Secure Multi-Party Computation on Peerster

Junzhen Lou
junzhen.lou@epfl.ch

Yuening Yang
yuening.yang@epfl.ch

Chen Chang Lew
chen.lew@epfl.ch

I. Executive Summary - 1 page

Paragraph 1: Background about the domain you chose [**?**] Multi-Party Computation (MPC) protocols allow multiple parties with private inputs to jointly compute a function while maintaining the privacy of their inputs. The concept of MPC was first introduced in the 1980s by Andrew Yao, who proposed the Millionaires' Problem. MPC has many vital applications in areas such as privacy-preserving data analysis, secure two-party computation, secure multi-party computation, electronic voting, and more. Permissioned Blockchain is a type of distributed ledger whose entrance and consensus participation are only restricted to certain users. Because of the access control and user authentication, it supports consensuses with higher throughput which might be vulnerable in the public chains. It is usually used between companies and has several application areas such as supply chain management, financial transactions, healthcare, government, etc. One of the famous permissioned blockchains is Hyperledger Fabric [1] developed under the Linux Foundation's Hyperledger project.

The goal of this project is to deliver a p2p community with access control, where participants can start an MPC, sell data for MPC to use without worrying about leakage, participate in MPC to earn fees, etc. The incentive is to make better use of the data while protecting privacy without relying on traditional methods such as third-party and to incentive people to use MPC. The system is built upon MPC and Blockchain, adopts synchronous protocols, and is able to achieve information-theoretic security against passive adversaries in fewer than half of the total nodes.

Our system has three main building blocks: Secure channel, MPC module, and Blockchain module. The secure channel is the basic tool used by MPC and Blockchain to send messages and support end-to-end encryption using RSA. MPC module handles the whole multi-party computation, including expression analysis, mathematical calculation, secret share, and interpolation. Blockchain module provides a permissioned blockchain to help people sell data in MPC without a central authority. In addition, it is able to filter potential adversaries for MPC by access control and authentication. The permissioned blockchain is also used to create consensus on the network assets and MPC participants, as well as distribute nodes' RSA encryption keys.

The secure channel is integrated into the node's message module. It keeps track of the RSA public keys of the nodes and supports RSA encryption and decryption for messages of all lengths. The MPC protocol uses Polynomial-based Shamir Secret Sharing and Lagrange Interpolation to serve the functionalities of Input Division and Output Reconstruction, respectively. Secure Addition Protocol and Secure Multiplication Protocol are executed to perform computations on basic calculation units of the target function. All computations are performed in a finite field. The blockchain module implements a permissioned blockchain where only chain members are able to perform actions on the blockchain. It offers miner and validator daemons, a transaction pool, and a transaction executor that verifies and executes transactions. Only specific types of transactions can be executed with pre-defined actions to minimize the attack interface. In addition, the blockchain module provides a credit system to select the next miner based on coinage and account balances, as well as a notification center where other modules of the nodes are able to register themselves to be notified when a new transaction is committed to the blockchain. To integrate these systems, we propose a full MPC process that involves sending a PreMPC transaction that proposes a new MPC and locks deposit for it, starting MPC that does the real multi-party computation, and sending PostMPCs to claim awards.

In summary, the additional features added approximately 15000 lines to the original Peerster project, which primarily consisted of basic communication functionality like unicast, broadcast, and gossip. Specifically, nearly 4000 lines were added for MPC logic and communication, and nearly 2000 lines for modifications to the gossip protocol and the encryption channel. Blockchain module takes almost 7000 lines of codes and the remaining lines are for the performance tests, the interactive CLI tool, and UI.

Correctness tests are provided at function, module, and system levels to ensure the system works as expected. Performance tests show that the execution time increases in a polynomial manner as the number of peers increase, which aligns with the theoretical expectation since the number of transactions increases as n square. In addition, this paper provides a detailed security analysis and concludes the system support access control, authentication, and accountability. It can provide data privacy as long as the number of passive attackers does not go over half of the total participants, and it is able to protect the on-chain properties against active attackers. However, the system requires all participants to stay online for both MPC and blockchain parts

II. Background and Related Work

In this section, the paper briefly mentions the basic concepts of Multi-Parity Computation and permissioned blockchain, which are also two main building blocks of MPC Peerster.

A. Multi-Party Computation

Secure Multi-party computation (MPC) focuses on the design and implementation of protocols for allowing a group of parties to jointly compute a function over their inputs without revealing any additional information to each other beyond the output of the function. MPC protocols are designed to provide strong privacy and security guarantees, ensuring that the inputs and internal workings of the computation remain secret even if some of the parties are compromised or adversarial.

MPC protocols can be implemented on top of a distributed system, with each party running a separate node in the system and participating in the MPC protocol to jointly compute the desired function. This can be useful in situations where the parties do not fully trust each other or where it is important to maintain the privacy of the data.

MPC has a wide range of potential applications, including secure distributed machine learning, privacy-preserving data analytics, and other scenarios where multiple parties wish to jointly compute a function over sensitive data. There have been numerous developments in MPC, including the design of protocols for various different types of functions and settings, as well as the construction of practical systems in real-world settings.

In academia, in 2017, a group of researchers proposed a protocol for privately releasing the output of a database query using MPC. The protocol allows a user to query a database and receive the results of the query without revealing the specific query to the database owner. In 2019, researchers from the University of California, Berkeley and the University of Maryland developed a protocol for privately releasing the output of a machine learning model using MPC. The protocol allows multiple parties to collaboratively train a model over their private data, and then privately release the output of the model to a third party without revealing the data or the model itself.

In industry, Unbound Tech released a commercial MPC platform for use in financial services and other industries. The platform allows multiple parties to perform joint computations over sensitive data without revealing the data to each other. The Privacy Pass extension for the Google Chrome web browser uses MPC to allow users to authenticate themselves to websites without revealing their personal information. The Enigma Catalyst platform is a decentralized platform for building and deploying MPC-based applications in the financial sector. And the MPC-based protocol Confidential Transactions, developed by Blockstream, is used in the cryptocurrency Bitcoin to enhance the privacy of transactions on the network.

Additionally, there are a number of active areas of research in the field of multi-party computation (MPC). Some current areas of focus include: developing efficient protocols for specific types of functions; investigating the security and efficiency of MPC protocols, including their resilience to various types of attacks and their performance in terms of computation and communication cost; constructing and implementing MPC protocols in practical settings, such as those involving large numbers of parties or those with limited computational resources; studying the foundations of MPC, including the development of formal models for capturing the security guarantees provided by MPC protocols and the construction of general-purpose MPC frameworks, and investigating the potential applications of MPC in areas such as secure distributed machine learning and privacy-preserving data analytics.

B. Permissioned Blockchain

The concept of blockchain is first proposed by Bitcoin [2], which serves as a decentralized digital ledger that records transactions across a network of computers. As its name indicates, a blockchain is a sequence of blocks chained by its hashes to protect data integrity. The decentralized nature of the technology means that no single entity controls the data, and it solves the trust problem of using the ”third party” to do transactions. Inspired by Bitcoin, the Ethereum blockchain defines the blocks as different versions of a state machine, and transactions are viewed as state transitions [3]. It allows user-defined transition actions which can be activated by sending specific transactions.

Different from the previously mentioned public blockchains, a permissioned blockchain is a type of distributed ledger whose entrance and consensus participation are only restricted to certain users. Because joins needs to be authorized, participants in a permissioned blockchain are usually authenticated, resulting in all actions in the network becoming accountable to a real party. In contrast, a public blockchain, for example, Bitcoin and Ethereum which is permissionless to join, allows anyone to participate in the consensus process, and users of them interact often anonymously with each other.

Permissioned blockchains are typically used between enterprises where the participants are known and trusted. It also provides greater access control to the network data and the operations, which suits the scenario that needs to be accessible by trusted parties, such as supply chain management, financial transactions, healthcare, government, etc. Compared to public permissionless blockchains, permissioned ones are more efficient and faster but also have less transparency to the public outside the chain.

Hyberledger Fabric, by which this project's design of the blockchain is inspired, is one of the most widely-used permissioned blockchain frameworks developed under the Linux Foundation's Hyperledger project [1]. It offers smart contract execution and a unique consensus approach considering that the chain participants are relatively trusted and authenticated. In case of this, there is no fork in the chain, the design speeds are fast and transactions in the chain are trackable and irreversible [4]. In addition, in each permissioned blockchain, which is called a channel in Hyperledger fabric, the data are

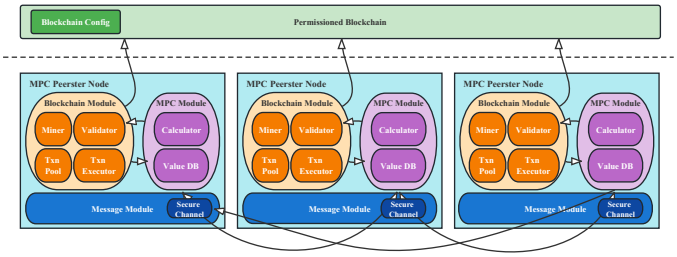

Fig. 1: System Architecture: The system has two main modules, MPC module, and Blockchain module, both making use of the Messaging module to communicate with other nodes and using function calls to communicate internally.

private from Fabric nodes not in that channel and from the outside world.

III. MPC Peerster Design

In this section, the paper will present a high-level view of the system and explains its two main building blocks of it: the Multi-party computation (MPC) module and the permissioned blockchain module as shown in Fig.1. It will then talks about how these two modules are integrated, i.e. the full version of the system, as well as provide another option to quickly start MPC without blockchain, i.e. the quick setup of the system.

A. Multi-Party Computation

As an important component of distributed computing, MPC protocols are always used in conjunction with distributed systems to enable secure distributed computation over sensitive data [5]. In such systems, multiple parties may each have their own data that they wish to jointly process or analyze without revealing the data to each other. MPC protocols can be used to enable the parties to collaboratively compute a function over their data, without revealing the data itself to any of the parties or to any external observers.

In this paper, we present an MPC protocol that performs expression analysis locally and uses Shamir Secret Sharing and Lagrange Interpolation to enable the parties to securely compute a function over their data with no privacy leakage [6]. On top of secure communication channels, this protocol achieves information-theoretic security (unconditional security) against passive adversaries. The design and implementation of MPC involve the following steps:

Function Identification. The first step in the protocol is to identify the function that the parties want to compute. The system will analyze the input string of an expression to decide which actions to perform and the sequence of doing them and always achieves the input-output relationship.

Input Data Division. The second step is to divide the input data among the parties in such a way that each party holds only a portion of the data. This process is called (threshold) secure secret sharing and can be done using techniques such as Shamir Secret Sharing using polynomials, Blakley's scheme using multi-dimensional hyperplanes, Asmuth-Bloom scheme using the Chinese Remainder Theorem, etc. In our system, we use Shamir Secret Sharing. Compared with Blakley's scheme, Shamir's scheme is more space-efficient: while Shamir's shares are each only as large as the original secret, Blakley's shares are t times larger, where t is the threshold number of players. Moreover, there is one critical theoretical difference between Shamir's scheme and the Asmuth-Bloom scheme. Shamir's scheme can achieve information-theoretic level security; specifically, if it is implemented carefully in a way that the polynomial coefficients are chosen randomly and uniformly, an attacker with fewer shares than the threshold (the minimum number of shares required for reconstruction) gains absolutely no information about the shared secret, even with unlimited computation power. In contrast, the Asmuth-Bloom scheme does leak some probabilistic information. An attacker can gain some probabilistic information about the shared secret, even if they have fewer shares than the threshold. At last, Shamir Secret Sharing is even more preferred for its flexibility. It allows specifying which subsets of parties are able to reconstruct the input data. This can be useful in situations where it is necessary to restrict access to the input data to only certain parties. Shamir Secret Sharing is a widely-used and well-studied secret-sharing scheme, and there is a large body of literature and tools available for its use and analysis. This makes it a reliable and well-understood choice for use in MPC protocols.

Computation Execution. The third step is to perform computations on the shares on each node. Parties can then use techniques such as secure function evaluation (SFE) to perform the computation over their respective inputs without revealing the inputs to each other. In our system, we take into consideration the BGW (Ben-Or, Goldwasser, and Widgerson) protocol. We decompose the target function into additions and multiplications, then perform secure addition protocol and secure multiplication protocol towards them, respectively. Like the BGW protocol, our system achieves information-theoretic security, and n-party SFE with $\lfloor n/2 \rfloor - 1$-privacy.

Output Data Reconstruction. The last step is to reconstruct the output: The parties can use secret reconstruction schemes to reconstruct the output of the computation from the partial results obtained by each party. In conjunction with the polynomial-based Shamir Secret Sharing scheme, polynomial reconstruction techniques include Lagrange Interpolation, Newton Interpolation, etc. In our protocol, We use Lagrange Interpolation. Compared with Newton's Interpolation, Lagrange Interpolation shows more numerical stability, more flexibility, and yet less space complexity. Lagrange Interpolation is generally more numerically stable than Newton Interpolation, meaning that it is less prone to errors and rounding errors when the data is noisy or the interpolation points are closely spaced. Lagrange Interpolation can be applied to a wider range of data sets than Newton Interpolation, as it does not require the data to be evenly spaced or to have a specific structure. Besides, although they both share the same level of time complexity $O(n^2)$, Lagrange Interpolation only requires the evaluation of a single polynomial function while Newton Interpolation involves the evaluation of multiple polynomial

functions, thus resulting in higher space complexity.

Due to the project development time limit, the MPC algorithm the system adopts works only for the synchronization scenario with passive adversaries strictly less than half of the total participants.

B. Permissioned Blockchain

In order to have a common view of which assets are in the network and their prices, as well as to ensure there is no free lunch for the MPC initiator to obtain MPC results without paying anything at the end, the MPC Peerster nodes run a blockchain. The structure of the Peerster chain's block is a simplified version of the Ethereum block, and each block contains the hash of the previous block, the block miner, the block height, as well as the list of the transactions, and a data map storing the current world states, all are protected by SHA256 hash.

Rather than the more commonly used public blockchains, a permissioned blockchain is built. All messages of transactions and blocks are sent in the public secure channel to stay private from the outside world, and strict checks are applied so that only the chain's participants can create a transaction or become a block miner. The same as Hyperledger Fabric mentioned before in the background section, our Peerster chain stores a unique blockchain configuration that specifies the chain's participant list, the policy to join the chain, the block production policy, etc. The configuration is stored in the world state for transactions to refer to and is protected by a state hash stored in the block header. The details of configuration is listed here: *Participants* is a set that specifies the chain participants. *MaxTxnsPerBlk* specifies the maximal number of transactions in a block. *WaitTimeout* specifies the maximal waiting time for the next transction before producing a new block. If no transaction is received before timeout, a new block will be produced even when the number of transactions inside the block does not reach the threshold. *MPCParticipationGain* specifies the basic working fees for each MPC participants and is used for MPC price calculation. *JoinThreshold* specifies the percentage of endorsement a node needs to gain before it join an exising permissioned chain as a new member.

Instead of separating users from nodes, i.e. users hold the accounts while nodes do not have explicit identities in the blockchain, in our permissioned chain only nodes have accounts and represent parties. This also fits the needs that in MPC each provider party of the data has a single identity. Fig.2 shows how to set up a node to join a blockchain. The first thing is to create an account for the node by creating an ECDSA key pair. After that, it can decide to create its own chain or wait for others to create the chain, and a node is able to create or join a chain only if the chain's config has its address in the participant list. It will then register a public key to the blockchain by sending a $RegEncKey$ transaction so that others can use that public key for encryption and connect with it in the point-to-point secure channel. After the set-up is done, it can start mining and verifying blocks.

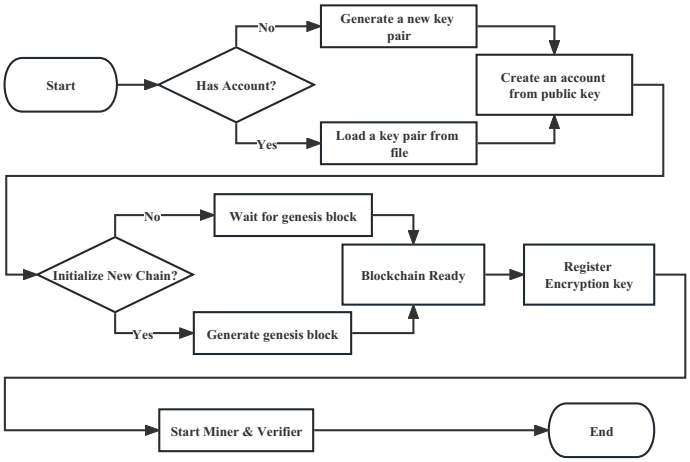

Fig. 2: How to start a chain: Each node will start with creating its blockchain account by either creating a new ECDSA key pair or loading an existing one. It then generates or synchronizes the genesis block and registers its encryption public key. After the genesis block is set, it starts normal mining and verifying

Credit System is used by Peerster chain to decide the miner of the next block every time when a new block is appended to the blockchain. It is inspired by Proof of Stake and uses coin age and account balances to calculate the current credit of the participants. The idea is that the ones with more balances in the chain have more incentive to protect the chain's environment to protect their own benefits. Different from PoS which chooses multiple participants with randomness and requires voting to decide the final block [7], our credit system simply takes the one with the highest credit as the next block miner. Every node would check whether the block miner is the expected one before they examine and append a new block to the chain. In addition, the miner in our system does not earn any rewards for mining a block. Since all these decisions are done locally at each node, there is no time and bandwidth overhead of the consensus, making the blockchain able to run pretty fast.

Txn Pool is an important component to temporarily store unprocessed transactions, and it serves as a first-in-first-out way. A miner will pull the transaction from the pool. If there are no available transactions, it will be blocked until a new transaction is put into the pool, or interrupted by the transaction waiting timeout set in the blockchain configuration. The pool runs a daemon loop to feed the transactions to the miner in time to avoid miner blocks for a long time.

A transaction is created and broadcast by a node in the public secure channel to all the blockchain participants, who directly put the transaction in the pool without verification. Only the current miner will pull transactions from the pool and the next block miner will regard the transaction as invalid if they pull a transaction that is already committed to the chain. The miner will put all transactions back into the pool if a block it is mining on becomes unavailable because of receiving another block. By doing this, we ensure that a transaction will never starve or be processed twice, and it will always be processed by an honest miner in time even when the miners

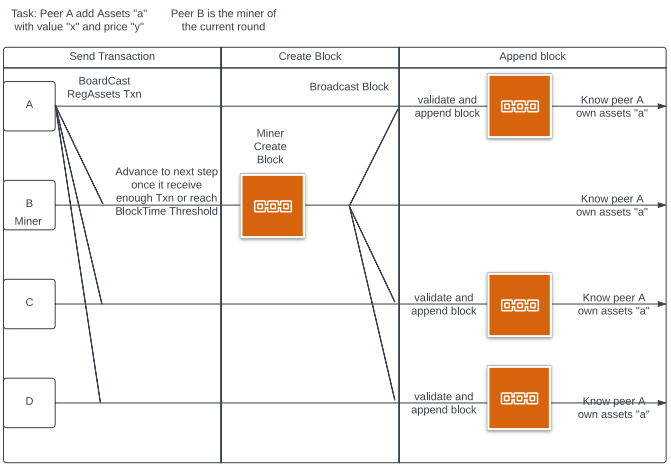

Fig. 3: How Blockchain adds assets. It will first broadcast a $RegAssets$ transaction, and the miner of that round will process the transaction. Once the miner receives enough Txn or waits until timeout, it will create a block and broadcast it to other peers. Other peers will verify the block and append the block. Thus other peers will now consent on a peer own certain asset

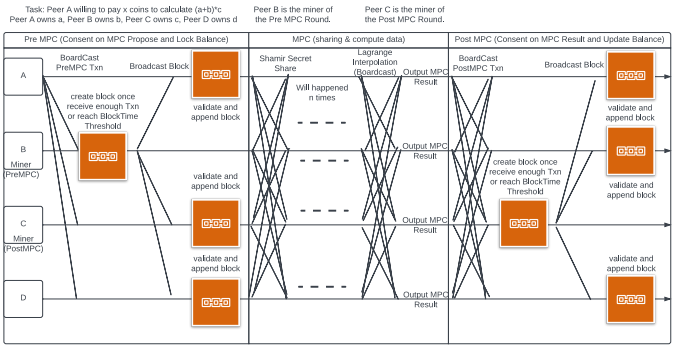

Fig. 4: MPC + Blockchain. A peer initiates the process by sending a $PreMPC$ transaction to establish consensus on the calculation to be performed and the budget for the round. Once consensus is reached, all peers begin the MPC process by sending messages using Shamir secret sharing to share the assets. These messages are encrypted and sent using "ShamirSecretMsg" and "EncryptedMsg" which then encapsulates in "privateMsg". During the MPC process, "InterpolationMsg" that encapsulates in "privateMsg" is broadcast among peers, who use Langrage interpolation to compute the final result. Once all peers have completed the calculation, they broadcast a $PostMPC$ transaction, which is processed by a miner and saved on the blockchain

change.

Transaction Executor is an integrated executor which takes a transaction, finds its dedicated executor based on the transaction's type, and updates the world state accordingly based on the execution result. The execution result of a transaction may require multiple updates in sequence and all updates are done atomically, i.e. they either all succeed, or if any of them fails then all updates shall be reverted. The transaction Executor is stateful and keeps track of the world state so that the following transactions can continue to work on the result of previous transactions. It supports the execution of only specific types of transactions to limit the functionalities of the Peerster blockchain to the scope of MPC and reduce the complexity and increase its security.

C. MPC on Blockchain (Full version)

Before starting MPC, nodes should first register its assets on the blockchain by sending a $RegAssets$ transaction (shown as Fig.3). Instead of the real assets, only the identifiers and the prices of the assets will be included in the transaction. After the assets are registered, other nodes in the network can view which assets are currently available to use in this network and how much they need to pay to start a round of MPC.

As shown in Fig.5 and 4, An initiator starts a round of MPC by sending a $PreMPC$ transaction to the blockchain. This transaction includes all the information an MPC needs, such as the expression to compute, the price to be paid, etc. The price should be at least the sum of the prices of total assets used by the computation plus basic working fees for each MPC participant. After the transaction is included in a block, the same amount of coins in the initiator account will be locked. The locked coins are no longer counted into the user's balance and the user is not able to spend them.

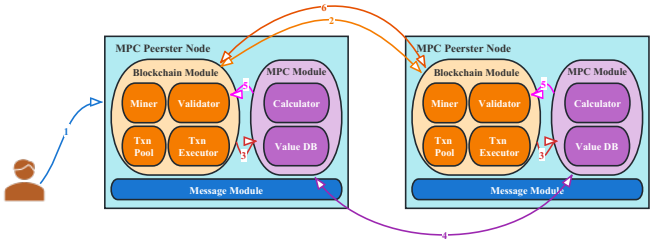

Fig. 5: Blockchain+MPC Interaction Flow: 1. User sends an MPC request to node; 2. The node send a new PreMPC transaction to the chain; 3. The blockchain module notifies MPC module to start a new MPC once it finds out a PreMPC transaction is committed in the chain; 4. MPC starts; 5. MPC module notifies Blockchain module to send a PostMPC transaction after it finishes the current round of MPC

Once a node finds out a PreMPC transaction is included in a block, it will automatically start MPC using the information included in that $PreMPC$ transaction. Since all nodes get the block, they will all start the same MPCs. After a round of MPC is finished, each node will automatically send a $PostMPC$ transaction which contains the corresponding $PreMPC$ transaction ID and the hash of the computed result. The result is hashed to avoid free lunch from the outside world, and the locked coins will only be released to the participants after more than half of the nodes' $PostMPC$ transaction for the same $PreMPC$ transaction is committed to the blockchain to avoid free lunch from the internal world. To be consistent with the privacy of permissioned blockchain, MPC uses both the p2p secure channel and the group secure channel to ensure MPC messages are only visible to part or all blockchain members and hidden from the outside world.

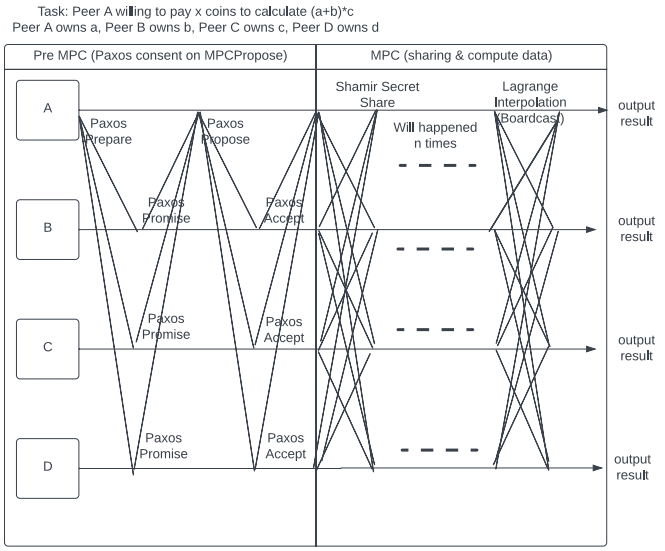

Fig. 6: Paxos+MPC Interaction Flow: 1. User sends an MPC request to node; 2. The node will start a Paxos process. 3. MPC process start once the Paxos accept threshold is reached. 4. output the result.

We make use of the permissioned blockchain's participant list and require all participants of the blockchain to be also the participants of MPC so that no other complicated consensus is needed for deciding MPC participant list. Because the identities of chain participants are carefully examined when they join the chain, the possibility of active adversaries as well as the number of passive adversaries going above half of MPC participants is lower and privacy during MPC is protected.

Because of the permissioned blockchain's simple consensus, if the maximal transaction waiting timeout in the chain's config is set correctly, MPC should start in a short time after submitting the PreMPC transaction to the blockchain. The related transactions will never be reverted to avoid free lunch. Apart from these, permissioned blockchain also uses the group secure channel to hide its messages from the outside world by its nature, which further minimizes the chances of secret leakage.

D. MPC on Paxos (Quick Start)

Setting up a blockchain benefits the whole MPC Peerster community in the long-term aspects. However, if parties only want to set up a temporary network to run some computations, or if they know well about which assets are in the network and would like to do pricing offline, then setting up a blockchain might be too costly for them. In this case, we also provide a quick setup of the network with the help of multi-Paxos. As shown in Fig.6, multi paxos is run to consent on the expression to calculate. Once consent is reached for that expression, the MPC will start parallelly in another thread and we can start a new paxos round to consent on new expressions to calculate.

IV. Implementation

In this section, the paper presents the implementation details of MPC Peerster, including how the code is structured and the details of how important functionalities are built. It starts with an overview of the codebase. After that, it will explain the details of the secure channel where the security of MPC and permissioned blockchain rely, followed by MPC and permissioned blockchain themselves. Finally, it will briefly introduce the CLI tool and the web UI.

The implementation is written in Go and the code is available at https://github.com/lilyyangyn/peerster.

A. Codebase overview

The code base has two main directories: `peer` defines all the interfaces an MPC Peerster node provides and the node's implementation. The implementation is divided into modules, including messaging module, consensus module, blockchain module, MPC module, etc. `permissioned-chain` is an important library that defines all blockchain-related structures, how different types of transactions are executed, and how new blocks are verified and appended to the blockchain.

Other supportive libraries include `transport` that implements how nodes' packets are sent over UDP or go channels, `types` that defines the message types the nodes use to communicate, `registry` that is a registry for handlers of different message types, `storage` that offers different types of storage for nodes and the blockchain to use, as well as `cmd` and `httpserver` which offer the command line tools and a local web UI to facilitate user interactions with the nodes.

B. Secure channel

A secure channel is implemented to support MPC secret-sharing schema and permissioned Blockchain communication. Two types of secure channels are implemented: one is point-to-point communication and is strictly secure, and the other is suitable for broadcasting a message to a group and is only weakly secure.

Point-to-point secure channel encrypts messages and sends the cipher bytes in a private message. Each MPC Peerster node will generate an RSA key pair locally at the node start, and broadcast its RSA public key in the heartbeat message so that other nodes can obtain the keys. The plain messages are encrypted by the receiver's public key so that the receiver is able to decrypt them with its private key. The cipher bytes are put into a special message type called private message, which is explained in Group secure channel. The point-to-point channel is secure even under all circumstances assuming the cryptographic primitives are unbreakable.

Group secure channel does not encrypt a message and only put it in a private message that specifies which nodes are its receivers. There are two types of private messages. One uses nodes' IP addresses to specify the receivers, and the other uses nodes' account addresses on the blockchain to specify the receivers. The group secure channel may leak privacy if the on-path nodes are dishonest, and the MPC secrets should always use the p2p channel to send.

C. Multi-Party Computation

As mentioned in the design section, MPC protocols usually involve four major steps: function identification, input division,

computation execution, and output reconstruction. Our MPC protocol parses the expression locally to decide which functions to perform and the execution sequence. It makes use of Shamir Secret Sharing and computationally efficient Lagrange Interpolation as building blocks for input division and output reconstruction, respectively, to ensure privacy security. By doing these, parties are able to jointly compute a function over their data without revealing the data to each other or to any external observers.

Function Identification Function identification is the process of converting an input expression into a suitable data structure for calculations in MPC. The input expression is first transformed from infix notation to postfix notation, which allows for step-by-step calculation and determination of necessary assets. If a variable or asset is determined to be owned by others, the peer will wait for shares of the asset to arrive before continuing with the calculation. Once the local calculation is finished, all the peers will broadcast an interpolation message to share the local result. When enough interpolation messages are received from other peers, the final answer can be reconstructed using Lagrange interpolation.

Shamir Secret Sharing (SSS) is the second step in MPC after Function Identification, and it is the first MPC action that requires network interactions. It is used to distribute the secrets among MPC participants by dividing a secret into a number of shares, each of which is held by a different party. The secret can only be reconstructed by combining a sufficient number of the shares in the process called Lagrange Interpolation explained in the later section. By using SSS, our system ensures that each participating node can only learn part of the secret and the result can still be reconstructed at the end of the computation.

An important technical detail is that the Shamir Secret Sharing scheme must be performed in a finite field. There are several reasons why regular integer arithmetic or ring is not ideal. First, in Lagrange Interpolation, the calculation used in the reconstruction needs to divide one "number" by another, and division is not defined in $\mathbb{Z}$, the set of integers, leading to rounding errors or overflow, undesired floating number result or compulsory type conversion. Second, Integer arithmetic will narrow down the possible values for adversaries to conduct exhaustive search attacks, thus largely damaging the security. Third, also a more subtle issue is that the SSS implicitly assumes that a polynomial of degree n does not have more than n roots, which is not true in rings. For example, the polynomial $x^2 - 1$ has four roots $1, -1, 4, -4$ in the ring $\mathbb{Z}_{15}$ instead of the $1, -1$ that it has in a field such as $\mathbb{Z}_{17}$. These problems will be naturally solved using finite field arithmetic.

To implement SSS, we first implemented a polynomial in a finite field by applying all operations with a modulus. The leading node should first construct d randomly chosen number $r_1, ..., r_d$ and set the polynomial to $f(x) = s + r_1x_1 + ... + r_dx_d$. A share $s_i = f(\alpha_i)$ is computed and sent under the secure channel to each participant, where α is the hash of the participant's public key

Computation execution decomposes the calculation function into two basic units: addition and multiplication.

The addition computation is relatively straightforward. This is because one special property of the Shamir Secret Sharing scheme is linearity, i.e., the reconstruction of the secret from the shares is a linear mapping. In practice, this means that any linear operations performed on the individual shares can be translated to operations performed on the secret upon reconstruction. This form of additive homomorphism makes linear secret-sharing schemes so useful in applications such as secure computation. And this property makes linear operations able to be performed indirectly on the secret, while actually being performed locally on the shares by each individual party, without requiring the exchange of additional information or other forms of communication among the parties.

While the addition computation of two secrets can be directly derived from the addictive homomorphism of the SSS scheme, the evaluation of multiplication is far more complicated. To obtain the product of two secrets, each node needs to multiply its own shares of the two secrets, but this leads to the underlying multiplication of two polynomials which will inevitably double the degree. As a result, we need the secure degree reduction of polynomials to provide service for output reconstruction, and potential further computation such as executing two consecutive multiplications $a * b * c$. Secure degree reduction is performed by an additional round of Shamir Secret Sharing and Lagrange Interpolation.

Addition is implemented in two steps. First, $a, b, ...$ are shared by $a_1, ..., a_n; b_1, .., b_n$. Second, $\forall P_i$: compute $c_i = f(a_i, b_i, ...)$. In contrast, multiplication needs to be done in four steps. First $a, b, ...$ are shared by $a_1, ..., a_n; b_1, .., b_n$. Second, $\forall P_i$: compute $d_i = a_i * b_i$. Third, $\forall P_i$: share $d_i -> d_{i_1}, ..., d_{i_n}$. Fourth, for all P_i: compute $c_j = L(d_{1_j}, ..., d_{n_j})$.

Lagrange Interpolation is used to reconstruct the data from given shares and perform the desired computation. It is a mathematical technique for constructing a polynomial function that passes through a set of given points.

To reconstruct the output, we will need to use Lagrange interpolation. Reconstructing output can be done in three steps. First, a is shared by $a_1, ..., a_n$. Second, $\forall P_i$: compute a_j to P_i. Thrid, $\forall P_i$: compute $a = L(a_n, ...a_n)$.

D. Permissioned Blockchain

The block miner, block validator as well as transaction executor constitute a full node in Peerster permissioned blockchain. The miner is responsible for producing new blocks using the received transactions, the validator verifies whether the received new block is valid to be appended to the local blockchain, and the transaction executor controls all things related to the transaction executions and is used by both the miner and the validator.

Peerster permissioned blockchain follows Ethereum's definition to construct its blocks and transactions, i.e. a block should have fields such as previous block hash, miner, and height, as well as storage of the current state machine's state and a list of signed transactions together with their hashes, and a transaction should have a sender, a recipient, a value of coins

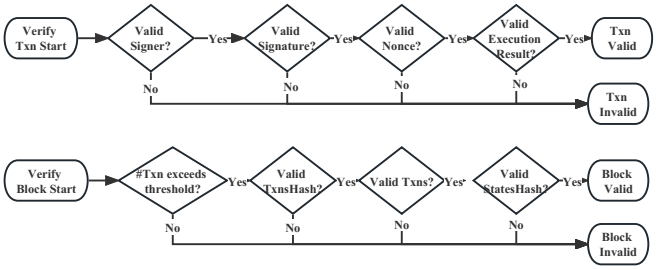

Fig. 7: Verify Txns and Blocks: The transaction's sender, signature as well as nonce are validated before it is executed. It succeeds if it passes all the checks and has a valid result given the current world state. The same as transaction validity, we need to check the blocks' transaction number as well as the hash of the transaction before we executed the transactions in it. The hash of the world state after all transaction executions are computed and checked with the provided state hash to ensure the same execution result.

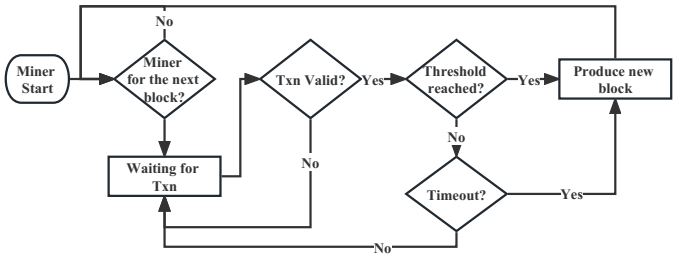

Fig. 8: Miner: it will block until it becomes the miner of the next block. Then it starts pulling transactions from the transaction pool and validates the transaction. The process ends when the number of transactions in the block reaches the maximum or if the waiting timeout is reached.

attached, a nonce from the sender's account to avoid replay attack, and a data field to attach data. A signed transaction is a transaction with a signature created by the transaction sender. Fig.7 shows the details of how a block and a transaction are verified. The validation of blocks and transactions follows how Ethereum does, except that we additionally verify that the block miner and the transaction sender should be a participant in the permissioned chain.

1) Block Miner:

Miner is a daemon process that starts after the blockchain is set up. It will block until the node becomes the next miner and will block again after producing a block. Fig.8 shows the details of how a miner works. To avoid empty blocks containing no transactions being produced, while also ensuring that new blocks can be produced in a short waiting time even when there are only a few amount transactions, two limits are proposed to the miner: the maximal number of transactions per block, and the maximal waiting time for the next transaction before producing a non-empty block, both configurable in the chain's configuration. Once the miner gathers enough amount of transactions, or once the waiting time is reached, a new block will be produced and broadcast to all nodes inside the permissioned blockchain.

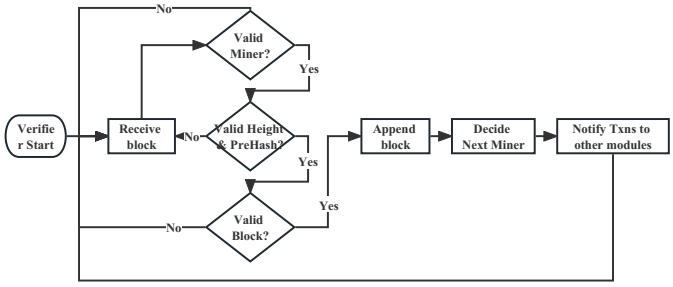

Fig. 9: Validator: it listens for the received blocks and verifies if the block is mined by the expected miner and whether it is at the correct height and includes the previous block's hash in its header. The block is then validated for its contents' correctness and appended to the chain as finalized version. A new miner will then be decided and the verifier will notify through Txn Notification Center other modules of the node, which is waiting for a certain type of transaction to be shown in the blockchain.

2) Block Validator:

Validator is also a daemon process that starts after the blockchain is set up. It receives blocks from the miner, validates the blocks, and appends them to the blockchain. Fig.9 shows how a validator works. After a block is appended to the blockchain, the validator will also call the credit system to select the next miner and notifies the mining daemon if the node itself is decided to be the next miner.

Txn Notification Center is called by the validator once a block is successfully appended to the blockchain. It helps to notify the local node that a new transaction is committed to the blockchain and the node is then able to perform actions related to that transaction. A node should register the listener to the notification center to be notified, and the listener is registered by transaction type.

3) Transaction Executor:

A transaction executor takes a transaction, find the dedicated executor according to the transaction type and call that executor. It keeps track of the world state so that a sequence of transactions can be executed continuously, each based on the execution result of the previous one. The modification of a failed transaction will be reverted to ensure that all transactions are executed based on the correct world state.

Since the permissioned chain is a blockchain with dedicated use cases, only a fixed group of transaction types are supported. They are listed below.

InitConfig: A transaction initializes the blockchain configuration and adds it to the blockchain state map. It is created by the Zero address and is only inside the genesis block as the very first transaction. Any *InitConfig* transactions received by a miner will be considered invalid.

Coinbase: A transaction sets up the initial balance of the blockchain accounts. It is created by Zero address and is only inside the genesis block behind the *InitConfig* transaction. Any *Coinbase* transactions received by a miner will be considered invalid.

RegEnckey: A transaction registers the encryption public key

so that the corresponding node is able to be communicated using the p2p secure channel. It is sent by a node when it joins the permissioned chain and once an encryption key is set for a blockchain account, it cannot be set twice using the *RegEnckey* transaction.

RegAssets: A transaction registers or updates the node assets identifier together with its prices so that other nodes can pay the owner to use the assets in an MPC. The same type of transaction can be sent multiple times for a sender to register different assets or update the price of existing assets.

PreMPC: A transaction is expected to be sent before an MPC starts. It checks whether the value attached to this transaction is enough to pay the prices of assets used in the following round of MPC and the basic working fees for MPC participants. It then locks the coins from the sender's account so that they will no longer show in the sender's balance and is not able to be spent or locked again by the sender. The execution of the *PreMPC* transaction does not differentiate the initiator from normal workers, so even if the initiator uses all himself's assets in the following MPC, the assets prices and the working fee of himself are still locked as deposit and can be claimed back after the round of MPC finishes. Apart from locking the coins, a special structure will also be created and added to the world state so that the blockchain can easily check how many participants have finished this round of MPC and when to release the locked money to the participants. The transaction will fail if the executor finds the attached value is not enough to pay the MPC, or if there is not enough amount of coins in the sender's account to be locked.

PostMPC: A transaction is expected to be sent by every MPC participant after the MPC finishes. The executor will update the record in the world state, which is created by the *PostMPC* transaction, to add the sender of this *PostMPC* transaction as an endorser of this record. Once the number of endorsers exceeds the half of total MPC participants, the locked coins will be released to all existing endorsers. After that, MPC participants who are yet to send the *PostMPC* transaction are able to claim their rewards directly when they send their endorsement. The record will be deleted from the blockchain's states once all participants send their endorsement to save the chain's storage. The transaction will fail if no record of the corresponding *PreMPC* transaction is found, or if the sender is not recorded as a participant in this round of MPC.

E. Multi-Party Computation on Blockchain

The implementation of integrating MPC with blockchain just follows the description in the design section, except that every node should register a listener to the blockchain validator so that they can be notified once a new *PreMPC* transaction is committed. The round of MPC for that *PreMPC* starts immediately after that commitment.

F. UI

Both a CLI tool and a Web UI are provided for users to choose from.

Fig. 10: UI main page, showing the concept of our system.

CLI tool provides an interactive list so that the user can scroll the list or filter the list by typing to find the desired options. It starts with setting up the node's blockchain address by loading or creating ECDSA key pairs. If users want to set up the chain by themselves, they should then provide a YAML file to the CLI to read the blockchain configuration. Otherwise, the user should ask the node who sets up the blockchain to include his account in the chain configuration and add his node IP address to its routing table, and their local CLI will wait until the node gets a genesis block.

After the blockchain is set up, the CLI will provide all available actions for the user to select, including adding or showing assets, showing account balance, showing blockchain data, adding a new peer to the routing table, etc. Regarding blockchain data presentation, blocks are lazily loaded from the latest block to the genesis block. Only a batch of the limited number of blocks is loaded and the user needs to click "More" to load the next batch of blocks. The user can select each block to enter another page which shows the details of the block together with its transactions. The option of MPC Calculation will only be available if all blockchain participants have registered their encryption public key to the blockchain.

Web UI is a more user-friendly place to interact with. However, it hides all details about the blockchain from the user. It relies on the CLI tool to set up a blockchain, and blockchain data can only be viewed in the CLI tool. As shown in Fig.10 the UI is implemented in Wix and is available at https://chenlew.wixsite.com/mpcpeerster. The UI has two main functions, add asset and MPC calculation. As shown in Fig.11 the UI allows the user to set assets to DB, and show the current assets owned by other peers. On the other hand, As shown in Fig.12, the UI allow the user to perform MPC calculation and show the history of MPC result query by the peer.

V. Evaluation

In this chapter, the paper discusses in detail how we set up the experimental environment, how the system is tested for correctness, as well as the performance and security analysis on MPC Peerster.

A. Environment Setup

The experiment is conducted on MAC OS, MacBook Pro with 2.6 GHz 6-Core Intel Core i7.

Fig. 11: Page for adding new assets to the database. The user interface is straightforward and will display the current block address, as well as the assets that are owned by others.

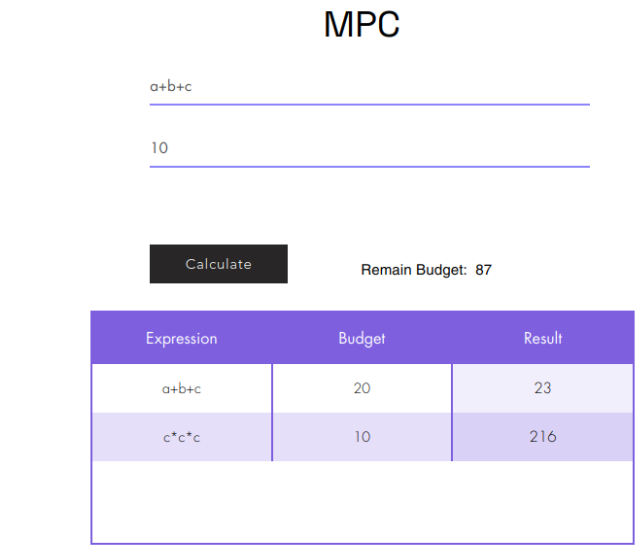

Fig. 12: Page for performing MPC calculation. The user interface is straightforward and will display the history query and the result of the current user.

We use the latest version of the code in github. More precisely, `lilyyangyn/peerster commit 8ab11442e9fc55bf37e153d924d9dd7c3a59163b`. For testing purposes, we run all test all the nodes using the way we run the integration test.

B. Correctness

To test the correctness of the individual function, module, and system, we wrote unit tests, module tests, and system integration tests.

Unit tests were done on each part of the systems on individual functions. Regarding MPC, computational correctness is the top priority of all tests. First, we tested finite field arithmetic, including basic calculation units (addition, subtraction, multiplication, and division), and random generators of big primes and polynomials. We also conducted tests about Shamir Secret Sharing and Lagrange Interpolation under different circumstances from simple to complex ones. The tests mainly focus on the correctness and consistency of changing node numbers and polynomial degrees. Regarding computation execution, we tested the addictive homomorphism of SSS and the correctness of a multiplicative calculation. Furthermore, we derived and tested continuous computation of MPC, which is the core to fulfill real-world purposes. In the blockchain part, we wrote tests on the transaction, block, and blockchain levels, including the verification and execution of each transaction type, the creation and validation of blocks, and blockchain initialization, validation as well as appendment. We also wrote tests about miner and validator daemons with different test cases to ensure they work correctly.

Module tests focused on the MPC module and the Blockchain module and required us to run the full node to test. In order to test the MPC module, we begin by conducting unit tests for Shamir Secret Sharing and Interpolation to confirm that the shares can be accurately reconstructed. We then proceed to add unit tests for single addition and multiplication to verify the accuracy of the values. Next, we conduct integration tests by providing assets to the test node and running a range of expressions, from basic ones like (a+b) to more complex ones (a*b+c*d-d*b, etc.), and evaluate the results. Lastly, we perform stress tests to determine the capacity limits of our MPC module. In order to test the Blockchain module, we conducted a variety of tests to ensure its proper functioning. This included sending different types of transactions in different sequences and under varying levels of transaction density. Additionally, we made adjustments to the maximum transaction waiting time and the maximum number of transactions per block in the blockchain configuration to evaluate its correctness.

Integration tests were done at last to test the correctness of the full MPC process. The full process contains both blockchain interactions and a pure round of MPC that was automatically launched by a blockchain commitment and should automatically launch blockchain transaction processing when it is finished. We ensured that the full MPC would not be stuck and was able to work smoothly under high parallelism.

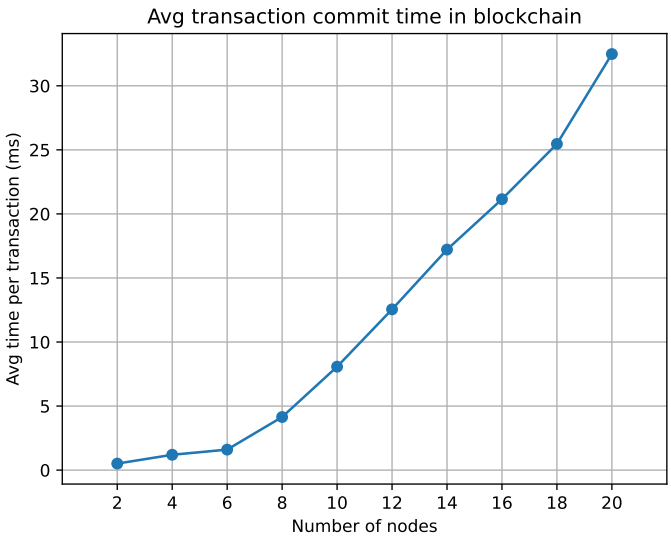

Fig. 13: Runtime chart for a transaction to be committed with different number of nodes

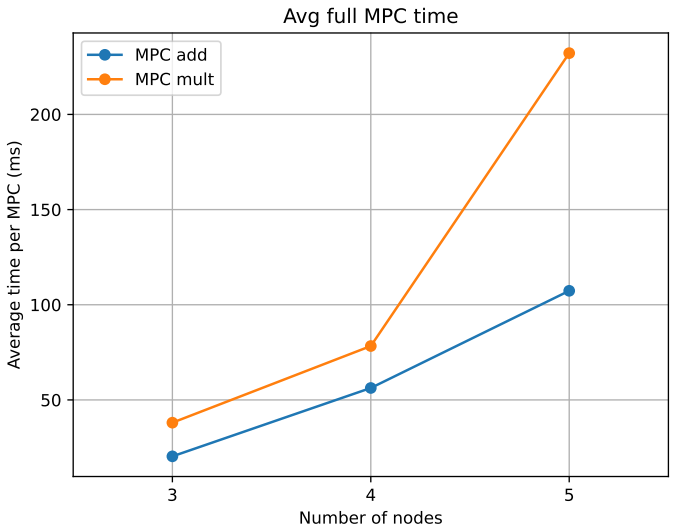

Fig. 14: Runtime chart for full MPC run with different number of nodes with add and multiply

C. Performance

We designed and conducted multiple performance tests with a focus on system throughput in stress tests, and execution time distribution to analyze the bottleneck. The first performance test we conducted was to evaluate the time required for committing a transaction. As illustrated in Fig. 13, the execution time increases in a polynomial manner, which aligns with the theoretical expectation since the number of transactions increases as n square. The second test was to measure the average execution time for one MPC calculation. As shown in Fig.14, the overall performance is quite good, taking only milliseconds. However, as the number of nodes increases, the average time also increases. This is due to the complexity of multiplication being n square, resulting in the average time of multiplication increasing at a faster rate than addition. The third test was to measure the time distribution of different building blocks. As shown in Fig.15, Fig.16 and Fig.17, MPC calculation is the stage that takes the most time among all stages, and more multiplications lead to more time taken for MPC calculation.

D. Security

The security of MPC Peerster system highly relies on the security of MPC and the security of Blockchain. In this section, we will start by analyzing the security of these two modules and conclude by reviewing the security of the whole system.

1) MPC Security:

In our model, we assume there is only the presence of passive adversaries who do not actively disrupt the protocol but may try to learn as much as possible about the inputs and outputs of the parties. And there is one passive adversary that controls all the malicious nodes and is able to learn information from all malicious nodes. This is a common threat model in the design of cryptographic protocols, as it reflects the behavior of many real-world adversaries who may try to passively gather information without being detected. And the security analysis of the MPC protocol also assumes all communications are transmitted in bilateral secure channels. So if two players exchange data, a third player has no information at all about what is sent. Such channels might be available because of physical circumstances, or we can implement them relative to a computational assumption using cryptography.

As explained in previous sections, the MPC protocol uses Shamir Secret Sharing and Lagrange Interpolation to achieve information-theoretic level security. These methods are based on polynomials. Assume at the beginning of the MPC protocol, every secret is shared using a polynomial of degree d. As basic math indicates, the reconstruction of a polynomial of degree n needs at least $n+1$ (different) points. To prevent the adversary from reconstructing the polynomial from the shares she owns, the number of shares that are possibly known by the adversary should be less than $d+1$. This means if the adversary controls t malicious nodes, $t < d+1$ should always hold, leading to the result $d \geq t$, for d and t are all integers.

Remember in the secure multiplication protocol, every node needs to multiply its own two shares to obtain a product share. This is essentially the multiplication of two polynomials of degree d, resulting in a product polynomial of degree $2d$. In order to reconstruct the product shared by a polynomial of degree $2d$, we need at least $2d+1$ points from different nodes. Considering the fact that there are n nodes in total, the points we need to reconstruct the product should be no more than the total number of nodes, that is $n \geq 2d+1$.

Combining the two inequalities above, we derive the conclusion $t < n/2$ where n is the total number of nodes and t is the number of malicious nodes controlled by passive adversaries. This bound is optimal for perfect security and privacy in this

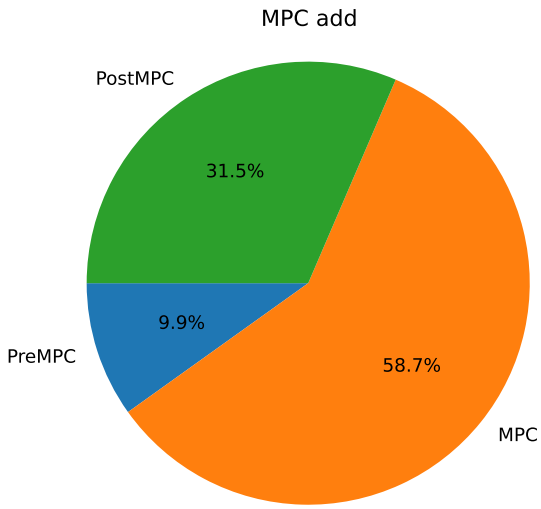

Fig. 15: Timeline of full mpc process for expression = a+b+c, takes 42ms

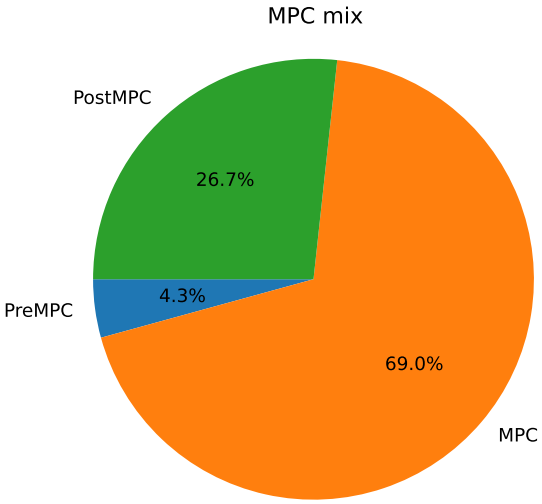

*Fig. 16: Timeline of full mpc process for expression = (a+b)*c, takes 60ms*

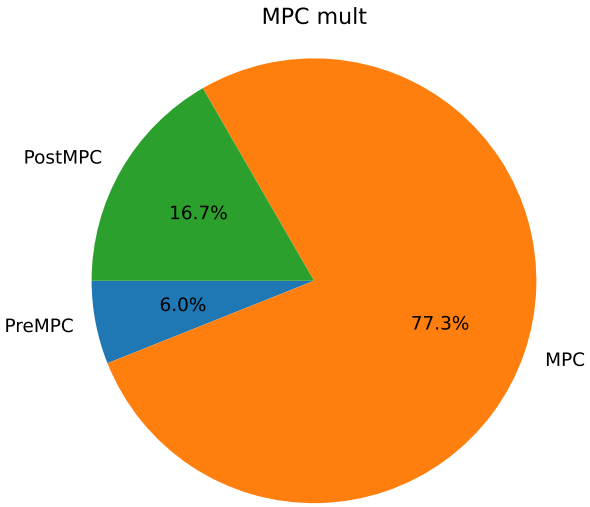

*Fig. 17: Timeline of full mpc process for expression = a*b*c, takes 107ms*

setting. And the degree d of the polynomials used to share the original secrets should satisfy two inequalities: $d \geq t$ and $n \geq 2d+1$. As you can see, the doubled degree in the secure multiplication protocol is the reason why MPC protocols using SSS require an adversary number strictly less than half of the total nodes, and why we only consider any number $n \geq 3$ of players, because the polynomial of degree zero would be constant thus meaningless in terms of privacy.

2) Blockchain Security:

Just like other blockchains, our blockchain ensures that the final version of the blockchain will not accept faulty transactions and blocks. This is done by requiring each node to run a validator daemon to validate the block before appending it to the local chain, and the honest node will only accept and work on valid blocks.

Each transaction should be authenticated by sending it together with its sender's signature. In this case, no fake transaction can be created by an attacker without obtaining the sender's private key. To avoid a replay attack, each account is associated with a nonce starting from zero. The nonce will be included in the transaction to be signed, and the sender's account will be extracted from the chain's world state and used to compare with the transaction nonce to pass the validation. The account's nonce will be increased for each transaction being committed to the blockchain so that the same transaction cannot pass the validation twice.

A permissioned blockchain is used so that nodes can only join a network with the network members' proof. The participants' identities are therefore usually validated and all their actions inside the network can be traceable and accountable to a real party. In addition, since both the permissioned network and MPC wrapped their messages in a public secure channel, the privacy of the data on the chain and in MPC is also protected.

A dedicated transaction executor is used so that only limited types of transactions with pre-defined behaviors can be executed and committed to the chain. This significantly minimizes the abilities of attackers by forbidding them from tricking other nodes to execute the attacker-defined malicious code

The weakness appears in the consensus that the permissioned chain uses to select the next block miner. Since nodes only accept blocks mined by the valid miner, if the selected miner goes offline or if a malicious miner does not produce a new block or produces a faulty block, the whole blockchain will stop being extended. However, it is still acceptable as the MPC algorithm the permissioned blockchain assists is also able to proceed only when all nodes are online. Regarding the miner node being malicious, it can only cause a DoS attack by not producing the correct block but not letting others accept a faulty block. Since the node's identity is connected to a real party, the DoS attack can also be solved offline. In this case, there is no incentive for them to launch the attack.

3) Transaction Execution Security: Before concluding the system security, we would first analyze the transaction executions where weakness might appear. There are four types of transactions that could be mined by a miner: *RegEnckey*,

RegAssets, *PreMPC* and *PostMPC*, which register nodes public key to be used in p2p secure channel, register assets' identifier with prices, lock the amount to be paid in PreMPC from the initiator's account, release rewards to MPC works, respectively. Since an active adversary is not able to let honest nodes accept faulty blocks or produce fake transactions, what he can do is switch or ignore transactions.

Switching transactions will not cause a problem because there is no race between the same type of transactions, and different types of transactions have strict execution dependencies and switching their sequence simply cause the transaction execution to fail and therefore a faulty block.

Ignoring transactions does not cause a problem as well. Ignoring *RegEnckey*, *RegAssets*, *PreMPC* simply causes the result that the MPC will not start, and no one will lose anything. Ignoring *PostMPC* might cause the MPC rewards never be unlocked, however, in this case, the initiator will also be unable to get his deposit back, which results in everyone, including the miner losing the money. Therefore, a logic miner will have no incentive to do that.

Another attack that might be launched to *PostMPC* is that the attackers do not work in MPC but directly submit a *PostMPC* to the blockchain. What they are trying to do is to unlock the reward without actually working for MPC, and it is hard to check whether the result is fake without knowing the real secret value. However, this attack has no chance to launch in the reality as long as the total number of attackers is strictly less than half of the total number of participants. The logic behind this is that MPC will not return a result if any of the MPC participants are the free riders, and therefore an honest node will never send their *PostMPC* to the blockchain to unlock the reward. If the number of attacks is less than half of the total number of participants, the threshold of unlocking the rewards will never be reached.

4) System Security:

In conclusion, MPC Peerster is able to protect data privacy under the scenario that the number of passive adversaries is strictly less than half of the total participants.

The system is built on a permissioned blockchain to calculate payment so that participants are incentive to be involved in MPC and provide their secrets to be used in MPC. The permissioned chain keeps its actions and data secret from outside, which further increases the data privacy in MPC. The blockchain always works correctly, i.e. no faulty transaction will be processed and no faulty block will be accepted, even for active adversaries as long as the total number of them is strictly less than half of the blockchain's participants. In addition, all the commitments and attacks in the blockchain are accountable to real parties because of the authentication requirement of the blockchain.

MPC Peerster system can proceed without being stuck if all participants are online or join lately. Though the requirement is hard to reach in a wide decentralized network, it does not cause many difficulties in our system. This is because the use cases of MPC and the permissioned blockchain usually require the identities of participants to be checked and contain only a limited number of parties who are able to communicate offline.

VI. Conclusion

Our project focuses on the development of a decentralized multiparty computation (MPC) system in Peerster. MPC allows multiple parties to jointly compute a function without revealing their input data to each other. By decentralizing the system, we aim to increase security and privacy while also allowing for greater scalability and fault tolerance. We have implemented several key features such as Shamir Secret Sharing, interpolation, and communication protocols to enable secure computation among multiple parties. Additionally, we have also incorporated a permissioned blockchain to manage the access and authorization of participants in the system, as well as create incentives for nodes to participate in MPC. We have performed several unit and integration tests, as well as performance tests to evaluate the efficiency and security of our system. Overall, our project aims to provide a secure and efficient solution for decentralized multiparty computation.

References

[1] Hyperledger-Fabric. Hyperledger fabric: A blockchain platform for the enterprise. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/index.html

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Tech. Rep., 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[3] V. Buterin, "Ethereum whitepaper," Tech. Rep., 2014. [Online]. Available: https://ethereum.org/en/whitepaper/

[4] IBM-Research. What is hyperledger fabric? [Online]. Available: https://www.ibm.com/topics/hyperledger

[5] Y. Lindell and B. Pinkas. Multiparty computation: An introduction. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3387108

[6] U. Maurer. Secure multi-party computation made simple. [Online]. Available: https://crypto.ethz.ch/publications/files/Maurer02b.pdf

[7] Ethereum. Proof-of-stake (pos). [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/