USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Aion: Robust and Efficient Multi-Round Single-Mask Secure Aggregation Against Malicious Participants

Yizhong Liu, Zixiao Jia, Zian Jin, Xiao Chen, Song Bian,
Runhua Xu, Dawei Li, and Jianwei Liu, *Beihang University;*
Yuan Lu, *Institute of Software, Chinese Academy of Sciences*

https://www.usenix.org/conference/usenixsecurity25/presentation/liu-yizhong

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: Aion: Robust and Efficient Multi-Round Single-Mask Secure Aggregation Against Malicious Participants

Yizhong Liu[†], Zixiao Jia[†], Zian Jin[†], Xiao Chen[†], Song Bian[†], Runhua Xu[ᛤ], Dawei Li[†*], Jianwei Liu[†*], Yuan Lu[ᛋ]

[†]School of Cyber Science and Technology, Beihang University, [ᛤ]School of Computer Science and Technology, Beihang University, [ᛋ]Institute of Software, Chinese Academy of Sciences

Email: {liuyizhong, jiazixiao, jinzian, chenxiao, sbian, runhua, lidawei, liujianwei}@buaa.edu.cn, luyuan@iscas.ac.cn

A Artifact Appendix

A.1 Abstract

This paper introduces Aion, a multi-round single-mask secure aggregation scheme with evolving input validation against malicious clients and aggregators. The artifact includes implementations of the core components evaluated in our paper, covering the following experiments: Secure Aggregation Protocol Simulation **(E1)**, Input Validation Evaluation **(E2)**, and Gradient Inversion Attack **(E3)**. In **E1**, we evaluate Aion's performance and robustness in terms of execution time, message overhead, and network resilience under varying system parameters. **E2** evaluates Aion's L_2-norm-based input validation against poisoning attacks on multiple datasets. **E3** reproduces a gradient inversion attack with access to labels and BatchNorm statistics, demonstrating the effectiveness of Aion's mask-based privacy protection.

A.2 Description & Requirements

This section details the experimental setup, specifying the hardware and software environments, as well as the benchmarks employed to generate the reported results.

A.2.1 Security, privacy, and ethical concerns

There are no security, privacy, or ethical concerns associated with the execution of this artifact.

A.2.2 How to access

Our artifact is available at https://doi.org/10.5281/zenodo.15605465.

A.2.3 Hardware dependencies

Evaluating our artifact requires a system equipped with at least an NVIDIA GeForce RTX 3060 GPU and with no specific CPU requirements. However, for efficiency, we recommend using a higher configuration, e.g., NVIDIA GeForce RTX 4090 GPU and Intel Core i9-14900KF.

A.2.4 Software dependencies

E1 is compatible with both Windows and Linux, except for the ACORN scheme, which is only supported on Linux. **E2** is compatible with Windows, while **E3** is for Linux. We strongly recommend deploying **E2** and **E3** using Docker, which is compatible with both Windows and Linux. All required dependencies are listed in the `requirements.txt`.

A.2.5 Benchmarks

Our experiments require the CIFAR10, FMNIST, EMNIST-Byclass, and SHAKESPEARE datasets, which are automatically downloaded from their official sources during code execution. The ResNet-18, LeNet-5, and ResNet-9 models are used in **E2** and **E3**.

A.3 Set-up

This section provides the steps necessary to install and configure the environment for evaluating the artifact.

A.3.1 Installation

We use Anaconda to set up the environment of **E1**.

1. Create a new Conda environment named aion and activate it: `conda create --name aion python=3.8`
`conda activate aion`
2. Use pip to install the required packages: `pip install -r requirements.txt`

We recommend using Docker for installation of **E2** and **E3**. The pre-built images can be downloaded as follows:

E2: `docker pull aionaion/input_validation:latest`
E3: `docker pull aionaion/gradattack:latest`

A.3.2 Basic Test

For **E1**, enter the folder `pki_files` and run `setup_pki.py`:
`cd pki_files`

```
python setup_pki.py
cd ..
```

E1 has multiple configs.

```
-c [protocol name]
-n [number of clients (power of 2)]
-i [number of iterations (training rounds)]
-A [number of aggregators]
```

Aion supports batches of clients with size power of 2, e.g., 128, 256, 512, 1024, 2048, 4096.

Example command:

```
python abides.py -c aion -n 128 -A 8 -i 10
```

For **E2** and **E3**, once the image is pulled, activate the main container to run the functionality test:

E2: `docker run --gpus all --rm aionaion/input_validation:latest`

E3: `docker run --gpus all --rm aionaion/gradattack:latest`

You can also verify that all required dependencies are correctly installed without using Docker:

E2: `python ./FL_Backdoor_CV/roles/autorun.py`

E3: `python examples/attack_cifar10_gradinversion.py #remove spaces in file path`

For **E2**, if the training progresses to Round 301, you can safely terminate the process. For **E3**, we recommend waiting until the attack completes to inspect the reconstructed image.

A.4 Evaluation workflow

In this section, we present the main claims of the paper along with the experiments designed to support them.

A.4.1 Major Claims

The major claims presented in our paper are as follows:

(C1): Aion consistently demonstrates superior performance in terms of execution time and communication overhead compared to state-of-the-art schemes across varying numbers of clients (q) and aggregators (n). This is proven by the experiment (**E1.1**) described in Section 7.1, whose results are illustrated in Figure 2 (for varying q) and Figure 3 (for varying n).

(C2): Aion maintains robust and efficient performance even as the security parameter for secret sharing (p) increases, showing significantly less performance degradation compared to competing schemes. This is proven by the experiment (**E1.2**) described in the supplementary material (Appendix C), whose results are illustrated in Figure 13.

(C3): Aion exhibits strong resilience and superior time efficiency under adverse network conditions, consistently outperforming other masking-based schemes. This is proven by the experiment (**E1.3**) described in the supplementary material (Appendix C), whose results are illustrated in Figure 14.

(C4): Aion effectively defends against poisoning attacks when the poisoning ratio is below 50%. This is proved by experiment (**E2**), described in Section 7.2, with the corresponding results presented in Figures 4 and 5.

(C5): Aion is shown to be robust against gradient inversion attacks even when attackers have access to private image labels and BatchNorm statistics, as evidenced by experiment (**E3**) in Appendix F.2 and the corresponding results in Figure 16.

A.4.2 Experiments

The following experiments provide supporting evidence for the main claims of the paper.

(E1.1): This experiment aims to verify that Aion has superior performance in terms of execution time and message overhead compared to state-of-the-art schemes (Flamingo, ACORN, SecAgg+) when varying the number of clients (q) and aggregators (n).

Execution: To reproduce the results for varying client numbers (Figure 2), run the following commands, iterating through the number of clients `-n` with values of 128, 256, 512, 1024. The number of aggregators is fixed at 8.

```
python abides.py -c aion -n 128 -A 8 -i 10
python abides.py -c aion -n 256 -A 8 -i 10
python abides.py -c aion -n 512 -A 8 -i 10
python abides.py -c aion -n 1024 -A 8 -i 10
```

To reproduce the results for varying aggregator numbers (Figure 3), run the following commands, iterating through the number of aggregators `-A` with values of 8, 16, 32, 64. The number of clients `-n` is fixed at 512.

```
python abides.py -c aion -n 512 -A 8 -i 10
python abides.py -c aion -n 512 -A 16 -i 10
python abides.py -c aion -n 512 -A 32 -i 10
python abides.py -c aion -n 512 -A 64 -i 10
```

Results: The script will output the execution time (in seconds) and message overhead (in bytes) for the initialization and aggregation phases. You can plot these values to reproduce Figures 2 and 3. The performance is expected to surpass the results of the baseline schemes reported in the figures.

(E1.2): This experiment verifies that even as the security parameter p (the prime size for secret sharing) increases, Aion's performance remains robust and efficient, degrading much more gracefully than competing schemes.

Execution: Vary the `--prime_bits` with values of 2048, 3072, 4096, with the number of clients `-n` set to 256 and the number of aggregators `-A` set to 8.

```
python abides.py -c aion -n 256 -A 8 -i 10
--prime_bits 2048
python abides.py -c aion -n 256 -A 8 -i 10
--prime_bits 3072
python abides.py -c aion -n 256 -A 8 -i 10
--prime_bits 4096
```

Results: The script will report the initialization and aggregation times for each configuration. After running all commands, plot the execution times against the prime

sizes to reproduce Figure 13. The performance is expected to surpass the results of the baseline schemes reported in the figure.

(E1.3): This experiment shows Aion's strong resilience and superior time efficiency under adverse network conditions, including high latency and low bandwidth.

Execution: Varying latency (corresponding to Figures 14a, 14b):

For each scheme, run the script using different `--latency` values (e.g., 10, 100 ms):

```
python abides.py -c aion -n 512 -A 8 -i 10
--latency 10
python abides.py -c aion -n 512 -A 8 -i 10
--latency 100
```

Varying bandwidth (corresponding to Figures 14c, 14d):

For each scheme, run the script using different `--bandwidth` values (e.g., 1, 5, 10 Mbps):

```
python abides.py -c aion -n 512 -A 8 -i 10
--bandwidth 1
python abides.py -c aion -n 512 -A 8 -i 10
--bandwidth 5
python abides.py -c aion -n 512 -A 8 -i 10
--bandwidth 10
```

Results: The script will output the total time overhead (computation + communication) for the initialization and aggregation phases under the simulated network conditions. By plotting these results, you can reproduce Figure 14. Aion is expected to have the lowest time overhead across all tested conditions, with its advantage being most pronounced in challenging environments such as high latency and low bandwidth.

(E2): This experiment evaluates whether Aion can effectively defend against poisoning attacks.

How to: To run the experiments, Docker users should enter the container with: `docker run --gpus all -it aionaion/input_validation:latest /bin/bash`. Then execute the scripts in the `roles` folder (e.g., `attack1_cifar10.py`). You can adjust the poisoning ratio using `number_of_adversaries`, the mask ratio using `weight`, and the boost rate of malicious gradients using `mal_boost`.

Execution: Please run the command:

```
python ./FL_Backdoor_CV/roles/
attack1_cifar10.py  --aggregation_rule aion
--number_of_adversaries 5
#remove spaces in file path
```

This process is expected to take approximately 20 minutes. Upon completion, the ASR and TER values will be recorded in `.pt` files within the `results` folder. To reproduce the results shown in Figure 4, you can vary the `number_of_adversaries` parameter from 5 to 50.

```
python ./FL_Backdoor_CV/roles/
attack2_cifar10.py
--aggregation_rule aion --weight 0.1
python ./FL_Backdoor_CV/roles/
attack3_cifar10.py
--aggregation_rule aion --mal_boost 50
#remove spaces in file path
```

You can vary the `weight` parameter from 0.05 to 0.5 and the `mal_boost` parameter from 25 to 250. The results contribute to Aion curves presented in Figures 6 and 8.

Results: You can check the results by running: `python check_results.py`.

(E3): : This experiment aims to demonstrate that Aion is robust against gradient inversion attacks.

How to: To run the experiments, please enter the container's interactive environment. The following commands also mount a local `results` directory into the container to save attack outputs.

```
docker run --rm -it --gpus all -v "%cd%
results:/app/results" aionaion/gradattack
/bin/bash (For Windows)
sudo docker run --rm -it --gpus all
-v "$PWD/results":/app/results
aionaion/gradattack /bin/bash (For Linux)
```

To run the experiments, execute `attack_cifar10_gradinversion.py` in the `example` folder. Each run will generate a reconstructed image.

Preparation: You can optionally train the checkpoint using Aion as the target model.

```
python3 examples/train_cifar10.py
--n_epoch 48 --logname CIFAR10/Aion
--defense_aion --weight 0.1
```

However, this step is not required, as we have provided pre-trained checkpoints in the `checkpoint_old` folder.

Execution: Please run the command:

```
python3 examples/attack_cifar10_
gradinversion.py --batch_size 16 --BN_exact
--tv 0.1 --bn_reg 0.005 --defense_aion
--weight 0.1 --attack_checkpoint
checkpoint_old/aion_epoch=48.ckpt
#remove spaces in file path
```

To perform an attack on a newly trained model, be sure to update the `attack_checkpoint` parameter accordingly. Note that newly generated checkpoints are saved in the `checkpoint` folder.

Results: This process is expected to take approximately 10 minutes. Upon completion, the reconstructed image, `reconstructed.png`, will be saved in the `results` folder on your local machine (the one you mapped to the Docker container).

A.5 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/usenixsec2025/.