve Trees: Practical and Transparent Zero-Knowledge Accumulators

Matteo Campanelli
Protocol Labs
matteo@protocol.ai

Mathias Hall-Andersen
Aarhus University
ma@cs.au.dk

Simon Holmgaard Kamp
Aarhus University
kamp@cs.au.dk

A Artifact Appendix

A.1 Abstract

We provide a Rust implementation of Curve Trees instantiated with Bulletproofs. The repository includes an implementation of the *select and rerandomize* primitive, which proofs that a commitment is a rerandomization of a commitment in a committed set. This primitive is then used to construct an accumulator as well as a simple anonymous payment system.

A.2 Description & Requirements

Our results were produced using an AWS C6i.2xlarge instance with 16GB of RAM and 8 vCPUs. This corresponds to 4 physical cores on an Intel Xeon 8375C processor with 2.9 GHz clock speed. The benchmarks were compiled using version 1.68.0 of the rust compiler. We found similar performance on our laptops. The field arithmetic is optimized for newer x86_64 chips, but the code will still work on other architectures.

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

The code is available at https://github.com/simonkamp/curve-trees/tree/4467be81737732a5b2794b5ad70459681b3bd19c.

A.2.3 Hardware dependencies

Any system with access to the internet and a rust compiler and standard library should be able to run the artifact.

A.2.4 Software dependencies

The only software dependency is the rust compiler, which can be downloaded from https://rustup.rs/.

A.2.5 Benchmarks

None.

A.3 Set-up

Install the rust compiler (https://rustup.rs/). Install jq command for formatting output (https://jqlang.github.io/jq/).

A.3.1 Installation

Clone the repository.

A.3.2 Basic Test

Run `cargo build`.

A.4 Evaluation workflow

After the set-up above, verify the functionality of the artifact by running the tests:

```
cargo test --release
```

A.4.1 Major Claims

(C1): The implementation of the select and rerandomize primitive matches the performance reported in table 1 of our paper. This is proven by the experiment (E1).
(C2): The implementation of the accumulator matches the performance reported in table 2 of our paper. This is proven by the experiment (E2).
(C3): The implementation of the $\mathbb{V}$Cash matches the performance reported in table 3 of our paper. This is proven by the experiment (E3).

A.4.2 Experiments

Each experiment will run benchmarks of the proving and verification time for the set sizes and curves reported in the relevant table of our paper.

(Before experiments): *[All tables] [1 human-minutes + 30 compute-minutes]:* We recommend using the script `gen_fmt_estimates.sh` to run all the benchmarks. The output can be reprinted using `fmt_estimates.sh`.

(E1): *[Table 1] [5 human-minutes + 0 compute-minutes]: Demonstrates proof size, proving and verification times for select-and-rerandomize of Curve Trees (batching and non-batching case) in different curves and on different parameters (Table 1). The results are found under "Table 1 (Accumulator)" in the output of* `fmt_estimates.sh`.

(E2): *[Table 2] [5 human-minutes + 0 compute-minutes]: Demonstrates proof size, proving and verification times for accumulators from Curve Trees in different curves on sets of size* 2^{30} *(Table 2). The results are found under "Table 2 (SelectAndRerand)" in the output of* `fmt_estimates.sh`.

(E3): *[Table 3] [5 human-minutes + 0 compute-minutes]: Demonstrates transaction size, proving and verification times for* $\mathbb{V}$*Cash (Table 3). The results are found under "Table 3 (Pour)" in the output of* `fmt_estimates.sh`.

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/.