Demo: Outsourcing Secure MPC to Untrusted Cloud Environments with Correctness Verification

*Oscar Bautista, *Kemal Akkaya and †Soamar Homsi
*Dept. of Electrical and Computer Engineering
Florida International University
Miami, FL 33174
Email: {obaut004, kakkaya}@fiu.edu
†Air Force Research Laboratory, Rome. NY, U.S.A
Email: soamar.homsi@us.af.mil

***Abstract*—Advances in Secure Multiparty Computation (MPC) is increasingly making this technology more attractive to solve problems on applications involving privacy-preserving computation. Considering the plethora of MPC protocols, some perform under a malicious security with dishonest majority attack model such as *SPDZ*, which also includes an important feature that enables the MPC nodes to verify the correctness of the computation. Despite those advances, in most cases, they consider that the computation nodes also supply the input data, which is not a realistic assumption for many practical use cases. In this demo, we show how our approach tackles the MPC outsourcing problem under malicious security with dishonest majority, while providing all the previous guarantees, namely, the verification of the correctness of the computation in addition to confidentiality of the inputs and outputs.**

***Index Terms*—MPC, SMPC, SPDZ protocol, MAC checking, client-server separation, multi-cloud environment, live demo**

I. Introduction

Secure Multiparty Computation (MPC) has emerged as a viable solution for practical applications in different domains where the privacy of the input data is of paramount importance.

Several MPC protocols tackle this problem under a variety of assumptions. We focus on the SPDZ protocol, which operates under a malicious security with dishonest majority attack model, and guarantees the privacy of the inputs with up to $n-1$ dishonest nodes. The original SPDZ protocol [1] also uses Message Authentication Codes (MACs) to verify the correctness of the computation. However, similarly to other MPC protocols, it assumes that the computation nodes also supply the input data, which does not represent many practical applications where data suppliers and MPC nodes are different entities.

Although this problem was tackled partially in [2], where clients supply the input data without exposing it to the MPC nodes and without being involved in the computation, that approach only worked for one client, and more importantly, it did not guarantee that the MPC nodes followed the protocol correctly. The original verification protocol will not work in this case, as this exposes the output to the MPC nodes before revealing it to the clients. For some applications, this may not be desirable; moreover, they might require keeping the output of the computation private from the MPC nodes deployed in the untrusted cloud.

We propose an approach to fill this gap by enabling verification of the correctness of the computation while maintaining the same levels of assurance of the client's data confidentiality. We introduce a separate entity named Honest Server (*HS*), which performs this verification task. This *HS* is different from the powerful MPC cloud nodes in that the former is hosted on-premises and does not require enormous computation resources. After the secure computation is completed, the MPC nodes send the shares of the results and corresponding MACs to the *HS*, which reconstructs the clear text outputs and verify their correctness. Depending on the result of the verification, the execution aborts, or it continues with passing the results back to the clients (or the application-dependent destination for this data) via secure channels.

In this demonstration, we show the execution of our approach to MPC in the outsourced setting using the SPDZ protocol, including client data outsourcing, secure computation, and aggregation of results for reconstruction, verification and transmission back to one of the clients.

II. Architecture

The MPC in the outsourced setting application computes a joint function on private inputs owned by specific parties called Clients. To preserve their privacy, those inputs are secret-shared to the computation nodes by following an input protocol. The demo MPC application is executed in the cloud, as it is the case with production use cases. All parties are connected by TCP links as shown in Fig. 1. Each component of the test system is described here-under.

a) MPC Nodes:

The MPC nodes or MPC servers are distributed nodes that perform the secure computation by following the SPDZ protocol. These MPC nodes are implemented in the untrusted cloud; for that reason, it is not desired that they get any knowledge about the clear text input or the output of the computation. In the outsourced setting, the MPC nodes receive

Approved for Public Release on 13 Sep 2021; Distribution Unlimited; Case Number: AFRL-2021-3084.

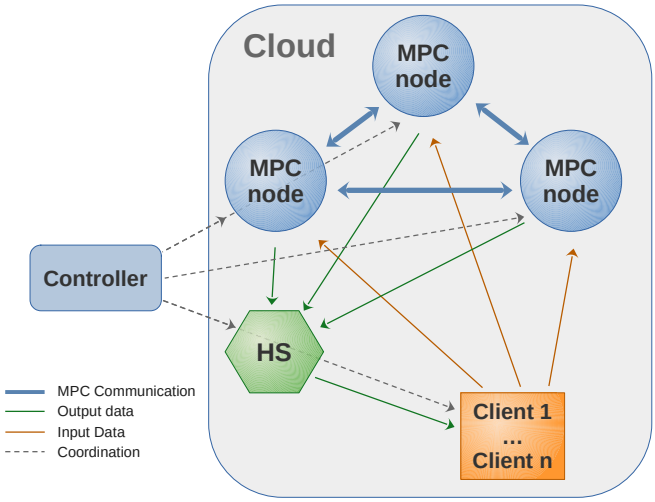

Fig. 1. General diagram of the setup for the demonstration.

masked inputs from the clients and send their corresponding results' shares to the *HS*. The MPC nodes are connected to each other using TCP links that communicate messages on rounds as the computation progresses.

b) Clients:

The client nodes provide the input data. These can be sensors, an aggregator of data from a group of sensors, or any other device that cannot perform the MPC directly because it does not have enough computation power, or because it is somehow inconvenient that such device performs the secure computation directly (for instance, location sensors or images captured by a swarm of drones). Therefore, they outsource the data and the computation to the MPC nodes. In an actual use case, clients can be geographically dispersed or form subgroups segregated over different locations. In any case, the clients do not communicate with each other; therefore, we can use a single Virtual Machine (VM) and spawn several client application instances to simulate a variable number of clients.

c) Honest Server (HS):

This server receives the shares of the outputs and verifies the correctness of the computation using the MAC shares that accompany each data share as defined by the SPDZ protocol. Depending on the specific application, the results are sent back to the clients or forwarded to the next stage in the application's data processing system. For instance, if the data comes from sensors, the next stage could be a central control system that makes decisions based on the result of the secure computation. Note that the *HS* opens the output of the computation but does not know about the input data. Additionally, the *HS* may or may not aid in the generation of the preprocessed material.

For the demonstration, we are deploying the *HS* in the cloud to simplify the setup and connectivity with the MPC nodes. Additionally, we use a controller node that orchestrates the execution of the whole MPC system, including clients and MPC nodes.

III. Implementation

We modified this [3] Java application, eliminating the limitation of a single client to implement the different MPC players for our approach. In addition, we added more functionality, such as allowing the specification of the IP addresses via command line parameters enabling a distributed deployment instead of running all the parties in the same host.

In general, the data that clients supply for the computation can be images, sensor readings, or a database with different features of many samples for a specific application. In any case, this data needs to be converted to integers and then masked during the execution of the input protocol that the MPC nodes and clients run to bring this data to the MPC nodes while preserving their privacy. For convenience, the Java application generates input data as vector of integers whose length is configurable via command line parameters.

Additionally, we expanded the Java application with the creation of the *HS*, which involved implementing the communication between this *HS* and the MPC nodes and the clients.

The controller node runs a simple Python application that establishes a connection to a predefined port setup to listen for connections on the cloud nodes (i.e., MPC node, *HS*, or clients). To start the MPC execution, the controller node generates strings composed of the command to run the MPC application and the corresponding command-line arguments, including IP addresses and the input data length. Finally, these strings are sent over the network to be executed by each player.

The type of the VMs created in GCP are e2-small which are configured with 2 vCPUs and 2GB of memory.

IV. Demonstration

For this demonstration, we deployed 3 MPC servers, the *HS*, and one additional host spawning different processes that implement the clients on Google Cloud Platform (GCP). Specifically, we selected a single region to deploy all players. The resulting performance is equivalent to running the experiment in a LAN environment, where all point-to-point delays are very similar.

The whole list of experiments for the full paper included several MPC executions varying parameters like number of MPC servers, number of clients, and length of input data. Nonetheless, from the execution point of view, all those are performed in the same way. Therefore, this demonstration focuses on the setup and preparation step by step of one of those experiments, followed by the execution of the MPC coordinated by the controller node.

In MPC, multiplication and addition are the basic operations which serve as components of more complex operations. (E.g., matrix multiplication, linear regression application for machine learning). Therefore, for the demonstration, the MPC system computes the multiplication of the inputs from different clients. In this way, when 3 clients supply their private values, the MPC servers compute the multiplication of the 3 values they provide. Furthermore, to simulate the supply of continuous data, each client supplies a vector of values whose elements are processed consecutively. E.g., each client supplies

```
02:00:10.241 [SCE-1] INFO dk.alexandra.fresco.framework.sce.SecureComputationEngineImpl - Running application: dk.alexandra.fresco.outsourcing.server.ddnnt.Ddn
ntInputServer$UnMaskingApp@6800c1dd using protocol suite: dk.alexandra.fresco.suite.spdz.SpdzProtocolSuite@66c722ad
02:00:10.267 [SCE-1] DEBUG dk.alexandra.fresco.framework.sce.SecureComputationEngineImpl - Evaluator done. Evaluated a total of 12 native protocols in 1 batche
s.
02:00:10.267 [SCE-1] INFO dk.alexandra.fresco.framework.sce.SecureComputationEngineImpl - The application dk.alexandra.fresco.outsourcing.server.ddnnt.DdnntInp
utServer$UnMaskingApp@6800c1dd finished evaluation in 14 ms.
Input share: BigIntegerFieldElement{value=232191425190458805093650563062316411809, modulus=BigIntegerModulus{value=340282366920938463463374607431768211283}}
Input share: BigIntegerFieldElement{value=158442290897668720255728732490919624428, modulus=BigIntegerModulus{value=340282366920938463463374607431768211283}}
Input share: BigIntegerFieldElement{value=282088300612117111253904806175662534075, modulus=BigIntegerModulus{value=340282366920938463463374607431768211283}}
Input share: BigIntegerFieldElement{value=176844117123686194220422907618133923369, modulus=BigIntegerModulus{value=340282366920938463463374607431768211283}}
Input share: BigIntegerFieldElement{value=290285451278480244645797828691339705114, modulus=BigIntegerModulus{value=340282366920938463463374607431768211283}}
Input share: BigIntegerFieldElement{value=305849302458745290516689059466522953674, modulus=BigIntegerModulus{value=340282366920938463463374607431768211283}}
Input share: BigIntegerFieldElement{value=795275861406346496264416826964887096, modulus=BigIntegerModulus{value=340282366920938463463374607431768211283}}
Input share: BigIntegerFieldElement{value=336145581180499098126894022499537491253, modulus=BigIntegerModulus{value=340282366920938463463374607431768211283}}
Input share: BigIntegerFieldElement{value=143098364189754874939021762899831515554, modulus=BigIntegerModulus{value=340282366920938463463374607431768211283}}
Input share: BigIntegerFieldElement{value=550126880277583181770870793911163122l9, modulus=BigIntegerModulus{value=340282366920938463463374607431768211283}}
```

(a) MPC server

```
Receiving outputs from server 2: 1 2 3 4
02:00:10.350 [main] INFO dk.alexandra.fresco.outsourcing.client.ddnnt.DemoDdnntOutputClient - C4: Received output shares from server Party(2, 34.85.204.4:8044)
MAC reconstructed from shares:  107956053069900749088271366070300453305
MAC calculated by the HS:       107956053069900749088271366070300453305
MAC Check Passed!
MAC reconstructed from shares:  143014295998716158064937529079265516 77
MAC calculated by the HS:       143014295998716158064937529079265516 77
MAC Check Passed!
MAC reconstructed from shares:  223978696817206121751208380505059886623
MAC calculated by the HS:       223978696817206121751208380505059886623
MAC Check Passed!
MAC reconstructed from shares:  194898771432942423194013393175351768863
MAC calculated by the HS:       194898771432942423194013393175351768863
MAC Check Passed!

Outputs received: [8624, 10296, 35547, 4761]
```

(b) Honest server

Fig. 2. Extract from logs of execution of a sample secure MPC.

a vector of size n, then the output of the computation is also a vector of size n.

The demonstration shows how the input outsourcing preserves the privacy of the client's data; it also exemplifies through the logs of execution the correctness verification through MAC checking as shown in Fig. 2.

The general testing procedure follows this sequence:

1) Start the different VMs; after a few seconds, we could see the public IP addresses assigned to each of those VMs.
2) Open the file config.py located in the controller's node application folder and update the IP address according to the new IP assignment.
3) At the controller node, configure the number of executions and the parameters for each run (# of servers, # of clients, and the input data length).
4) Run the test Python application in the VMs. This application starts listening for a connection from the controller node, which will provide the commands for the MPC execution.
5) Run the test application in the controller node and follow the instructions on the screen.
6) The system will perform either a single execution or a series of executions depending on the controller node's test configuration. A sample *HS*'s output log is shown in Fig. 2(b).

V. Conclusion

Through this live demonstration we show the implementation of our proposed approach for MPC in the outsourced setting using SPDZ protocol. In this approach, sensitive data and computation is outsourced to powerful MPC servers hosted in the untrusted cloud, maintaining the privacy of the inputs and outputs while also verifying the correctness of the secure computation.

Acknowledgements

This research was supported in part by the Air Force Research Laboratory/Information Directorate's (AFRL/RI 's) Internship Program for summer 2021, the 2020 Summer Faculty Fellowship Program (SFFP) through the Air Force Office of Scientific Research (AFOSR), Contract Numbers FA8750-15-3-6003 and FA9550-15-0001, and the U.S. National Science Foundation, award number USNSF-1663051.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

References

[1] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual Cryptology Conference*. Springer, 2012, pp. 643–662.

[2] I. Damgård, K. Damgård, K. Nielsen, P. S. Nordholt, and T. Toft, "Confidential benchmarking based on multiparty computation," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 169–187.

[3] Alexandra Institute, "FRESCO - a FRamework for Efficient Secure COmputation running in the outsource setting," https://github.com/aicis/fresco-outsourcing.