Aardvark: An Asynchronous Authenticated Dictionary with Applications to Account-based Cryptocurrencies

Derek Leung, *MIT CSAIL;* Yossi Gilad, *Hebrew University of Jerusalem;*
Sergey Gorbunov, *University of Waterloo;* Leonid Reyzin, *Boston University;*
Nickolai Zeldovich, *MIT CSAIL*

https://www.usenix.org/conference/usenixsecurity22/presentation/leung

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

A Artifact Appendix

A.1 Abstract

The artifact is an implementation and empirical evaluation of Aardvark, an authenticated dictionary.

The artifact contains two sets of benchmarks for evaluation in the paper. First, it contains microbenchmarks of vector commitment operations which compare those used in the paper with those in EDRAX (a related system), and with a basic Merkle Tree. Second, it contains a benchmark of the dictionary operations themselves from the perspective of both a validator and an archive, with the dictionary integrated into the backend of the Algorand cryptocurrency. The objective of these benchmarks is to substantiate the paper's claims of computational efficiency, which is difficult to analytically evaluate. In particular, these benchmarks measure the latency of key vector commitment and dictionary operations.

The artifact may be validated by downloading it from the public GitHub repository URL provided and running the evaluation scripts, which are part of the repository. The expected result of artifact evaluation is that the latency measurements match those in the paper.

A.2 Artifact check-list (meta-information)

- **Algorithm:** Authenticated dictionary
- **Program:** Custom benchmarks, included
- **Compilation:** g++ 9.3.0, rustc 1.54.0-nightly (126561cb3 2021-05-24), go 1.16.4
- **Metrics:** Latency
- **Output:** File, measured characteristics, expected result included
- **Experiments:** OS Scripts
- **How much disk space required (approximately)?:** 1GB
- **How much time is needed to prepare workflow (approximately)?:** 6hrs
- **How much time is needed to complete experiments (approximately)?:** 13hrs
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** MIT, GPLv3
- **Archived (provide DOI)?:** Yes, https://github.com/derbear/aardvark-prototype/tree/dd8f6aaf5f76173118f3f3decbe099bda5972ce2

A.3 Description

A.3.1 How to access

Clone the repository and its submodules from GitHub at the following URL: https://github.com/derbear/aardvark-prototype/tree/dd8f6aaf5f76173118f3f3decbe099bda5972ce2. For instance, run

```
git clone --recurse-submodules \
https://github.com/derbear/aardvark-prototype.git
git checkout dd8f6aaf5f76173118f3f3decbe099bda5972ce2
```

EDRAX and its dependencies are under the `edrax` subdirectory, the implementations of vector commitments and Merkle trees are under the `veccom-rust` subdirectory (which depends on the `pairing-fork` subdirectory), and the Algorand implementation resides in the `go-algorand` subdirectory with the Aardvark implementation in `go-algorand/ledger`.

The `--recurse-submodules` option initializes the repositories to their correct versions. The commits corresponding to this document's version of the artifact for the top-level repository, `veccom-rust`, and `edrax` are all additionally labelled `usenix22-artifact` through `git tag`. To confirm that the versions of all submodules are correct, run `git submodule status --recursive` from `aardvark-prototype`, which should produce the following hashes.

```
1f1a3748d1530da1e75fadbce987ee6e6fa3fd1d edrax
530223d7502e95f6141be19addf1e24d27a14d50
 edrax/ate-pairing
a34850b2df66a186c8d947b4d72acc839926321f edrax/xbyak
cff079d3f78daa48d25183292960c21da9cdf152 pairing-fork
d72ed3c8b0e4624053360591fcc8d03ce720ae90 veccom-rust
```

If you did not supply the `--recurse-submodules` option above, you can alternatively initialize these submodules by running the following command from `aardvark-prototype`.

```
git submodule update --init --recursive
```

A.3.2 Hardware dependencies

To reproduce results regarding the authenticated dictionary's scalability, 32 cores are required. The provided benchmarking script in the repository assumes the presence of at least 64 cores.

Around 110MB of disk space is required to clone the entire git repository. Around 1GB of disk space is required to run the experiments.

A.3.3 Software dependencies

Building the software depends on the compilers `g++ 9.3.0`, `rustc nightly-2021-05-25`, and `go 1.16.4`; on the `libgmp3` library; and on the build tools `cmake`, `make`, `autoconf`, `automake`, and `libtool`. Running benchmarks depends on `numactl`.

A.3.4 Data sets

N/A

A.3.5 Models

N/A

A.3.6 Security, Privacy, and Ethical Concerns

N/A

A.4 Installation

The following instructions assume that your working directory is `$TOP` and that you are running Ubuntu 18.04 or 20.04. (Older versions of Ubuntu may require modifying these steps.)

A.4.1 Obtaining the source code

```
git clone --recurse-submodules \
https://github.com/derbear/aardvark-prototype.git
git checkout dd8f6aaf5f76173118f3f3decbe099bda5972ce2
git submodule update --init --recursive
```

A.4.2 Installing dependencies for EDRAX

```
sudo apt update
sudo apt install cmake g++ libgmp3-dev

# ignore errors while building dependencies here
cd $TOP/aardvark-prototype/edrax/ate-pairing ; make
cd $TOP/aardvark-prototype/edrax/xbyak ; make

cd $TOP/aardvark-prototype/edrax ; cmake . && make
```

A.4.3 Installing dependencies for vector commitments

```
# install rustup
curl --proto '=https' --tlsv1.2 \
-sSf https://sh.rustup.rs | sh
# input 1 for standard installation

# add to shell profile for this to be persistent
source $HOME/.cargo/env

rustup install nightly-2021-05-25
rustup default \
nightly-2021-05-25-x86_64-unknown-linux-gnu

cd $TOP/aardvark-prototype/veccom-rust ;
cargo build --release
```

A.4.4 Installing dependencies for Aardvark, integrated into Algorand

```
sudo apt update
sudo apt install autoconf automake libtool numactl

wget https://golang.org/dl/go1.16.4.linux-amd64.tar.gz
tar -C $TOP -xzf go1.16.4.linux-amd64.tar.gz

# add to shell profile for this to be persistent
export PATH=$PATH:$TOP/go/bin
export GOPATH=$TOP/go

cd $TOP/aardvark-prototype/veccom-rust ;
cargo build --release
cd $TOP/aardvark-prototype/go-algorand ; make
# input N when prompted, and ignore Makefile error
```

A.5 Experiment workflow

The following instructions assume that your working directory is `$TOP`.

A.5.1 EDRAX microbenchmark

The EDRAX binary resulting from compiling calls into the EDRAX implementation. It executes 100 iterations to warm up the machine state and then performs 1000 measurements of the implemented Verify, CommitUpdate, and ProofUpdate operations. The script `edrax/bench.sh` invokes the binary with the argument 10, which corresponds to vectors with size 1024, and writes the results as a CSV file to the file `bench.csv` to the current directory.

A.5.2 Aardvark vector commitment microbenchmark

The binary resulting from compiling `veccom-rust/src/bin/run_aardvark_bench.rs` calls into the implementation of vector commitments for Aardvark, as well as an implementation of a Merkle Tree. It executes 100 iterations to warm up the machine state and then performs the passed-in number of measurements of the operations described in §4.1. The script `veccom-rust/bench.sh` invokes the binary with the argument corresponding to vectors with size 1024 and with 1000 iterations, and it writes the results as a CSV file to the file `bench-results.txt` to the current directory.

A.5.3 Aardvark dictionary benchmark

Aardvark is implemented as a modification of the database of the Algorand cryptocurrency and is contained inside the repository under the subdirectory `go-algorand/ledger`. The benchmark itself is written as a Go test within the file `perf_test.go`, and it consists of a workload generation program (written as a Go test `TestWorkloadGen` for convenience), as well as timed benchmarks (written as a Go test `TestTimeWorkload`).

To generate the workload (which takes roughly 5 hours on the paper hardware), run the following:

```
cd $TOP/aardvark-prototype/go-algorand/ledger ;
./bench.sh
```

This will create in the `ledger` subdirectory the files `workload-{init-}{c,d,m}`, which correspond to the initialization data and sample load transactions for creation, deletion, and modification benchmarks, respectively. Once the workloads are created, the benchmarks may be run against them.

Note that if you are executing these commands over an SSH connection, a dropped connection will terminate the generation process, and you will need to reissue the command from the beginning. We suggest using commands such as `nohup`, `screen`, or `tmux` to prevent a dropped connection from interrupting the command.

A.6 Evaluation and expected results

The paper claims that Aardvark is a secure authenticated dictionary with substantial storage savings and short proofs, and it can process more than a thousand operations per second.

The security of Aardvark is justified through a paper proof. The evaluation contains an analysis of the storage savings and proof sizes, which are straightforward to compute. The rest of the evaluation performs an empirical analysis to obtain the throughput of a prototype implementation of Aardvark, which is shown in the artifact.

The paper obtains the following empirical results in the evaluation.

1. While Aardvark's vector commitments are more computationally intensive than Merkle trees, their costs are similar to those in EDRAX without use of a SNARK.
2. A 32-core Aardvark validator can process 1–3 thousand operations per second. Validator costs benefit from parallelization.
3. Costs for archives are reasonable: each core can process about 10 deletion operations per second or 20 modification/insertion operations per second.

The concrete numerical results are displayed on Tables 1 and 4 as well as Figures 3 and 4 in §8. Raw expected results for vector commitment microbenchmarks are in `edrax/results` and `veccom-rust/bench-results`, while raw expected results for validator and archive operations are in `go-algorand/ledger/validators.csv` and `go-algorand/ledger/archives.csv` respectively.

The following instructions assume that your working directory is `$TOP`.

A.6.1 Microbenchmarks

The paper claims in §8.1, Table 1 concrete latency numbers for key vector commitment operations for EDRAX, our implementation of Aardvark, and our implementation of a basic Merkle Tree. Reproduce them as follows.

```
# benchmark EDRAX latency
cd $TOP/aardvark-prototype/edrax ; ./bench.sh
# time ./bench.sh takes <1min on paper's hardware

# benchmark vector commitments latency
cd $TOP/aardvark-prototype/veccom-rust ; ./bench.sh
# time ./bench.sh takes <3mins on paper's hardware
```

The output results for EDRAX are in `edrax/bench.csv`, while the expected raw results in the paper are in `edrax/results`. The output results for the vector commitments are in `veccom-rust/bench-results.txt`, while the expected raw results in the paper are in `veccom-rust/bench-results`.

A.6.2 Validator and Archive throughput

The paper claims in concrete latency measurements for insertion, modification, and deletion operations for our implementation of Aardvark for validators (§8.3, Table 4 and Figure 3) and for archives (§8.4, Figure 4). Reproduce them as follows.

```
# first, generate the workload as described in the
# previous section

# runs 3 scaling tests on validators
cd $TOP/aardvark-prototype/go-algorand/ledger ;
./cores.sh
# time ./cores.sh takes <4hrs on paper's hardware

# runs 3 tests on archives
cd $TOP/aardvark-prototype/go-algorand/ledger ;
./acores.sh
# time ./acores.sh takes <4hrs on paper's hardware
```

The results for validators are in files named `outN.txt`, where N is the number of cores and is either 1, 2, 4, 8, 16, or 32, while the results for archives are in a file named `aout1.txt`. By default, both of these tests run 3 trials each. Expected raw values for these results for 10 total trials each, manually merged, are in `go-algorand/ledger/validators.csv` and `go-algorand/ledger/archives.csv` respectively.

Note that if you are executing these commands over an SSH connection, a dropped connection will terminate the experiment process, and you will need to reissue the command from the beginning. We suggest using commands such as `nohup`, `screen`, or `tmux` to prevent a dropped connection from interrupting the command.

A.7 Experiment customization

Different vector sizes may be passed to the vector commitments libraries by modifying the command-line arguments which the `bench.sh` files pass to the binaries.

Modifying `go-algorand/ledger/perf_test.go` will allow modifying the number of initial accounts, the number of load transactions, the number of blocks, and other parameters input to Aardvark. (Modifying any variables here will require regeneration of the workload.)