Appendix

A. Artifact Appendix

In this appendix, we provide the information for obtaining the source code for Decentagram, building and reproducing the experimental results presented in Section VII, and the hardware and software requirements.

Description & Requirements. The evaluation of Decentagram comprises of 6 separate experiments:

- (E1) evaluates the gas cost to register, subscribe, and publish using Decentagram's Publisher, Subscriber, and Broker smart contracts.
- (E2) evaluates the gas cost of Decentagram's six different revoker contracts.
- (E3) evaluates the throughput of receipt processing implementation used in Decentagram's off-chain component, which is responsible for filtering blocks and notifying off-chain subscribers.
- (E4) evaluates the end-to-end latency of Decentagram's off-chain and on-chain notifications discussed in Section VII.D.
- (E5) demonstrates the revocation process of enclave components as described in Section VI.
- (E6) tests the compatibility of Decentagram with different blockchain networks. As mentioned in Section VII.A, we have deployed and tested Decentagram on Avalanche, Polygon, BNB, and Optimism.

As mentioned in Section VII, the first 2 experiments utilizes Ganache, a testing environment for the Ethereum blockchain, which can be installed locally. The next 3 experiments require a fully-synced Ethereum node and, in the case of block processing using enclave, Intel SGX hardware. We recognize that these requirements may be difficult to meet, so we provide an environment with these resources, and will describe the steps to access this environment. The last experiment requires a testnet wallet with enough funds to deploy contracts and send transactions in the target blockchain network. A testnet wallet with limited funds can be accessed in the provided environment.

How to Access. The source code for Decentagram including files to run the experiments to produce the results in this paper is available at https://doi.org/10.5281/zenodo.10224299.

Directory Structure. The repository has the following structure.

- **pubsub-onchain** contains the source code for the Publisher, Subscriber, and Broker contracts. Within this directory, are files for running experiment E1, and they are organized in the following subdirectories:
  - **PubSub**. The smart contract code acting as on-chain broker.
  - **tests**. Scripts to run the gas evaluations for the PubSubService contract. It should generate a figure and JSON file containing similar results described in Section VII.B. Besides, it also contains the minimal publisher and subscriber contracts described in Section VII.A.
  - **utils**. Code to send transactions in Ethereum, code to run network compatibility tests, and project configuration files.
- **revoker-onchain** contains the source code for the revoker contracts. Within this directory are files for running experiment E2, and they are organized in the following subdirectories:
  - **EnclaveRevoker**. Smart contracts for revoking enclave components.
  - **KeyRevoker**. Smart contracts for revoking general keys.
  - **libs**. Symbolic links pointing to *pubsub-onchain* and *ra-onchain*, since these contracts are used and referenced in the revoker contracts.
  - **tests**. Unit tests for smart contracts.
  - **utils**. Code to run the experiment and produces results shown in Table II.
- **Ethereum** contains the source code for the Ethereum client, which acts as an off-chain broker introduced in our paper. Within this directory are files for running experiment E3, and they are organized in the following subdirectories:
  - **include**. Header files for the Ethereum client.
  - **src**. Source code for the Ethereum client.
  - **tests**. Code to run the experiment and produce results shown in Figure 6.
    - **geth-go-throughput-eval**. Code to process Ethereum receipts using Go Ethereum library running under a *regular* environment.
    - **geth-enclave-throughput-eval**. Code to process Ethereum receipts using our Ethereum client implementation running under an *enclave* environment.
- **Revoker** contains the source code for the off-chain revocation list monitor. Within this directory are files for running the revocation demo, and they are organized in the following subdirectories:
  - **src**. Source code for the *Revoker* program.
  - **tests**.
    - **ProblematicApp**. The source code of an enclave program, which intentionally acts maliciously, by generating conflicting messages, and exposing private keys that should never be revealed under regular circumstances. This app will be providing credentials to be revoked during the revocation demo.
    - **End2EndLatency**. Code to run the end-to-end latency evaluation and produce results shown in Table III.
- **libs** contains all the libraries that are needed when compile the programs and contracts. It also contains helper programs to deploy contracts on the EVM-compatible

networks. Here we will be highlighting some of the important libraries:

- **ra-onchain**. This library contains the smart contracts for on-chain parsing and verification of the Decent Self-Attestation certificates generated by enclaves.
- **PyEthHelper**. This is the helper program to interact with the Ethereum network via JSON-RPC over HTTP. It is used to deploy contracts, send transactions, and call contract functions.

Hardware dependencies. To reproduce the results for experiments E1 and E2, no special hardware is needed. To reproduce the results for experiment E3, the machine must have SGX enabled, including the corresponding SDKs.

Software dependencies. The following software is required to reproduce the results for our experiments.

- Linux OS (tested on Ubuntu 22.04 LTS on x86_64)
- Node.js version 18.xx
- NPM version 9.x
- Python >= 3.10
- Python3-pip
- Python3-venv
- Ganache. Can be installed with npm.

```
$ npm install -g ganache@7.8.0
```

Using Provided Environment. Experiments E1 and E2 can be run on any machine provided it meets the hardware and software requirements described above. For experiment E3 as well as the revocation demo and the end-to-end latency evaluation, we provide a machine with access to a full Ethereum node and SGX hardware that can be used to run these experiments.

To access our machine, use the following command

```
$ ssh -p 56615 aereviewer@decent-aurora.soe.ucsc.edu
```

Major Claims.

- **(C1) Efficiency:** The cost to add a new publisher or subscriber to the on-chain broker remain constant as the number of publishers and subscribers increase, respectively. Additionally, the cost to notify subscribers increases linearly with the number of subscribers.
- **(C2) Performance:** Decentagram's receipt processing code is able to process receipts from new blocks at a rate much faster than that produced by Ethereum's network, allowing it to make timely notifications to subscribers.

Experiments

E1 & E2: on-chain gas eval [10 minutes]: The results of experiment E1 are discussed in Section VII.A, and the results of experiment E2 are shown in Table II.

[Setup] Under the Decentagram-main directory, run the following command:

```
$ ./A1_contracts_setup.sh
```

This setup script will first compile all the smart contracts, and then it will set up a Python virtual environment that will be used to run the tests.

[Execution] Under the Decentagram-main directory, run the following command:

```
$ ./A2_contracts_tests.sh
```

[Results - E1] After the tests are finished, the file Decentagram-main/pubsub-onchain/build/gas_cost.pdf will contain the results that are discussed in Section VII.A.

We have configured a public Github repo with Github Action that automatically runs the gas evaluation on each release, which can be found at: https://github.com/lsd-ucsc/decent-pubsub-onchain/releases. You may audit the full output log generated from the last GitHub Action run at: https://github.com/lsd-ucsc/decent-pubsub-onchain/actions/

[Results - E2] Under the Decentagram-main/revoker-onchain/build directory, the following results are generated:

- build/gas_cost_decent_revoker.json Gas cost of three types of Decent Revoker contracts
- build/gas_cost_key_revoker.json Gas cost of three types of general key revoker contracts

The results from these files are used to generate Table II.

Similar to the gas evaluation for E1, we have configured a public Github repo with Github Action that automatically runs the gas evaluation on each release, which can be found at: https://github.com/lsd-ucsc/decent-revoker-onchain/releases. You may audit the full output log generated from the last GitHub Action run at: https://github.com/lsd-ucsc/decent-revoker-onchain/actions

E3: off-chain throughput [30 minutes]: This experiment produces the results shown in Figure 6.

[Setup] Under the Decentagram-main directory, run the following command:

```
$ ./B1_offchain_setup.sh
```

This setup script will compile the Ethereum client and the Revoker.

[Execution - Non-enclave] Under the /Decentagram-main/Ethereum/tests/geth-go-throughput-eval/ directory, run the following command:

```
$ go run .
```

[Results - Non-enclave] The output will show the throughput of receipt processing rate for the non-enclave version of receipt processing implementation.

[Execution - Enclave] Under the /Decentagram-main/Ethereum/build/tests/geth-enclave-throughput-eval directory, run the following command:

```
$ ./GethThroughputEval
```

[Results - Enclave] The output will show the throughput of receipt processing rate for the enclave version of receipt processing implementation.

E4: End-to-end Latency [off-chain and on-chain] [5 minutes]: This experiment evaluates the end-to-end latency of the Decentagram, and produces results shown in Table III.

[setup] This experiment uses the smart contracts and Ethereum client that were built in the previous experiments. Under the Decentagram-main directory, run the following command:

```
$ ./A1_contracts_setup.sh
$ ./B1_offchain_setup.sh
$ ./B2_contracts_deploy.py \
  --geth-addr http://172.17.0.1:8548 \
  --key-file /etc/testnet_keys.json
$ ./B3_project_config.py --host-ip 172.17.0.1
```

These scripts will compile the contracts, deploy them to the Ethereum testnet.

NOTE: The contracts are deployed on the Ethereum Holesky testnet, and the Geth client can take some time to find peers in this network, so you may run into a timeout error. When this happens, please wait a few minutes and try again. Additionally, when retrying the deployment, the old transaction may still be stuck in the cache and you can get a replacement transaction underpriced error. In this case, also please wait a few minutes and try again.

NOTE: The Ethereum testnets are subject to network congestion which will increase the price to send transactions. If the congestion is high enough, this can cause transactions from our Geth client to fail. To ensure the transaction succeeds in a congested network, please increase the deposit amount in our Geth proxy located at line `204` of Decentagram-main/Revoker/tests/End2EndLatency/contracts/GethProxy.py from `value=w3.to_wei(0.0001, "ether")` to `value=w3.to_wei(0.1, "ether")`.

[Execution]

Step 1. Under the Decentagram-main/Revoker/tests/End2EndLatency/contracts directory, run the following command:

```
$ python3 GethProxy.py \
  --geth-addr http://172.17.0.1:8548 \
  --key-file /etc/testnet_keys.json
```

Wait for the message that says ”Geth Proxy starts to listen to incoming requests...”, and keep the terminal open with this process running.

Step 2. In a second terminal, under the Decentagram-main/Ethereum/build/src directory, run the following command:

```
$ ./EthereumClient
```

Keep the terminal open with this process running.

Step 3. In a third terminal, under the Decentagram-main/Revoker/build/tests/End2EndLatency directory, run the following command:

```
$ ./End2EndLatency
```

Once the tests are finished, the results will be in the Decentagram-main/Revoker/build/tests/End2EndLatency directory.

The process of this test is described in the following file: Decentagram-main/Revoker/tests/End2EndLatency/doc.

E5: Revocation Demo [off-chain and on-chain] [5 minutes]: This demo shows the end-to-end process of revoking enclave components, which includes deploying the revocation contracts, having the enclave generate conflicting messages, and then reporting these messages to the contract to have the enclave revoked. The setup is similar to the end-to-end latency evaluation, with the option to configure the revoker to run the leaked key demo.

[Setup] Under the Decentagram-main directory, run the following command:

```
$ ./A1_contracts_setup.sh
$ ./B1_offchain_setup.sh
$ ./B2_contracts_deploy.py \
  --geth-addr http://172.17.0.1:8548 \
  --key-file /etc/testnet_keys.json
$ ./B3_project_config.py --host-ip 172.17.0.1

# the above will run the conflicting message demo.
    To run the leaked key demo
$ ./B3_project_config.py \
  --host-ip 172.17.0.1 \
  --revoker EnclaveRevokerByLeakedKey
```

[Execution]

Step 1. Go to the Decentagram-main/Ethereum/build/src directory and run the following command:

```
$ ./EthereumClient
```

Keep the terminal open with this process running.

Step 2. In a second terminal, under the Decentagram-main/Revoker/build/src directory, run the following command:

```
$ ./Revoker
```

Keep the terminal open with this process running.

Step 3. In a third terminal, run the following commands:

```
$ cd Decentagram-main/Revoker/build/tests/
    ProblematicApp
$ ./ProblematicApp
$ cd ../../../..
$ ./B4_report_credential.py \
  --geth-addr http://172.17.0.1:8548 \
  --key-file /etc/testnet_keys.json \
  --server-cert-path /etc/decent_svr_cert.hex

# to run the leaked key demo
$ ./B4_report_credential.py \
  --geth-addr http://172.17.0.1:8548 \
  --key-file /etc/testnet_keys.json \
  --server-cert-path /etc/decent_svr_cert.hex \
  --revoker EnclaveRevokerByLeakedKey
```

Observe the revocation message to be shown on the terminal running Revoker (Step 2).

E6: Network Compatibility Test [5 minutes]: This experiment will determine whether Decentagram works with your blockchain network of choice.

[Setup] Under the Decentagram-main directory, run the following command:

```
# optional if already ran in E1
$ ./A1_contracts_setup.sh
```

[Execution] Under the Decentagram-main directory, run the following command:

```
$ ./venv/bin/python3 pubsub-onchain/utils/
    NetworkCompatibility.py \
  --api-url https://chain-proxy.wallet.coinbase.com?
      targetName=ethereum-holesky \
  --key-file /etc/testnet_keys.json

$ ./venv/bin/python3 libs/ra-onchain/utils/
    NetworkCompatibility.py \
  --api-url https://chain-proxy.wallet.coinbase.com?
      targetName=ethereum-holesky \
  --key-file /etc/testnet_keys.json

$ ./venv/bin/python3 revoker-onchain/utils/
    NetworkCompatibility.py \
  --api-url https://chain-proxy.wallet.coinbase.com?
      targetName=ethereum-holesky \
  --key-file /etc/testnet_keys.json
```

For each of the commands above, a log with details about the network compatibility and a JSON file containing the contract deployment info will be saved in the current directory.

Network Name	targetName
Ethereum Holesky	ethereum-holesky
Avalanche Fuji	avalanche-fuji
Polygon Mumbai	polygon-mumbai
BNB	bsc-testnet
Optimism Sepolia	optimism-sepolia

TABLE I
LIST OF NETWORKS THAT PASSES THE COMPATIBILITY TEST.

Table I shows the list of networks that we have tested on. You may replace the `targetName` in the URL to test on other networks.