Fair Multi-Party Computation using Public Bulletin Boards

Anasuya Acharya
CSE Department
IIT Bombay
Mumbai, India
17305R004@iitb.ac.in

Anugrah Gari
CSE Department
IIT Bombay
Mumbai, India
173050078@iitb.ac.in

Digvijaysingh Gour
CSE Department
IIT Bombay
Mumbai, India
17305R001@iitb.ac.in

Kuber Gautam
CSE Department
IIT Bombay
Mumbai, India
173050087@iitb.ac.in

***Abstract*—The paper presents a construction for Fair Multi-Party Computation using Public Bulletin Boards. We discuss existing implementations of such constructions that use secure hardware i.e., Intel SGX; and witness encryption. Here we present a different approach at fair MPC without using a secure hardware, using Ciphertext Policy - Attribute Based Encryption (CPABE). We detail the design aspects, how it is an improvement over existing constructions, and argue the security and feasibility for such a construction.**

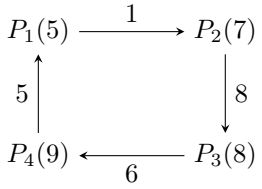
***Index Terms*—MPC, Fair MPC, bulletin boards, witness encryption, Intel SGX, CP-ABE**

I. Introduction

The basic idea of Multi-Party computation or MPC is to have different parties, wanting to communicate with each other and exchanging some values over point to point authenticated channels according to a predefined protocol. The goal of communication is to compute a function applied on the values of each party and to find the final output or result of that function. This should be done in such a manner that no party gets to know the value or private input of another party and yet they are able to generate the final output of the function applied on them.

Let us consider an example of 4 parties communicating with each other and trying to compute the sum modulo 10 of all the private inputs of the parties. Let us take 4 parties as P_1, P_2, P_3 & P_4.

- Let the input of parties be as follows:
  - P_1 : 5.
  - P_2 : 7.
  - P_3 : 8.
  - P_4 : 9.
- Funtion to be used : $f(x_1, x_2, x_3, x_4) = (x_1 + x_2 + x_3 + x_4) mod_{10}$
- Random number to be added : 6.
- First party adds it's input to random number modulo 10 and sends it to second party.
- The second party takes input from the first party and adds its input to it. This process is repeated by third party and so on.
- The first party receives the final output sent by fourth party and subtracts the random number from it.
- The value generated is the summation of all inputs modulo 10. This is then broadcast to all other parties.



MPC has lots of real world applications like voting, data analysis on confidential data, private information retrieval, distributed homomorphic encryption, and in any general scenario in which a public output needs to be computed from mutually private inputs without revealing any information about them.

A. Fairness

The notion of fairness is important in case of MPC. In the above example, party P_1 gets the final output of the function. Party P_1 now received the end output of the function. It may happen that P_1 after knowing the end output can abort the protocol, or calm network failure which is biased to other parties as now only P_1 knows the end output of the function and nobody else does. So to make it unbiased it is required that either all parties know the end output of the function or none of the parties know.

1) Definition: Fairness in MPC simply says that, either all parties receives the end output or no party does.

2) Fair MPC: The desirable definition of MPC itself says that, either all party know the end output of the function or no party does. This is referred as *fair MPC*.

But fairness comes with a **cost**. It has being proved that fair MPC for general functions is impossible to realize unless all of the parties are honest. This fails in any active adversarial setting. Lot of research work is going on towards the study of mitigation of this fairness problem. In particular, two prominent lines of research have being emerged over the years. The first research direction is towards the problem of achieving fairness in the standard model for restricted classes for function. And the second research direction studies fairness for general function by augmenting the computation

model or by relaxing the definition of fairness.

This paper explores a construction for achieving completely fair MPC using an external functionality, i.e. Bulletin Boards.

B. Adversaries

Adversaries in MPC may control one or more parties and has visible to themselves the joint view of all those parties. They can mainly be of two types: *Active adversary* and *Passive adversary*

- *Active Adversary*: Adversary that tries to manipulate the protocol or change the output by sending messages with invalid content, communicating out of turn or sending protocol aborts.
- *Passive Adversary*: Adversary that is honest but curious. It only listens to the protocol flow through the parties it controls and does nothing to disrupt its functioning i.e it knows the output but doesn't change it.

II. Aim

Our aim is to construct fair protocols out of any existing MPC protocol keeping the adversarial setup the same as the original; be it against passive or active adversaries, having majority honest parties and handling protocol aborts for general functions. In order to achieve this we use a public bulletin board functionality, Shamir Secret Sharing and Ciphertext-Policy Attribute Based Encryption without using secure hardware. The implementation is done using python. The correctness and privacy properties of the underlying MPC scheme are completely preserved.

Public Bulletin Board: In order to achieve fairness in MPC for general functions, we model the bulletin board as a public immutable ledger on which anyone is allowed to publish arbitrary strings. Bulletin board contains records of data along with their 'proof of publish'. The ledger is public, it means anyone can see the content. And it is also immutable means the content cannot be erased, nor can the proof of publish be forged. The implementation of such public bulletin board already exist in practice like the Google's certification transparency project which logs issued certificate and the blockchain-based ledgers such as Etherium and Bitcoin.

In our work, we model our own public bulletin board using a shared file with a proof of work based dependency ledger. This file preserves all the functionality as that of public bulletin board.

III. Literature Review

A. Base Paper Overview

This paper defines a method of implementing multi-party computation with public bulletin boards, presenting two constructions: using secure hardware(SGX), and using witness encryption.

For the construction that achieves fair MPC using Witness Encryption, any normal MPC protocol that computes the result $y = f(x_1, ..., x_n)$ is converted into a protocol that computes $Enc(y)$, that is an encryption of the result under a Witness Encryption Scheme. This is then posted onto the Bulletin Board from where it becomes publicly available to all the parties, along with it's proof of publish. Each party then retrieves this ciphertext and proof of publish and uses the decryption algorithm in which this proof of publish acts as the witness (decryption key). This ensures that either all the parties get the output (conditioned on the availability of the proof of publish) or none of them do. Witness Encryption, however, does not have any computationally efficient implementations (the only implementation known being using multilinear maps) and therefore this is just a feasibility result. The following section elaborates the protocol construction for achieving fair MPC using Secure Hardware, which was the computationally efficient implementation given in the paper.

B. Base Paper Protocol

1) **Key Generation** : Key is generated in SGX of any one party and shares are created from that key.
2) **Shares of Key** : Key share of each party is sent to them through the secure authenticated channels between SGX components.
3) **Commit** : Shares are then committed to the public bulletin board so that no party can change the key during the protocol execution.
4) **Protocol** : The original protocol is run using SPDZ MPC library in which the private inputs of each party are used to compute the function output.
5) **Encryption of Output** : The computation on output of the protocol is then ported onto the SGX of the parties that produces an AES encryption of the final output. This is then put on the public bulletin board for everybody to see.
6) **Reading of Output** : Output is then read by all parties.
7) Public Release Token is then generated and distributed by each party which is used for revealing the key shares and decrypting the output by all the parties in their own systems.

C. Shortcomings of the Protocol in Base Paper

- SGX, the only known 'secure hardware' realization, and the one used in the construction has already been hacked, meaning it is not secure anymore. Therefore, the existence of secure hardware is questionable at the moment.
- Symmetric key cryptography is used for encryption, meaning key for encryption and decryption are same which makes it vulnerable to a lot of attacks. In the construction, fairness is guaranteed by the fact that decryption key can only be computed after the result ciphertext has reached all parties. If the parties know the key while encryption, they need not publish to decrypt. The base

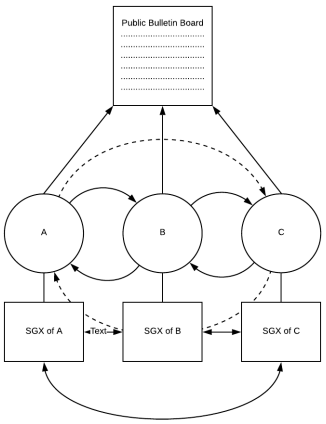


Fig. 1. MPC with BB and SGX

paper protocol averts this by porting all cryptographic computations to the SGX.
- Centralized key generation makes the protocol dependable on one single system which can be harmful for the protocol, creating a single point of failure.
- Proof of Publish of the encrypted output is not used. The protocol instead uses broadcasts of the release tokens for the commits. In such a case, the last party to release may wait for all other release tokens, decrypt to get the output and refuse to give its own release token to other parties, making the protocol unfair.

D. AES

Advance Encryption Standard (AES) is a symmetric key encryption algorithm which takes 128 bit data and has key of size 128, 192 or 256 bit. AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix. The number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key. Each round comprise of four sub-processes that are Byte substitution, Shift row, Mix Column, Add Round key.

E. Intel SGX

Intels Software Guard Extensions (SGX) is a set of extensions to the Intel architecture that aims to provide integrity and confidentiality guarantees to security sensitive computation performed on a computer where all the privileged software (kernel, hypervisor, etc) is potentially malicious. Intel SGX allows user code to allocate a private memory region called the *Enclave* which protect the user code from other processes running at high privilege.

F. CP-ABE

Attribute Based Encryption is a cryptographic primitive that uses encryption to enforce access control. ABE cryptosystems are public key cryptosystems that consist of a Master Secret Key and a Public Key. Consider an organization with different departments and different levels of employees (users) that access a shared database but have different permissions and visibility levels to that data. In such systems, the ABE public key is used to encrypt messages into ciphertext. Then, depending on the labels and permissions of each user, each user is assigned a key derived from the master secret key such that only the keys of users with appropriate access rights to the message can actually decrypt the message.

These permission labels of the users are called the user's 'attributes', and the rule base dictating which user types can access the data is modelled as a monotonic boolean formula tree called the 'policy'. The leaves of this tree perform equality checks on weather the attributes required attributes are present and propagate the true or false value to the 'and' or 'or' gates that form the internal nodes of the tree. The decryption algorithm performs this policy matching using the key and the ciphertext and only decrypts to reveal the ciphertext if the policy evaluates to true.

Attribute based encryption comes in two flavours: Ciphertext Policy ABE, and Key Policy ABE. In CP-ABE, the policy tree is encoded and defined within the ciphertext on encryption while the attributes for each user is encoded into the key. The opposite holds for KP-ABE. In our implementation, we use CP-ABE.

Practical and library implementations of such schemes using bilinear maps and pairings are already available and this is realizable using computationally feasible methods. In our implementation, we use the CP-ABE scheme as described in [4] as part of the MPC protocol to condition the decryption of the final result on the availability of a valid Proof of Publish.

G. Shamir Secret Sharing

Shamir Secret Sharing is a scheme where secret (data/key) is divided into many shares and each participant has its own unique share where some of the share or all of them are needed in order to reconstruct that particular secret. Here the goal is to divide the secret *S* (e.g data or key) into n shares of secret $S_1, S_2....S_n$ such that:

- Knowledge of any *k* or more S_i shares make *S* easy to reconstruct.
- Knowledge of any *k-1* or fewer S_i shares leave *S* completely undetermined (in the sense that all its possible values are equally likely).
- This scheme is called *(n,k)* threshold scheme. If k=n then all participants are required to reconstruct the secret.

This scheme supports operations like share addition, constant multiplication with shares - both of with are linear

operations. It also supports share multiplication under share switching in order to readjust the (n,t) values needed for reconstruction. This makes the scheme fully homomorphic. Finally, as share switching (converting from (n,t) shares to (n,t/2)-secret sharing) needs to be performed for multiplication, this scheme is only secure in honest majority settings.

IV. Protocol Construction

The protocol we designed mainly focuses on the computation of inputs of different parties without anybody actually revealing them to others. This is achieved by dividing the private inputs into several parts and sending them out to each party, their respective share, in such a way that unless all the shares are put together, any other subset reveals no information about the message. This is done using *Shamir Secret Sharing*.

Each party privately creates part of the Master Shared Key using their private randomness, creates a commit value out of it, and publishes it on the bulletin board. Then the function is performed on the inputs and encrypted output is published. Finally, each party read output and decrypts it using CPABE conditioned on availability and validty of proof of publish. Hence the goal is achieved.

For this to work out, we have defined some entities with specific functionality for our implementation:

- **Simulator**: This is an entity which is used for synchronizing the parties and to give directions. The job of the simulator is to inform the parties what action to perform and when to perform it by signalling each party and alerting it of the steps in the protocol.
- N different parties. These are the clients which are the end users of the protocol and for which this protocol is designed. Each of them have their own private inputs and are connected to all others using point to point secure channels.
- **Public Bulletin Board**: This is a public forum which is used to ensure a message published is visible to all parties. Any party can access it for reading but writing to this can be done using the publish method. This is a shared file between the different parties. The publish method takes as argument the data to be published and creates a proof of publish using a hash based proof of work of the data, a nonce and the previous record's proof of publish. This ensures immutability in the ledger.
- **Decryption**: This is a function used for decrypting to obtain the final output. Decryption is done using CPABE if all the parties have have published their encrypted output share on the public bulletin board, conditioned on availability of all valid proofs of publish.

V. Protocol Functionality

This section describes how the construction works theoretically and how each party gives their input, performs operations and output is generated and is broadcasted among all parties; and how the encryption and decryption is done.

1) The private key (Master Secret Key) for the CPABE scheme is a two touple. Each party generates two private random values, one to contribute to each part of the MSK.
2) These key shares **Shares** are then committed on the bulletin board by the respective parties as a proof of validity so that no party can change their share in the course of the protocol.
3) A preliminary MPC protocol with the private key shares as private inputs is performed to get as a result the Public Key for CPABE encryption. This Public Key is then published onto the bulletin board and made accessible by all parties.
4) Each party shares their private input using (n,n)-Secret Sharing to all other parties.
5) MPC is then carried out and shares of a function of the inputs are generated and stored by each party. Each party then takes the final shares that it has and encrypts it using the Public Key.
6) The CP-ABE decryption key involves one common secret derived as a function of the MSK, along with attribute string encodings. This common secret is computed by all parties using MSK shares and published to be made available to all parties.
7) Encrypted output shares are then published on the bulletin board in order for everybody to see.
8) Each party reads the output shares and uses the decryption functionality. The decryption function takes as input all the shares, the proof of publishes, and the common secret, uses it to generate attribute strings, evaluate the policy and decrypt the shares. Decrypted output is given back to the party calling this decrpytion function.
9) Finally, each party then reconstructs the shares to obtain the final output.
10) In this manner this protocol becomes fair as it gives the output to all the parties and since even if any party aborts the protocol, still the decryption function works for the rest of them, according to its design.

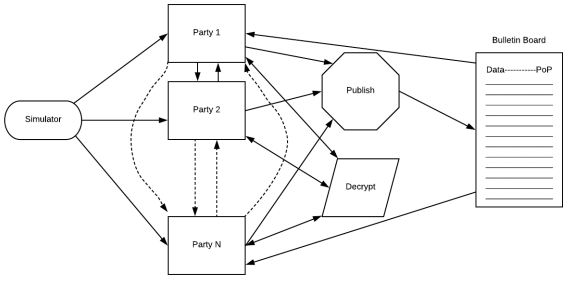


Fig. 2. Implementation Diagram

VI. Implementation Details

We use Python for implementing this protocol. The implementation is for the 3 party case, that can easily be generalized to the N party case. Parties are connected to each other by use of Client Server Mechanism. By using Clients of each Party , we send a message and by Servers of each Party we receive a message and then take further actions. Entire code is developed from scratch and it makes only use of native modules and numpy module.

We use a shared file to simulate the bulletin board. The bulletin board is modelled like a blockchain of records with the proof of work hash being a function of data in that record and that of the previous proof of work. The difficulty for our purpose is set as a constant in the publish functionality. While accessing/simulating bulletin board we made sure that protocol for accessing files should be similar to protocol for accessing bulletin board. All access to file are bottom up rather than top down, and not random access.

For Proof of Publish , we have made use of SHA-256 Hash and to make hash stronger , we increased its difficulty by changing nonce and hashing it unless its zero padding goes to desired level. For field operation of power function we used Fast Exponentiation Algorithm. For evaluating Modular Inverse , we made use of Extended Euclidean Algorithm.For evaluation inverse of Z-Matrix as discussed above (share switching and multiplication in Shamir Secret Sharing) we made use of Lagrange Interpolation. For exchanging shares among parties we made use of Shamir Secret Sharing. We also made use of Bilinear Maps and pairing for CP-ABE so that entire mechanism works out efficiently.
The code of the implementation can be viewed here.

VII. Conclusion & Future Work

To conclude, we give a construction of fair MPC using our own implementation of public bulletin board, Shamir Secret Sharing and CPABE. There were some challenges which we faced while implementing fairness, as we wanted to remove the use of SGX. We had to change the encryption algorithm from symmetric key encryption (AES) to asymmetric key encryption as using AES without secure hardware makes it vulnerable leading to more attacks. In order to achieve fairness, the input and intermediate values has to be distributed among all *n* parties so that when $t <= n$ (t = n here) parties comes together, they can decrypt the output of the function. This distribution is done using Shamir Secret Sharing.

There is still a lot of work which can be done like instead of making public bulletin board centralized, we can make it decentralized like Blockchain in order to make it fault tolerance. We require decryption to be a separate functionality that includes attribute generation for CP-ABE as key attributes cannot be generated before having the Proof of Publish, an parties cannot be given the leway to manipulate the generation of attributes. Future constructions may have decryption as part of the party itself and not an external entity. Since it is a long-standed proven result that complete fairness is not possible to achieve using only MPC, we try different approaches to achieve this by augmenting MPC with widely available and efficiently implementable external functionalities.

References

[1] Rai Choudhuri, Arka & Green, Matthew & Jain, Abhishek & Kaptchuk, Gabriel & Miers, Ian. (2017). Fairness in an Unfair World: Fair Multiparty Computation from Public Bulletin Boards. 719 − 728. 10.1145/3133956.3134092. URL: https://acmccs.github.io/papers/p719-choudhuriA.pdf

[2] Bethencourt, John & Sahai, Amit & Waters, Brent. (2007). Ciphertext-Policy Attribute-Based Encryption. Proceedings - IEEE Symposium on Security and Privacy. 321 − 334. 10.1109/SP.2007.11. URL: https://www.cs.utexas.edu/~bwaters/publications/papers/cp-abe.pdf

[3] Garg, Sanjam & Gentry, Craig & Sahai, Amit & Waters, Brent. (2013). Witness encryption and its applications. Proceedings of the Annual ACM Symposium on Theory of Computing. 467 − 476. 10.1145/2488608.2488667. URL: https://eprint.iacr.org/2013/258.pdf

[4] Shamir, Adi. (1979). How to Share a Secret. Commun. ACM. 22. 612 − 613. 10.1145/359168.359176. URL: https://cs.jhu.edu/~sdoshi/crypto/papers/shamirturing.pdf

[5] Code of implementation at this URL: https://github.com/digiblackdranzer/mpc.git