USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Dumbo-MPC: Efficient Fully Asynchronous MPC with Optimal Resilience

Yuan Su, *Xi'an Jiaotong University;* Yuan Lu, *Institute of Software Chinese Academy of Sciences;* Jiliang Li, *Xi'an Jiaotong University;* Yuyi Wang, *CRRC Zhuzhou Institute;* Chengyi Dong, *Xi'an Jiaotong University;* Qiang Tang, *The University of Sydney*

https://www.usenix.org/conference/usenixsecurity25/presentation/su-yuan

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: Dumbo-MPC: Efficient Fully Asynchronous MPC with Optimal Resilience

Yuan Su
Xi'an Jiatong University

Yuan Lu
Institute of Software
Chinese Academy of Sciences

Jiliang Li
Xi'an Jiatong University

Yuyi Wang
CRRC Zhuzhou Institute

Chengyi Dong
Xi'an Jiatong University

Qiang Tang
The University of Sydeny

A Artifact Appendix

This repository contains the artifact for the USENIX 2025 submission "Dumbo-MPC: Efficient Fully Asynchronous MPC with Optimal Resilience".

A.1 Abstract

Fully asynchronous multi-party computation (MPC) has superior robustness in realizing privacy and guaranteed output delivery (G.O.D.) against asynchronous adversaries that can arbitrarily delay communications. We design a concretely efficient fully asynchronous MPC—Dumbo-MPC with entire G.O.D. and optimal resilience against $t < n/3$ corruptions (where n is the total number of parties). The codebase includes the implementation for Dumbo-MPC.

Our artifact is built with Ubuntu 20.04 LTS, and can be run on a single machine with multi-threaded or multi-processes emulation, or in a distributed setting consisting of multiple amazon-web service (AWS) virtual machines. There are two important parameter choices of our artifact: (i) num_node: number of nodes in the MPC protocol, (ii) batchsize: the number of secrets to be shared.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

There are no concerns when running this artifact locally. Please note that executing experiments on your AWS infrastructure involves the creation of multiple EC2 instances, resulting in associated costs. Please manually check that any created machines are terminated afterward.

A.2.2 How to access

All of our files are publicly accessible and can be downloaded from: `https://github.com/dcy456/Dumbo-MPC`, which is also at the permanent Zenodo repository `https://zenodo.org/records/15123146`.

A.2.3 Hardware dependencies

Our code can be executed at the local machine installing Ubuntu 20.04 LTS, or among a few distributed AWS instances. For distributed evaluations, EC2 instances of type c6a.8xlarge with 32 vCPUs and 64 GB memory are recommended for reproducing the evaluation results reported in the paper.

A.2.4 Software dependencies

The following dependencies are required:

- Operating system: Ubuntu 20.04 LTS.
- System dependencies: `make bison flex libgmp-dev libmpc-dev libntl-dev libflint-dev python3 python3-dev python3-pip libssl-dev wget git build-essential curl tmux`. We require specific versions of programming languages: Python 3.8.x, Rust 1.86.0-nightly and Go 1.18.
- Python dependencies: `cffi Cython gmpy2 pyzmq pycryptodome pyyaml psutil reedsolo numpy pytest zfec`.
- External libraries: PBC (Pairing-Based Cryptography) library and Charm library.

A.3 Set-up

A.3.1 Installation

Setup at local machine: To locally run Dumbo-MPC, a user needs to install all dependencies listed in README.md file. Please follow the README.md file in the repository to install the necessary dependencies step by step.

Setup AWS machines: Here are the steps to set up the AWS machines for distributed tests:[1]

[1]The AWS setup is for evaluating reproducibility and can be skipped for availability and functionality evaluations.

1. Create your AWS credentials: Setup your AWS credentials to enable programmatic access to your account from your local machine. These credentials will authorize your machine to create, delete, and edit instances on your AWS account programmatically. First of all, find your access key id and secret access key. Then, create a file " /.aws/credentials" with the following content:

   ```
   [default]
   aws_access_key_id = YOUR_ACCESS_KEY_ID
   aws_secret_access_key = YOUR_SECRET_ACCESS_KEY
   ```

2. Add your SSH publlic key to your AWS account: You must now add your SSH public key to your AWS account. This operation is manual (AWS exposes little APIs to manipulate keys) and needs to be repeated for each AWS region that you plan to use. Upon importing your key, AWS requires you to choose a "name" for your key; ensure you set the same name on all AWS regions. This SSH key will be used by the python scripts to execute commands and upload/download files to your AWS instances. If you don't have an SSH key, you can create one using ssh-keygen: `ssh-keygen -f /.ssh/aws`.

3. Deploying Dumbo-MPC on AWS: Launch an instance on AWS (with Ubuntu 20.04 LTS). If you are not familiar with AWS, you can visit Get started with Amazon EC2 Linux instances to get some help. To connect to the launched instance, use SSH:

   ```
   ssh -i your_ssh_key_path
   ubuntu@public_ip_of_instance
   ```

   Then, upload codes of Dumbo-MPC into instance:

   ```
   scp -i your_ssh_key_path -r Dumbo-MPC_code_path
   ubuntu@public_ip_of_instance:~/
   ```

   or clone Dumbo-MPC project:

   ```
   git clone https://github.com/dcy456/Dumbo-MPC.git
   ```

   Finally, you should install dependencies according to README.md, and create an image (e.g., store the image with name `your-image-id`) for this instance. If you are not familiar with AWS, you can visit Create an AMI from an Amazon EC2 Instance to get some help.

A.4 Evaluation Workflow

A.4.1 Major claims (for functionalities)

(C1): The AsyRanTriGen protocol (i.e. the pessimistic offline phase of Dumbo-MPC) is implemented, and can produce Shamir-style multiplication triples over the field of curve BLS12-381.

(C2): The entire offline phase of Dumbo-MPC is implemented, beginning with the execution of a fast path (i.e. the OptRanTriGen protocol). If the fast path fails, a fallback algorithm is triggered. Finally, AsyRanTriGen protocol (pessimistic path) will be executed to restore the robust generation of multiplication triples. All multiplication triples (generated by either OptRanTriGen or AsyRanTriGen) are of Shamir-style and over the field of curve BLS12-381.

(C3): The state-of-the-art asynchronous triple generation protocol GS23 is also implemented, and its implementation can generate multiplication triples over the field of curve secp256k1.

(C4): The multiplication triples generated by the offline phase of Dumbo-MPC (either the fast path or the pessimistic path) can be used by the online evaluations of hbMPC, such as shuffling via a butterfly network.

A.4.2 Experiments (for evaluating functionalities)

The approach creates multiple processes within a single (local) machine. Each process corresponds to one node in our protocols, and these processes communicate using an interprocess communication (ipc) channel.

(E1): A quick start to locally run AsyRanTriGen protocol to generate multiplication triples can be:

1. `cd /path/to/Dumbo-MPC`.
2. `./run_local_network_test.sh asy-triple <num_node> <batchsize>`. For this test, our artifact supports several fixed numbers of nodes, including the choices of 4, 10, 22 and 31, and arbitrary batch size. For example, running `./run_local_network_test.sh asy_triple 4 200` will generate 200 multiplication triples. The experiment logs are shown at `./dumbo-mpc/AsyRanTriGen/log/` with recording the execution latency, and the generated triples are at `./dumbo-mpc/AsyRanTriGen/triples/`. [2]

(E2): Recall that in the offline phase of Dumbo-MPC, an OptRanTriGen protocol (fast path) is firstly executed to optimistically generate multiplication triples, and if a fast-path failure occurs, a fallback happens to switch into the AsyRanTriGen protocol. So we provide the following quick start to locally run the offline protocols of Dumbo-MPC, by simulating a fast-path failure at 10-th round to trigger fallback.

1. `cd /path/to/Dumbo-MPC`.
2. `./run_local_network_test.sh dumbo-mpc <num_node> <batchsize>`. For this test, our artifact supports 4, 10, 22 and 31 nodes, and arbitrary batchsize. For example, running `./run_local_network_test.sh dumbo_mpc 4`

[2] In all subsequent experiments, the latency associated with saving the generated multiplication triples to a file is not included.

200 will generate $200*9+200=2000$ triples in total, where all nodes agree on the 9-th round of fast path and each round outputs 200 triples, and the pessimistic path outputs another 200 triples. The experiment logs are shown at ./dumbo-mpc/dualmode/log/, providing the latency for each protocol. The generated triples from both paths are stored as files under the directory ./dumbo-mpc/dualmode/opt-triples/ and ./dumbo-mpc/dualmode/asy-triples/.

(E3): A quick start to locally run GS23 protocol can be:

1. cd /path/to/Dumbo-MPC/GS23.
2. ./scripts/local_test.sh scripts/run-beaver.py <num_node> <batchsize>. For this test, our artifact supports 4, 10, 22 and 31 nodes, and batchsize $\leq$ 5000. For example, running ./scripts/local_test.sh scripts/run_beaver.py 4 200 will generate 100 multiplication triples. The experiment logs are stored at Dumbo-MPC/GS23/log/, providing the execution time.

(E4): A quick start to locally run an online task (using the online protocol inherited from hbMPC), in aid of triples generated from Dumbo-MPC's offline phase, is as follows:

1. cd Dumbo-MPC/dumbo-mpc/online.
2. ./preprocessing.sh <num_node> <num_input>. Both OptRanTriGen and AsyRanTriGen protocols are used to prepare triples and random shares, where random shares are served as inputs. For example, shuffling 64 inputs by 4 nodes requires to generate 4608 triple, which can be completed by ./proprecessing.sh 4 64. Besides, one-minus-ones shares are also required to generate for online shuffle. All preprocessed data are stored at Dumbo-MPC/dumbo-mpc/online/sharedata_test/.
3. Run ./run_shuffle.sh 4 64 for 4 nodes to shuffle 64 inputs. The experiment logs can be found at Dumbo-MPC/dumbo-mpc/online/log/.

A.5 Evaluation at Distributed AWS Instances

The above subsection outlines the approach used to evaluate the functionalities of our implementation. This subsection provides detailed guidance on reproducing the results reported in our paper. To distributedly run experiments across multiple AWS instances, first start an AWS instance (as the experiments' manager node) from the previously created AMI and modify the awsinit.sh in the directory remote/ as follows: replace -image-id with your-image-id; for -instance-type, we recommend c6a.8xlarge or better; replace -key-name with your-ssh-key-name; -security-group-ids you can delete this option if you are not familiar with it.

(AWS-E1) [Choices of batchsize] (Figure 6 and Figure 7): In this experiment, we test the relationship between throughput and batch size in the OptRanTriGen and AsyRanTriGen protocols. First, launch instances via ./awsinit.sh <num_node> where num_node can be 4, 10, 22, 31. Then, follow the next steps to evaluate OptRanTriGen and AsyRanTriGen protocols with different batch sizes.

- For OptRanTriGen, do the following steps:
  1. cd remote/OptRanTriGen_scripts.
  2. Configure ./changeconfig.sh by IP addresses of created AWS instances.
  3. Repeat to the execution of OptRanTriGen by varying <batchsize> among 1000, 5000, 10000 and 12000: ./launch_optrantrigen.sh <num_node> <batchsize>.
  4. Stop execution by ./terminate.sh, after the outputs are returned. Then, the experiment logs of all participating nodes can be downloaded by running ./scplog.sh <num_node> <batchsize>. The manger node will store the experiment logs at ./log_<num_node>_8x/test_<batchsize>/. These logs provide each node's execution time and communication cost in OptRanTriGen.
- For AsyRanTriGen, its test is similar to that of OptRanTriGen, except a couple of differences: (i) in Step 1, enter the directory AsyRanTriGen_scripts instead of OptRanTriGen_scripts; (ii) in Step 3, run the script ./launch_asyrantrigen.sh <num_node> <batchsize> instead.

(AWS-E2) [Triple throughput for LAN and WAN settings] (Figure 8 and Figure 9): This experiment evaluates the triple throughput with a fixed batch size of 5000 for four different offline protocols in both LAN and WAN settings, including OptRanTriGen and AsyRanTriGen from Dumbo-MPC, as well as GS23 and hbMPC.

First, we launch a few AWS instances via ./awsinit.sh <num_node> where num_node can be 4, 10, 22, 31. The testing processes in the LAN and WAN settings are almost same, except that in the WAN setting, we first run the bash script set_latency.sh in the remote directory, which uses Linux TC tool to restrict the upload bandwidth of each instance to 500 Mbps and set the network's RTT to 150 ms.

Then we evaluate four offline protocols with a fixed batch size of 5000 as follows.

1. cd remote/<AsyRanTriGen_scripts>, where the directory of scripts <OptRanTriGen_scripts> can be one of GS23_scripts, GS23_scripts, GS23_scripts, hbMPC_scripts.
2. Configure ./changeconfig.sh by IP addresses of created AWS instances.

3. Launch the specific offline protocol: `./<run_script> <num_node> 5000`. Here the parameter `<run_script>` is a bash script from `launch_asyrantrigen.sh`, `launch_optrantrigen.sh`, `launch_GS23.sh`, and `launch_hbmpc.sh`, representing the scripts to launch OptRanTriGen, AsyRanTriGen, GS23 and hbMPC, respectively.

4. Stop running by `./terminate.sh`, after the execution finishes. Then, we can read the experiment results by connecting these instances to view these logs or running `./scplog.sh <num_node> 5000` to download these logs. These logs will be stored under `./log_<num_node>_8x/test_5000/` at the manger node. The experiment logs provide the execution time and communication costs of each participating node.

(AWS-E3) [Communication cost] (Figure 10): In AWS-E2, we already evaluated the performance of four offline protocols. Their communication cost can be found in the test logs.

(AWS-E4) [Performance with fallback (Figure 11)]: First, launch AWS instances via the script `./awsinit.sh <num_node>` where `num_node` is set as 4. Then, evaluate three protocols (Dumbo-MPC, GS23 and hbMPC) with a fixed batch size of 5000 for `num_node`=4.

- Run Dumbo-MPC's offline phase (i.e. OptRanTriGen with fallback to AsyRanTriGen): The experiment design is similar to that of E-2. First, the fast-path protocol (OptRanTriGen) is executed to optimistically generate multiplication triples, until a malicious node is simulated at the 10-th round of OptRanTriGen. Following this, all nodes switch to the pessimistic path through a secure fallback mechanism, ensuring agreement on the triples generated during the fast path. Finally, the AsyRanTriGen protocol (pessimistic path) is executed by all nodes.

  1. `cd Dumbo-MPC_scripts`
  2. Configure `./changeconfig.sh` by IP addresses of created AWS instances.
  3. Launch Dumbo-MPC: `./launch_dumboMPC.sh 4 5000`.
  4. Terminate Dumbo-MPC and access to the experiment results: Run `./terminate.sh` to terminate all instances. Then, we can run the script `./scplog.sh 4 5000` to download the experiment logs from all participating nodes. These logs will be stored at `./log_4_8x/test_5000/` at the manger node. The logs provide the execution time for each step, including every round of the OptRanTriGen protocol, the fallback process, and the AsyRanTriGen protocol. Besides, the communication costs of OptRanTriGen and AsyRanTriGen are separately enumerated, and can be found at logs.

- Run GS23: same as running GS23 for the LAN setting in AWS-E2.

- Run hbMPC: same as running hbMPC for the LAN setting in AWS-E2.

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/usenixsec2025/.