Practical Asynchronous High-threshold Distributed Key Generation and Distributed Polynomial Sampling

Sourav Das, *University of Illinois at Urbana-Champaign;* Zhuolun Xiang, *Aptos;*
Lefteris Kokoris-Kogias, *IST Austria and Mysten Labs;*
Ling Ren, *University of Illinois at Urbana-Champaign*

https://www.usenix.org/conference/usenixsecurity23/presentation/das

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: Practical Asynchronous High-threshold Distributed Key Generation and Distributed Polynomial Sampling

Sourav Das[1] *Zhuolun Xiang*[2] *Lefteris Kokoris-Kogias*[3] *Ling Ren*[1]
[1]*University of Illinois at Urbana-Champaign,* [2]*Aptos,* [3]*IST Austria & Mysten Labs*
souravd2@illinois.edu, xiangzhuolun@gmail.com, lefteris@mystenlabs.com, renling@illinois.edu

A Artifact Appendix

A.1 Abstract

Our artifact is built using docker, and can be run on a single machine with multi-threaded or multi-processes emulation, and in a geo-distributed setting with multiple amazon-web service virtual machines.

There are four important parameter choices of our artifact: (i) `num`: number of nodes in the ADKG protocol, (ii) `ths`: maximum number of faulty nodes, (iii) `deg`: the degree of the ADKG polynomial, and (iv) `curve`: the choice of the elliptic curve, which is either `bls12381` or `ed25519`.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

N/A

A.2.2 How to access

Our entire artifact is packaged as a single library and can be downloaded from:

- https://github.com/sourav1547/htadkg

We recommend using the version specified by the commit:

- https://github.com/sourav1547/htadkg/commit/0d221e8965c5cf6b18823d894ef48c0fabc34b6e

A.2.3 Hardware dependencies

The basic tests require a stable internet connection. The main experiments require `num` amazon web services instances.

A.2.4 Software dependencies

- Docker https://www.docker.com/
- Docker compose https://docs.docker.com/compose/
- Python `3.7.x` or higher https://www.python.org/

A.2.5 Benchmarks

None.

A.3 Set-up

Our artifact requires docker and docker compose. To install docker, see https://docs.docker.com/get-docker/. To install docker-compose, see https://docs.docker.com/compose/install/. Check that docker and docker compose are installed correctly by running the `$docker` and `$docker-compose` commands in the terminal. Upon successful installation, both of these commands will display the available options.

Start the docker daemon. In case of docker desktop start the docker daemon by starting the docker desktop application or by running the `$open -a Docker` command in the terminal.

A.3.1 Installation

Our entire artifact is packaged as a single library and can be downloaded from:

- https://github.com/sourav1547/htadkg

Once the docker and docker compose are installed, and docker daemon is running, is installed build the code using the following instructions. Note that building the `adkg` docker image will take approximately 10 minutes, possibly longer depending upon the internet connection.

1. `cd` to `htadkg` folder
2. Build using `$docker-compose build adkg`.
3. Run a docker image of adkg `$docker-compose run --rm adkg bash`

Upon successful installation, the last command will open a terminal with `root@fb0991941061:/usr/src/adkg#`.

Remark. If the docker daemon is not running, expect the following error message while building the `adkg` docker image.

```
docker.errors.DockerException: Error while
   fetching server API version: 500 Server
   Error for http+docker://localhost/version:
   Internal Server Error ("b'dial unix
   docker.raw.sock: connect: connection
   refused'")
[71057] Failed to execute script
   docker-compose
```

A.3.2 Basic Test

There are two modes to perform basic tests, and both of these tests can be run locally. We elaborate on each of these tests below.

- Emulating `num` threads inside a docker image.
- Emulating `num` processes inside a docker image

Emulating multiple threads. After completing the steps mentioned in §A.3.1, run this test using the following command inside the `adkg` docker.

```
$pytest tests/test_adkg.py -o log_cli=true
   --num 4 --ths 1 --deg 2 --curve ed25519
```

The command runs an ADKG protocol with four nodes where at most one node is corrupt, and we want to secret share the ADKG secret key using a polynomial of degree two. Note that `num>3*ths` and `ths-1<deg<num-ths`. It is possible to change these values arbitrarily as long as they satisfy these constraints. We recommend running this basic test with less than 10 nodes for quicker results.

Emulating multiple processes. This approach creates multiple processes within a single docker image. Each process corresponds to one ADKG node and these processes communicate using an Inter-process communication (`ipc`) channel. To start the experiment, run the following command after following the instructions in §A.3.1.

```
$sh scripts/launch-tmuxlocal.sh
   scripts/adkg-tutorial.py [NUM_NODES]
```

For this basic test, our artifact supports 16, 32, and 64 nodes. To evaluate with arbitrary `num`, `ths` and `deg`, first, generate the corresponding configuration files using `gen_config.py`. We recommend testing with 16 and 32 nodes for quicker results.

Remark 1. Although this process runs `NUM_NODES` number of ADKG nodes, our artifact only displays the log of the first four nodes. All remaining logs are available at `dump.log`.

Remark 2. Since each node in this basic test communicates using `ipc`, the bandwidth usage of this basic test approximates the bandwidth usage we report in Table 3 of the paper.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1) *The ADKG protocol improves the average runtime and bandwidth usage per node compared to the state-of-the-art, as summarized in Table 3 of the paper.*

A.4.2 Experiments

(E1) [Setup AWS machines] [30 human-minutes]. Here are the steps to set up the AWS machines:

1. Create an AWS account and sign in to the AWS Management Console.
2. Open the EC2 console and then choose Launch Instance.
3. Choose an Amazon Machine Image (AMI), which is a pre-configured virtual machine image that contains the operating system, docker and docker compose.
4. Select the region where you want to launch your instances, and then choose an instance type, which specifies the hardware of the host computer.
5. Configure the instance details, such as the number of instances, network settings, and IAM roles.
6. Choose the storage and add any additional storage volumes, if required.
7. Configure the security group, which acts as a virtual firewall for your instance to control inbound and outbound traffic.
8. Review and launch the instances.
9. Create a key pair to securely access the instances remotely over SSH. Download and save the private key file (.pem extension) on your local machine.

We automate steps 4, 5, and 8 using the AWS CLI and the configuration file https://github.com/sourav1547/htadkg/blob/main/aws/aws-config.json. Note that the configuration file specifies the regions, security group IDs, image IDs, key file path, key name, instance type, and other parameters needed to launch and configure the instances. Make sure to fill in these fields with the appropriate values. We describe the configuration file in more detail in https://github.com/sourav1547/htadkg/tree/main/aws

(E2) [ADKG experiments][1 human hour + 3 compute hours]: Follow the instructions on https://github.com/sourav1547/htadkg/blob/main/aws/README.md.

1. `cd /path/to/htadkg/aws`
2. Update the config with appropriate parameters. Run `python3 -m aws.run-on-ec2` to start the AWS instances and run the adkg command specified in the config. This command creates a current.vms file which consists of instance ids of the VMs created during this run. Subsequent runs of this command will reuse the same VMs.
3. Upon completion, the raw data from each ADKG node will be available in the `/path/to/htadkg/data/` folder in your local machine.
4. After you are done testing you can delete the VMs using `python3 -m aws.delete_vms`.