Reef: Fast Succinct Non-Interactive Zero-Knowledge Regex Proofs

Sebastian Angel, Eleftherios Ioannidis, and Elizabeth Margolin, *University of Pennsylvania;* Srinath Setty, *Microsoft Research;* Jess Woods, *University of Pennsylvania*

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

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

Open access to the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

USENIX Security '24 Artifact Appendix: Reef: Fast Succinct Non-Interactive Zero-Knowledge Regex Proofs

Sebastian Angel* Eleftherios Ioannidis* Elizabeth Margolin* Srinath Setty† Jess Woods*
*University of Pennsylvania †Microsoft Research

A Artifact Appendix

A.1 Abstract

We present the artifact of Reef (USENIX SECURITY 2024). It is a Docker container that makes installing Reef very straightforward. Once the container is built, the scripts to reproduce our experiments can be found in https://github.com/eniac/Reef/blob/main/tests/scripts. These run Reef end-to-end: we commit to a particular document, compile a particular regex to R1CS, and then prove and verify (non)matching of that regex with that document. Reef is written in Rust.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

There is no risk to a machine's security or data privacy during evaluation.

A.2.2 How to access

The Docker image for Reef's artifact can be found on GitHub: https://github.com/eniac/Reef/releases/tag/v1.0.0-baseline

A.2.3 Hardware dependencies

Evaluating "no recursion" baseline requires a sufficient amount of RAM memory (>80GB). Reef and the other baselines require <20GB. The baselines also require the ADX and BMI2 CPU features (due to circom).

A.2.4 Software dependencies

The Docker container uses Cargo, Rust's build system and package manager, to fetch all of Reef's dependencies. Docker community, latest version is required.

A.2.5 Benchmarks

We have four motivating applications and datasets.

(1): Password Strength Good passwords: randomly generated; Bad passwords: from NordPass list of the top 200 most common passwords [2], a list of weak passwords.
(2): Email Redactions Enron email dataset [3].
(3): ODoH Blocklisting Regexs: Pi-hole [1], which is a DNS sinkhole; Queries: randomly generated.
(3): Genetic Matching Base pairs and common mutations of the BRCAI and BRCAII genes, available from the US National Institutes of Health [4, 5].

All of the Regexs for our experiments can be found in Figure 36 of our (extended eprint) paper [6]. All of the documents can be found in our repo: https://github.com/eniac/Reef/tree/main/tests/docs.

A.3 Set-up

Make sure you have installed Docker CE: https://docs.docker.com/get-docker/.

A.3.1 Installation

Get this version of the github: https://github.com/eniac/Reef/releases/tag/v1.0.0-baseline and build the Docker image:

```
docker build -t reef .
docker run -m 16g -it reef
```

The memory requirements can be changed depending on which experiments you are planning on running.

A.3.2 Basic Test

A simple example is to try to prove and verify that the document regex `.*b` (any string that ends with a b) matches the document `aaaaaaaab`, using the `ascii` alphabet.

```
echo aaaaaaaab > document
reef -d document --commit ascii
reef -d document -r ".*b" --prove ascii
reef -r ".*b" --verify --cmt-name document.cmt ascii
```

After each `reef` step, you should see "reef", and also "Proving Finished"/"Verification Finished", respectively. There will also be some auxiliary information printed about the batch size, number of constraints, and which optimizations are on.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Proving is efficient in terms of time and memory for a variety of apps, docs, regexes. This is clear in the proving/solving/verification times/memory usage/proof and commitment size of (E4), but especially in relation to the baseline (E1), which represents the basic state of the art before Reef.

(C2): Reef's use of recursion and SAFAs is a better general-purpose solution than non-recursive proof systems and DFAs. It also makes possible applications that were not previously possible, like DNA (a large document) and passwords (a complicated Regex). Compare (E3) to (E1) and (E2).

(C3): Reef's nlookup optimizations (projections and hybrid tables) provide meaningful benefits. This is proven by the difference in outcomes of (E3) and (E4), especially in the case of the DNA application.

A.4.2 Experiments

We include two sets of experiments. The regular experiments (**E1, E2, E3, E4**) which can be run to get the results in the paper. However, many of the baselines take multiple days to run. As an alternative we also include a set of mini experiments (**E1-mini, E2-mini, E3-mini, E4-mini**). These mini experiments are limited to those applications that can finish in under 1 hour.

Note that most of the experiments require recompiling Reef with the correct flags. This is included in each experiment's script. We reference figures from our eprint [6] in the following text.

(E1): DFA Baseline [7.5 human-hours + 24 compute-hours + 76GB disk]
Execution: `./tests/scripts/naive.sh`
Results: Written to `./tests/results/memory`, `./tests/results/timings`.
Relevant memory files for DFA baselines in `./tests/results/memory` with `_naive` suffix. Timings written to corresponding application file in `./tests/results/timings`. Each individual experiment has a header: `{first 10 characters of the input}_{length of the input},{experiment type},{timestamp},{regex},{# automata transitions},{number of automata states}`.
DFA baseline experiments have `naive` as the experiment type in the header. We have an optional Jupyter Notebook script to help interpret the raw results files: `DataCleaning.ipynb`. Results should match Figure 7: Column 2, Figure 32, Figure 33: Column 5-6, Figure 34: Column 4-5.

(E1-mini): DFA mini Baseline [1 human-hour + 10 compute-hours + 6GB disk]
Execution: `./tests/scripts/naive-mini.sh`
Results: See **(E1)**. Mini experiments contain a single iteration of each experiments, as opposed to 10. Additionally, due to their long execution time `naive-mini` excludes the email applications and only runs the ODOH applications. Results should be consistent with the **ODOH** rows of: Figure 7: Column 2, Figure 32, Figure 33: Column 5-6, Figure 34: Column 4-5.

(E2): DFA + Recursion Baseline [46 human-hours + 52 compute-hours + 6GB disk]
Execution: `./tests/scripts/nwr.sh`
Results: See **(E1)**. Relevant memory files are suffixed `_nwr`. DFA + Recursion Baseline experiments have `nwr` as the experiment type in the header. Results should match Figure 7: Column 3, Figure 31, Figure 33: Column 5-6, Figure 34: Column 6-7.

(E2-mini): DFA+Recursion mini Baseline [25 human-minutes + 4 compute-hours + 1.5GB disk]
Execution: `./tests/scripts/nwr-mini.sh`
Results: See **(E1-mini)**. Due to their long execution time `nwr-mini` excludes the email applications and only runs the ODOH applications. Results should be consistent with the **ODOH** rows of: Figure 7: Column 3, Figure 31, Figure 33: Column 5-6, Figure 34: Column 6-7.

(E3): SAFA + nlookup Baseline [3.5 human-hours + 34 compute-hours + 18GB disk]
Execution: `./tests/scripts/safa_nlookup.sh`
Results: See **(E1)**. Relevant memory files are suffixed `_safa_nlookup`. SAFA + nlookup Baseline experiments have `safa+nlookup` as the experiment type in the header. Results should match Figure 7: Column 4, Figure 30, Figure 33: Column 3-4, Figure 34: Column 8-9.

(E3-mini): SAFA + nlookup mini Baseline [40 human-minutes + 4 compute-hours + 1.5GB disk]
Execution: `./tests/scripts/safa_nlookup-mini.sh`
Results: See **(E1-mini)**. Due to their long execution time `safa_nlookup-mini` excludes the DNA applications and only runs the ODOH, email, and password applications. Results should be consistent with the **ODOH, Redactions, and Password** rows of: Figure 7: Column 4, Figure 30, Figure 33: Column 3-4, Figure 34: Column 8-9.

(E4): Reef Evaluation [1 human-hour + 8 compute-hours + 9GB disk]
Execution: `./tests/scripts/reef.sh`
Results: See **(E1)**. Relevant memory files are suffixed `_reef`. Reef evaluation experiments have `reef` as the experiment type in the header. Results should match Figure 5, Figure 6, Figure 7: Column 5, Figure 29, Figure 33: Column 3-4, Figure 34: Column 10-11.

(E4-mini): Reef mini Baseline [1 human-hour + 8 compute-hours + 6GB disk]
Execution: `./tests/scripts/reef-mini.sh`
Results: See **(E1-mini)**. Results should be consistent with: Figure 5, Figure 6, Figure 7: Column 5, Figure 29, Figure 33: Column 3-4, Figure 34: Column 10-11.

A.5 Notes on Reusability

Reef can be build very easily without the Docker image and used for applications outside of the ones used in our experiments. Instructions are in the README of the repo: https://github.com/eniac/Reef. The only dependency is rust/cargo: https://www.rust-lang.org/tools/install.

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/.

References

[1] Regex filters for pi-hole. https://github.com/mmotti/pihole-regex/blob/master/regex.list.

[2] https://nordpass.com/most-common-passwords-list/, 2023.

[3] https://www.cs.cmu.edu/~enron/, 2023.

[4] https://www.ncbi.nlm.nih.gov/gene/672, 2023.

[5] https://www.ncbi.nlm.nih.gov/gene/675, 2023.

[6] S. Angel, E. Ioannidis, E. Margolin, S. Setty, and J. Woods. Reef: Fast succinct non-interactive zero-knowledge regex proofs. Cryptology ePrint Archive, Paper 2023/1886, 2023. https://eprint.iacr.org/2023/1886.