WaterBear: Practical Asynchronous BFT Matching Security Guarantees of Partially Synchronous BFT

Haibin Zhang, *Beijing Institute of Technology;* Sisi Duan, *Tsinghua University, Zhongguancun Laboratory;* Boxin Zhao, *Zhongguancun Laboratory;* Liehuang Zhu, *Beijing Institute of Technology*

https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-haibin

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

USENIX'23 Artifact Appendix: WaterBear: Asynchronous BFT with Information-Theoretic Security and Quantum Security

Haibin Zhang, Sisi Duan, Boxin Zhao, and Liehuang Zhu

A Artifact Appendix

A.1 Abstract

This document provides a tutorial on how to use the Waterbear codebase. In particular, five protocols are evaluated in the paper: BEAT-Cobalt; WaterBear-C; WaterBear-Q; WaterBear-QS-C; WaterBear-QS-Q. The results are evaluated on Amazon EC2 instances, reproducing the results of which requires an account on AWS. As reproducing all of our results in the paper is time-consuming (which takes us a few weeks), this document only focuses on how to use the codebase. If one is interested in reproducing the results in the paper, please refer to README under the waterbear/ec2 folder of our codebase for details.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

There is no security, privacy, or ethical concerns.

A.2.2 How to access

Our code can be obtained from `https://github.com/fififish/waterbear`, and the stable version can be found at `https://github.com/fififish/waterbear/releases/tag/usenixsec`.

To prevent testers from being able to download dependencies, we provide a complete code base including all dependencies, which can be obtained from `https://github.com/fififish/waterbear/tree/waterbear-with-dependencies`. All dependencies are included in "waterbear/src/".

A.2.3 Hardware dependencies

All experiments are deployed on EC2 of Amazon Web Services. We use both t2.medium and m5.xlarge instances for our evaluation. The t2.medium type has two virtual vCPUs (Intel Xeon expandable processor with maximum frequency of 3.3GHz) and 4GB memory and the m5.xlarge has four vCPUs (Intel Xeon Platinum processor with maximum frequency of 3.1GHz) and 16GB memory. Please refer to `https://aws.amazon.com/cn/ec2/instance-types/` for more information about the EC2 instance.

Most results we reported in the paper are conducted on m5.xlarge instances. We recommend m5.xlarge for reproducibility of our results.

A.2.4 Software dependencies

We ran our experiments using Ubuntu 20.04. More specifically, we choose "ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-20220610" on AWS.

To compile our code of protocols, we require go1.15.14 linux/amd64. We require the following libraries.

Additionally, several open source libraries are required. One can download the libraries using the following commands:

- go get -u google.golang.org/grpc
- go get -u golang.org/x/net
- go get -u golang.org/x/text
- go get -u golang.org/x/crypto/...
- go get -u golang.org/x/sys
- go get -u google.golang.org/genproto/
- go get -u github.com/klauspost/reedsolomon
- go get -u github.com/klauspost/cpuid
- go get -u github.com/cbergoon/merkletree
- go get -u github.com/golang/protobuf

Alternatively, one can also use the following command to get the dependencies:

```
make go

make install
```

The dependencies have tested as of Aug 2023. In case of failures of executing the commands above, one can alternatively download the dependencies and place them under waterbear/src/.

A.2.5 Benchmarks

We provide a script for reproducing our results. One can use python 3.x to run the script, we used python 3.8.10 in the experiment. The script can be found under the waterbear/ec2 folder. We provide a README on how to start the experiments. Alternatively, one can directly ssh to the servers to start the experiments manually.

For each experiment, we vary two major parameters: n and b, where n is the number of servers and b is the batch size. Given a fixed n, we vary b from 1 to a sufficiently large number (e.g., 30,000) to generate the results. For each experiment, we obtain the results from all servers, exclude ones with outstanding results (e.g., extremely large throughput), and obtain the average result of the servers. We repeat each batch size in experiment five times to obtain the results reported in the paper.

A.3 Set-up

A.3.1 Installation

1. Pull the code from https://github.com/fififish/waterbear or download the stable version from https://github.com/fififish/waterbear/releases/tag/usenixsec. Or pull the complete code base including all dependencies from https://github.com/fififish/waterbear/tree/waterbear-with-dependencies, which can be compiled directly without downloading any dependencies.

2. Set the environment by:

$$\textit{export}\ \text{GOPATH} = \text{\$PWD}$$

$$\textit{export}\ \text{GOBIN} = \text{\$PWD}/\textit{bin}$$

$$\textit{export}\ \text{GO111MODULE} = \textit{off}$$

3. Download the dependencies by running the following commands:

$$\textit{make go}$$

$$\textit{make install}$$

4. Compile the code by running the following commands:

$$\textit{make build}$$

   The compiled file will be created in "$PWD/bin". We recommend setting up $PWD as ./waterbear.

A.3.2 Basic Test

1. Modify the configuration file "$etc/conf.json$" to choose which protocol to execute. Details about the protocols are included in "$etc/node.txt$". The *id* of each server should be unique. By default, we use monotonically increasing ids, 0, 1, 2, $\cdots$. If one tests the code locally, modify IP addresses and port numbers of all servers manually. The IP addresses and port numbers of all servers can be modified by script when testing on AWS.

2. To run BEAT-Cobalt, generate keys for threshold PRF first by running the following command:

$$\textit{keygen}\ [n]\ [k]$$

   Here, n is the number of servers, and k is the threshold to generate the common coin. We set up $n = 3f + 1$ and k to $f + 1$ for most of our experiments.

   If the keys are successfully generated, they are located under waterbear/etc. Make sure that the generated keys are placed under the repository of *all* servers.

3. For all the servers, run the command below to start the servers:

$$\textit{server}\ [id]$$

   Here, $[id]$ is configured in conf.json and is different at each server.

4. Start a client to send transaction to start the protocol by running the following command:

$$\textit{client}\ [id]\ 1\ [b]\ [msg]$$

   Here, $[id]$ is the identifier of the client. We do not require the client to be registered. One can use any id that is unique, e.g., 1000. $[b]$ is the batch size. $[msg]$ can be any message. One can ignore the $[msg]$ field and a default message is included in the codebase.

5. All servers will print text like " *****epoch ends". This means the success of the epoch. One can repeat the operation of client after the epoch ends to start a new epoch.

A.4 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/.