Initial Public Offering (IPO) on Permissioned Blockchain using Secure Multiparty Computation

Fabrice Benhamouda
Algorand Foundation
New York, NY, US
fabrice.benhamouda@normalesup.org

Angelo De Caro
IBM Research
Zurich, Switzerland
adc@zurich.ibm.com

Shai Halevi
Algorand Foundation
New York, NY, US
shaih@alum.mit.edu

Tzipora Halevi
Brooklyn College
Brooklyn, NY, US
thalevi@nyu.edu

Charanjit Jutla
IBM Research
Yorktown Heights, NY, US
csjutla@us.ibm.com

Yacov Manevich
IBM Research
Haifa, Israel
yacovm@il.ibm.com

Qi Zhang
IBM Research
Yorktown Heights, NY, US
q.zhang@ibm.com

***Abstract*—In this work, we add secure multiparty computation capabilities to the permissioned blockchain architecture of Hyperledger Fabric, and use them to implement the clearing price mechanism for initial public offering (IPO). As with any blockchain, the core property in Fabric is that all peers must see the same ledger, so using *confidential data* on the ledger is a challenge. To address this challenge we use cryptographic secure multiparty computation (MPC), which requires that we integrate a few new mechanisms into the Fabric architecture. Specifically we need to let the peers access local information such as their respective secret keys, and also send messages to each other while executing smart contracts. We also had to add to Fabric a library that implements the required cryptographic tools, and to make that library accessible from the smart contracts.**

We demonstrated the effectiveness of this solution by using it to implement the clearing price mechanism for IPOs. We designed an efficient cryptographic protocol for the IPO clearing price mechanism, and used our integrated system to run it on Fabric. Although not fully optimized yet, the performance of the resulting implementation is more than fast enough for this particular application, ranging from 8 to 23 seconds to execute an IPO sale.

***Index Terms*—blockchain, confidentiality, Hyperledger Fabric, integration, ipo, secure multiparty computation**

I. Introduction

Blockchain technology saw explosive growth over the last few years. Far beyond just digital currencies, blockchain is often touted as a revolution that is "Changing Money, Business, and the World" [1]. At its core, a blockchain is a distributed architecture for "agreeing on things": It provides an immutable append-only shared ledger, allowing multiple entities to access the ledger and ensuring that they all agree on its content, without having to rely on any single trusted party. The "things" on the ledger are records of transactions, and applications that use blockchain must come with some logic to determine what constitutes a transaction, and what records are stored on the ledger. Blockchains come in two flavors, public (premissionless) or private (permissioned). In the current work we focus on the latter, where direct access to the ledger is controlled and is often limited to a small set of parties.

The core principle that everyone must see the same ledger, makes it challenging to handle transactions that depend on confidential data. But there are many use-cases that can benefit from using confidential data on the ledger. Consider for example a consortium of hospitals that set up a blockchain to record patient treatment, for purposes of audit and compliance. The data itself must be kept confidential to a single hospital, but there is great value in computing aggregate statistics across multiple hospitals to gauge the effectiveness of different treatment options. (More examples are described e.g., in [2], [3], [4]).

A natural solution for storing confidential data on a blockchain is to write it on the ledger in encrypted form. This way, everyone sees the same ledger, but only the secret-key owner has access to the cleartext data. This solution, however, still leaves the question of how to use that confidential data in transactions. In the example from above, imagine that the hospitals want to jointly devise a model that predicts the effect of a certain treatment protocol. How can they achieve this without trusting a single entity with all the cleartext data needed to train that model?

An exciting possibility is to use for this purpose the cryptographic technology of secure *multiparty computation* (MPC). This technology, invented more than three decades ago [5], [6], allows a set of mutually suspicious parties to compute any aggregate function on their respective confidential inputs without revealing their secrets, just by exchanging cryptographic messages back and forth. Research into efficient secure-MPC protocols has advanced in leaps and bounds in recent years, making them suitable for many real-world applications. The possibility of using secure-MPC for confidential data on a blockchain was noted before (e.g., [2], [3], [4]), but it only now begins to enter actual blockchain systems. In this work we use this solution in *Hyperledger Fabric*, which is perhaps the most widely used permissioned blockchain.

Hyperledger Fabric [7] (or just Fabric for short) is an enterprise level permissioned blockchain platform, written in Go. The nodes with direct access to the ledger in Fabric are called *peers*, and each peer belongs to some organization. Transactions are originated by *clients*, who contact the peers to submit a *transaction proposal*. The peers then execute a smart contract — called a *chaincode* in Fabric — to determine whether or not the transaction should be accepted, and what should be recorded in the ledger. If the chaincode execution is successful, the peer *endorses* the transaction by signing it, and only transactions that were endorsed by sufficiently many peers will be recorded in the ledger. Since the endorsement process is when the application logic is executed, we run secure-MPC protocols during endorsement.[1]

To demonstrate the power of confidential information on Fabric, we used it to implement a system for IPO trading. This is an example of a *clearing price auction*, where a single seller sells multiple shares at the same price to many buyers. The system that we implemented supports multiple banks and brokers (e.g., brokerage firms) and many investors. They are all treated as *clients* of the blockchain, and the blockchain itself is maintained by a small number of organizations, each with its own peer(s).[2]

The life cycle of an IPO begins when a bank lists it publicly on the ledger, specifying the bank ID, the ticker (a.k.a., stock symbol), a description, the number of available shares, and the date and time of the sale. Then, brokers record IPO orders on the ledger on behalf of investors. Each order includes the ID of the bank that listed the IPO, the IPO ID, the broker ID, an order ID,[3] and the order details in encrypted form. The order details specify how many shares this investor is willing to buy at each price point.

Later the listing bank invokes the sell-IPO process, and the peers engage in a protocol to determine the clearing price of this IPO and the total demand for shares at this price(which could sometimes be larger than the total available shares). These details are written to the ledger (in the clear), and the IPO is marked as "sold." The share allocation to each investor can be computed locally by the broker from these details and from the order details of the investor: If the number of available shares is S and the total demand for shares at the clearing price is $s \geq S$, then an investor that was willing to buy B shares at the clearing price will actually get only $b = B \cdot S/s$ shares.[4]

The use of a blockchain is highly beneficial for this application, since it provides strong traceability and auditability properties at any point of the IPO life cycle. These advantages, however, can only be realized if we can work with confidential orders without having to rely on a trusted party. Otherwise we would need an out-of-band "supervision" of the trusted party at all times to ensure that no collusion or price fixing occurs, negating most of the utility of the blockchain. What is needed is a system where all the relevant information is recorded on the blockchain, but no single party can ever get access to the secret orders (neither before, during, nor even after the sale).

To achieve the security goals above, our system is designed in a way that prevents any single organization (or even a collusion of a small number of them) from learning anything about the orders. Our use of the MPC-over-Fabric architecture to achieve the goal above is similar to the one described in [4], but not quite the same: One difference is that the entities with the secrets in [4] are the peers themselves (playing roles of buyers and sellers in auctions), whereas in our system the secrets "belong" to the clients of the blockchain (specifically the brokers in our case). In our solution, the brokers break their secrets into shares, so as to ensure that no single peer can see them, and the peers use secure-MPC to perform computations on these shares without ever reconstructing the original secrets.

Our cryptographic protocol is designed to be efficient for the application at hand, and includes a careful combination of additive secret-sharing, GMW-style protocol [6] for comparing integers, and an offline/online method for bridging between binary and mod-N shares of secrets. These ideas yield significant savings (of $3\times$ or more) over straightforward use of existing protocols, see details in Section IV.

In order to integrate secure computation into Fabric, the smart contract code (aka "chaincode") running at the different peers needs to access the relevant secret keys, and they must also be able to communicate with each other. We give the chaincode access to local state (that includes secret keys) using the construct of "decorators" in Fabric, and we built chaincode-to-chaincode communication channels over Fabric's peer-to-peer communication infrastructure. We also adapt existing secure-MPC libraries, both to allow access from within the blockchain, and to extend them to handle our clearing-price protocol.

The rest of the paper is organized as follows: Section II discusses related work. Section III describes the design of our blockchain based IPO trading system. Details of the MPC protocol used in the system are described in Section IV. Performance measurements are listed in Section V. Finally, conclusions are drawn in Section VI.

[1]Recording the transaction on the ledger in Fabric is only done after another process, in which it is sent to an *ordering service* that batches transactions into blocks and checks for ordering consistency. However, this process is orthogonal to our study, so we ignore it in this report.

[2]For example, the organizations running the blockchain could be the exchange, a regulator, an accounting board, and a consortium of banks.

[3]The order ID in our system includes information that lets the broker identify the investor (such as account number). But it can be easily modified to provide investor anonymity, for example by the broker including a random number and keeping a local table that maps these random numbers to the corresponding investor accounts.

[4]Publishing the total demand at the clearing price on the ledger does not harm privacy, as this number can anyways be computed by the investors, knowing only their own demand vs. allocation at the clearing price.

II. Related Work

The challenge of using private/confidential information in blockchain architectures is well recognized. Several (partial) solutions are already available and more were suggested in the literature.

One partial solution is based on *zero-knowledge proofs* (ZKP) [8]. A ZKP is a cryptographic protocol by which one party (a prover) can convince others that some statement is true, without revealing additional information. This provides a good solution to cases where a smart contract depends on the secret data of *a single party*: The party with the secret runs the smart contract on its own, then proves to everyone else that she did so correctly. This approach is used for example in the Zcash currency [9], that utilizes a very general form of ZKPs. However, ZKPs are not sufficient in settings where the smart contract depends on the secret information of more than one participant, for example, if we have two parties with secret balances, and the smart contract needs to compare them.

Another partial solution in a permissioned blockchain is to partition the ledger into separate *channels* (which are essentially different ledgers), thus limiting what each party can see to only the channels that this party can access. This solution is implemented in Fabric, but it still requires that all members of a channel trust each other with all the data on this channel. Below we list some existing blockchain systems that we know of, offering support for confidential data in certain settings.

Blockstream Confidential Assets (CA) [10] uses simple ZKPs in conjunction with additive homomorphic commitments to manipulate secret data on the ledger. This combination of ZKP and additive homomorphism is not strong enough for things like comparing two secret values, however.

Solidus [11] is a system for confidential transactions on public blockchains, aiming to hide even the identities of the participants in those transactions. Designed for banking environments, it uses publicly-verifiable Oblivious-RAM (which combines ZKPs with Oblivious-RAM) to hide the identities of the individual bank customers. Similar to other ZKP-based solutions, Solidus is designed for settings where each transaction depends only on secrets of one participant (i.e., one of the banks).

Hawk [3] is an architecture for a blockchain that can support private data. It uses a trusted component (called a manager) to handle that secret data, which is realized using trusted hardware (such as Intel SGX). The Hawk paper remarks that secure-MPC protocols can also be used to implement the manager, but chose not to explore that option in their context (with a huge number of parties).

Enigma [2], [12] uses secure-MPC protocols to implement support for private data on a blockchain architecture. The main difference between our solution and Enigma is that we integrate secure-MPC protocols within the blockchain architecture itself, while Enigma uses *off-chain computation* for that purpose. A comparison between on-chain and off-chain secure-MPC can be found in [4].

Calcatopia [13] and Tangram [14] These are new startups that promote the use of secure-MPC with blockchain architectures (possibly using some off-chain computations). While the available details on these systems are scant, they seem to be using only secure two-party computation in their examples.

III. Integration in Hyperledger Fabric

We begin by describing some important components in Fabric, specifically peers vs. clients, application chaincode, system chaincode, and peer-to-peer communication. More details on Fabric can be found in [7].

a) Peers and Clients: Being a permissioned blockchain, Fabric features a small set of *peers* that have direct access to the ledger, and a (possibly much larger) set of *clients* that can contact these peers in order to read or modify the ledger. Importantly, it is the client's responsibility to initiate new transactions by contacting one or more peers and sending them a transaction proposal. All the invoked peers are supposed to see exactly the same transaction details, and they all run the application logic against these details and the state of the ledger, to decide if the transaction should be endorsed.

b) Application chaincode: The *application chaincode* implements the application logic, and it is executed during the endorsement of transactions. Currently, the programming languages supported for chaincode are Go, JavaScript (via Node.js), and Java. A chaincode must first be *installed* by an administrator on the peers, then it can be *instantiated*, making the network aware of it. When a client submits a transaction proposal to the peers, this transaction references an existing chaincode, and that chaincode is executed to endorse it.

The chaincode is executed within an environment which is somewhat isolated from the rest of the peer: it is running in a Docker container and communicates with the peer only using gRPC messages over mutual TLS.

c) System chaincode: In contrast to application chaincode, a *system chaincode* (SCC) runs directly in the peer process. SCCs are instantiated at peer's startup, and have access to all of the peer's infrastructure (including direct access to the ledger and the communication layer). Fabric comes with a number of SCCs, providing services to the chaincodes (such as reading from the ledger), and also validating their results.

d) Peer-to-peer communication: Peers communicate with each other over gRPC channels secured by TLS. Since Fabric is a permissioned blockchain, peers have certificates (signed by the peer's organization), which are also leveraged for communication security and identification. Peers can send each other two kinds of messages: either point-to-point or "gossiped" (broadcasting to a group of peers). The communication SCC that we developed for our system uses the point-to-point API.

A. The IPO Trading System

a) System overview: Our system includes a few organizations (such as the exchange, banks, etc.), each owning a peer. The system ensures the confidentiality of the order

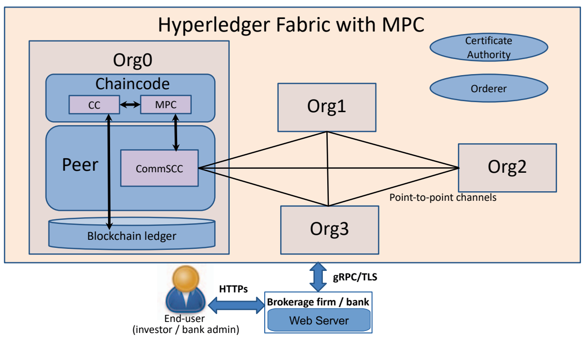


Fig. 1. Architecture overview of blockchain based IPO trading system with organizations Org0, Org1, Org2, Org3.

details (how many shares each investor is willing to buy for each price point), even if *all the organizations but one* collude to try to obtain some information about the orders. Figure 1 depicts a scenario with four organizations, but that number can vary. For a specific IPO, each investor can place their orders via their broker's web server, which acts as a client for the blockchain. The broker breaks the investor's orders into shares, and stores the shares on the ledger in encrypted form. When the time comes to determine the clearing price, the organizations collaborate with each other via an MPC protocol to compute it. This computation is done without the order information ever being revealed in the clear, not even to the organizations that participate in the computation.

Integrating secure computation (and our protocol in particular) into Fabric took a significant design and implementation work. Below we sketch the major changes that we had to implement in order to make it work. There were three main issues that we needed to resolve in our implementation:

- Letting instances of the chaincode in different peers communicate with each other;
- Giving the different chaincode instances their keys and letting them know who the other peers are; and
- Using existing secure-MPC libraries from within Fabric.

We strove to address these issue without drastic changes to either Fabric's codebase or the underlying secure-MPC libraries. Thus, we leveraged Fabric's existing mechanisms, such as system chaincodes, Fabric's peer-to-peer communication layer, and chaincode input decorators (described below). The details are as follows.

b) Communication channels: The default endorsement process in Fabric is stand-alone: the chaincode instance at every peer gets the transaction proposal and the relevant state from the ledger, runs on its own, and returns the intended changes that need to be made. In our case, on the other hand, the instances at different peers have to communicate with each other, to be able to run secure-MPC protocols.

As explained above, a chaincode runs in a separate process within a Docker container environment, which isolates the chaincodes from each other and from the peer. The chaincode can only communicate with the peer it is attached to using gRPC messages. The secure-MPC communication must therefore go through the peers themselves. We augmented Fabric with an additional system chaincode (SCC), called *Communication SCC* (CommSCC, for short), that gives the application chaincode limited access to the peer's built-in communication layer's point-to-point messaging API, in order to send and receive messages.

We also designed a re-usable library in Go that abstracts calls to the system chaincode and that provides a streaming-based interface for peer-to-peer communication.[5] Concretely, this library has two main functions, `MakeLink` and `MakeBufLink` ,that create a communication link[6] to a specific remote peer. These functions output an object satisfying the Go interfaces `io.Writer` and `io.Reader` (i.e., providing streaming-based blocking `Read` and `Write` functions), as well as providing a function `Flush` forcing to send the potentially buffered data.

The particular implementation in Fabric of chaincode-to-peer communication raises a subtle issue: the chaincode is essentially prevented from issuing two calls to a system chaincode in parallel, otherwise answers might get mixed up. We therefore had to enforce a strict sequential ordering to calls to the CommSCC, using a special Go routine. This however restricts the communication patterns that can be used by the secure-MPC library, else deadlocks could occur.

Another issue that we had to solve is peer discoverability, namely how to tell the chaincode in one peer who are the other peers that it needs to communicate with. In our implementation, we opted for simple solutions where peer addresses are specified either statically in configuration files at the peers, or dynamically by the client as part of the transaction proposal. These solutions require no code modifications to Fabric.

c) Local State: The default execution mode for application chaincode in Fabric is that different peers run identical copies of the code on identical proposals and ledger state. In our case we need some specialization of the code at different peers, at least enough to give the different instances their secret keys and tell each one how to contact the other instances that participate in this protocol.

We leverage a Fabric internal mechanism called *chaincode input decoration*. That mechanism lets the peer inject inputs to the local copy of the chaincode input, in addition to what was supplied by the transaction from the client. We implemented a new *chaincode input decorator*, that loads the peer's local from the file system, and, each time a chaincode is invoked, injects these local data into the inputs passed to the chaincode.

d) Using existing secure-MPC libraries: Most of the secure-MPC libraries (including the one that we used) are written for stand-alone applications, rather than to be used as a component in a larger system. We implemented our protocol from Section IV over the publicly available GMW-MPC implementation due to Seung-Geol Choi [15], which is

[5]The reason we provide a streaming-based interface (instead of a message-based interface) is to map more closely to socket interfaces used by most current secure-MPC libraries.

[6]The term "link" is used instead of channel as to avoid confusion with Hyperledger Fabric channels, which are essentially independent ledgers.

written in C++. We had to modify and improve several aspects of that library:

- The original library assumed complete control over sockets for communication. We had to abstract this interface so as to be able to use the Fabric communication mechanisms instead.
- We added support for *layered MPC computation*, where at intermediate points in the computation the parties can decide to reveal partial outputs, then continue with another GMW-MPC that may depend on the revealed values. This is used for an efficient implementation of the binary-search step in our protocol from Section IV.
- We also augmented the library so we can perform multiple instances of GMW-MPC in parallel, on single-instruction multiple-data (SIMD) circuits. (This feature let us replace the binary search step with a more general t-ary search.)
- We also implemented a few one-off protocols (beyond the use of GMW), that are used for pre-computation and use of correlated randomness, as described in Section IV.

Yet another aspect that we had to resolve is that this library (like the vast majority of secure-MPC libraries) is written in C++, while the chaincode that needs to invoke it is in Go. To integrate them, we used the SWIG library [16] that allows calling C++ code from other languages. We replaced the default container running the chaincode in Fabric (called `fabric-ccenv`) by our own container, which includes our secure-MPC library and SWIG. We also had to patch the part of the Node.JS SDK that packages the chaincode in Fabric: The Fabric SDK is responsible to create an archive with the chaincode source files. This archive is uploaded (a.k.a., “installed”) on the blockchain by the SDK, the source files are then compiled, and the resulting executable file is run in the chaincode container. The stock Fabric SDK only includes Go and C source files in the archive, our solution includes a patch to ensure that SWIG source files (`*.swig`, `*.swigcxx`) are also included.

e) Encrypting content on the ledger: Another issue that we tackled is how to get the encrypted content on the ledger to begin with. Roughly speaking, the only way to get new content on the ledger in Fabric is to include it in a transaction proposal, and all the peers must see the same proposal. We therefore had the Fabric client (who prepares the transaction proposal) supply all the encrypted content. Namely, the broker’s web server, acting on behalf of an investor, would break the investor’s order into shares, and encrypt the share for peer j using peer j’s public key. In addition, the broker also encrypted everything under its own key, to make it possible for it to read it back from the ledger, mostly for purposes of displaying it in our user interface.

B. User Interface

Our blockchain based IPO trading system has banks that create and sell IPOs, and brokers representing investors who want to buy shares. The user interface for these entities consists of their respective websites, and the corresponding web servers play the role of clients in the Fabric blockchain architecture.

When the bank administrator creates a new IPO from the bank’s website, the web server of the bank contacts the peers to store the details of the IPO on the ledger (in the clear). Then investors can log in to their brokerage accounts and place orders to buy shares of the new IPO. To place an order, investors in effect specify a histogram indicating how many shares they are willing to buy at each price point. The broker’s web server then shares the histogram among all the peers (as described in Section IV), and uses the public key of each peer to encrypt its share. It also encrypts the entire order under its own secret key, to be used if the investor later wants to read the order from the ledger. Finally, a bank administrator will activate the sell-IPO process from the bank website. The bank’s web server will then invoke the sell-IPO transaction, which will run the protocol from Section IV.

IV. Cryptographic Protocols for IPO Trading

In this section we describe our cryptographic design in some detail: its functionality, the protocol steps, and its security properties. This protocol was designed to be efficient for the task at hand, and some aspects of it required careful combination of additive secret-sharing techniques (allowing easy additions of shared data) with a comparison procedure following the GMW style of protocols [6], and offline/online design. While the underlying add-and-compare functionality is well studied, our methodology for optimizing such protocols yielded significant improvements over a straightforward application of known techniques. We begin by describing the function that we compute and the high-level structure of our protocol, then describe the building block protocols and our methodology for optimizing them.

A. The Clearing-Price Function and High-Level Protocol

In a typical IPO, the number of shares for sale is public information (denoted S), and we also assume a publicly known upper bound P on the clearing price. Investors submit *orders* to the system (via a broker), and then the system computes and makes public the clearing price, as well as the share allocation (who gets how many shares). An order is a complete histogram, $\mathsf{order}_i : [1..P] \to \mathbb{N}$, with $\mathsf{order}_i(p)$ representing the number of shares that investor i wants to buy at price point p.[7] The clearing price of the IPO is the highest price for which all the shares are sold. Namely, given n orders we need to compute

$$\begin{aligned}\mathsf{Price}_{S,P}(\mathsf{order}_1, \ldots, \mathsf{order}_n) \\ = \max\{0 < p \le P : \mathsf{Vol}(p) \ge S\}, \quad (1)\end{aligned}$$

where $\mathsf{Vol}(p) = \sum_{i=1}^{n} \mathsf{order}_i(p)$ is the total demand of shares at price p.[8]

[7]We consider price points as integers in $[1..P]$, and assume rational investors, so the order_i’s are non-increasing: the higher the price, the lower the number of shares the investor is wiling to buy.

[8]This is not the only way to set the IPO price, but in this work we work with this formula.

The main security goal is the privacy of the investors' orders, making the IPO a sealed-bid clearing-price auction. Traditionally, such auctions are performed by a trusted auctioneer (a consortium of investment banks in the case of IPOs). The orders in such traditional auctions, however, are completely known to the investment banks (and their involved staff). The pitfalls of trusting an auctioneer are well documented [17] and further economic advantages of a truly sealed-bid auction for stock-markets is discussed in [18].

Our solution in this work is not a "pure peer-to-peer" multiparty protocol among the investors themselves. While possible in theory, the sheer number of investors would make such a solution infeasible in practice. Instead, our approach is *splitting the trust*: Rather than a single trusted auctioneer, we have a small set of organizations, each with a peer on the blockchain. The system is set such that no single organization (or even a small collusion) learns anything about the orders, and it takes a collaboration of all of them to compute the clearing price. The risk of an untrustworthy auctioneer is much reduced, since it takes a collusion of all the organizations to compromise the orders' confidentiality.

In this work we contend ourselves with the *semi-honest security model*, where all peers are assumed to follow the protocol correctly, and the only security goal is preventing the peers from learning anything about the secret orders from the correct protocol execution. This model may be justified in cases where the application has external audit and compliance mechanisms to ensure that the peers run the specified code. Of course, extending the protocol efficiently to stronger adversarial models is desirable, this is a future-work item.

The Protocol: Let n be the number of investors and m the number of peers, where n could be huge and m is small (e.g., $m = 4$). The protocol proceeds as follow:

a) Step 1: additive sharing of the orders: For each price point $p \in [1\,..\,P]$, each investor i uses additive secret-sharing to share the $\mathsf{order}_i(p)$ value among the m peers, modulo some large number N that is a power of two. Specifically, for each price point p, each investor i chooses $m-1$ random integers $R_{i,1,p}, \dots, R_{i,m-1,p} \in [0\,..\,N-1]$, and sets

$$R_{i,m,p} := \mathsf{order}_i(p) - (R_{i,1,p} + \ldots + R_{i,m-1,p}) \bmod N.$$

In our implementation we use $N = 2^{32}$, and we assume that $N > 2S$ and also $N > 2\mathsf{Vol}(p)$ for every price point p. The i'th investor sends (via its broker) to the j'th peer the shares $R_{i,j,p}$ for all the price points $p \in [1\,..\,P]$.[9] Each peer $j \in [1\,..\,m]$ thus gets $R_{i,j,p}$ for every investor $i \in [1\,..\,n]$ and every price point $p \in [1\,..\,P]$.

b) Step 2: summation: Each peer $j \in [1\,..\,m]$ simply adds up locally its shares (over all investors) for each price point p, obtaining $V_{j,p} := \sum_i R_{i,j,p} \bmod N$. For each price point p, the total demand at p is thus secret-shared additively among the m peers, namely $\mathsf{Vol}(p) = \sum_j V_{j,p} \pmod N$.

c) Step 3: comparisons: Next, the peers need to find the largest price p^* such that $\mathsf{Vol}(p^*) \geq S$. To find p^*, the peers conduct a binary search over the different price points $p \in [1\,..\,P]$. Importantly, in our setting *there is no need to hide the intermediate comparison results* of the binary-search, as they do not reveal anything more than can be deduced from the final clearing price p^*. (In particular they do not reveal the values $\mathsf{Vol}(p)$.) Thus, at each pivot-point p of the binary search, the peers will run a secure-MPC protocol (see Section IV-C)to compare $\mathsf{Vol}(p)$ to S, recovering the comparison result in the clear, and then continue to the next pivot point, until they reach the clearing price p^*.

d) Epilogue: allocations: After the clearing price is determined, the peers still need to compute allocations, i.e., how many shares to sell to each investor. For this, they *recover in the clear* the total demand at the clearing price, $\mathsf{Vol}(p^*)$. the number of shares sold to each investor i is set at $\mathsf{alloc}_i := \mathsf{order}_i(p^*) \cdot S/\mathsf{Vol}(p^*)$. Thus, the total number of shares sold is $\sum_i \mathsf{order}_i(p^*) \cdot S/\mathsf{Vol}(p^*) = \mathsf{Vol}(p^*) \cdot S/\mathsf{Vol}(p^*) = S$, as needed.[10]

B. Optimizing the Cryptographic Building Blocks

The protocol above features some operations that are easier to evaluate over a large ring, and others that are better implemented over the binary field $\mathbb{F}_2$. Namely, we first add the secret-shared values $\mathsf{Vol}(p) = \sum_{i=1}^{n} \mathsf{order}_i(p)$ (step 2, easy to implement as an arithmetic circuit modulo a large enough N), and then compare the results to the number S of available shares (step 3, best done using a Boolean circuit).

A natural approach *which we do not quite follow* is to begin with values that are shared modulo N for the purpose of addition, then convert them into bitwise xor-sharing for the purpose of the comparison protocol. Below we describe an optimization strategy that makes this conversion unnecessary, letting us keep the shares modulo N. Instead we shift the cost to a pre-computation stage that computes correlated randomness, and use this randomness to compute the binary comparison circuit directly on the arithmetic shares. This optimization strategy provides significant savings (at least 3×, maybe more), compared to the approach of converting to xor sharing and running standard GMW on the result.

C. Comparison of Integers

The main building block for the protocol from Section IV-A is the comparison protocol, where the input is an additive sharing of a value V modulo $N = 2^\nu$, and the parties need to compare it to a public value S, getting the result in the clear. Namely, the j'th peer holds a value $V_j \in \mathbb{Z}_N$, we denote $v = \sum_j V_j \pmod N$, everyone knows the public input S, and the desired output is a bit signifying whether $V < S$.

We note that since $V, S < N/2$ then $V - S \in [1 - \frac{N}{2}\,..\,\frac{N}{2} - 1]$. Hence the high-order bit in the binary representation of $\delta = V - S \bmod N$ is one if and only if

[9]In our setting, this is done by posting to the ledger an encryption of these $R_{i,j,p}$'s under the public key of peer j.

[10]We note that revealing $\mathsf{Vol}(p^*)$ is not a security problem, since every investor (with $\mathsf{alloc}_i(p^*) > 0$) can deduce it just from its own input and output, using $\mathsf{Vol}(p^*) = S \cdot \mathsf{order}_i(p^*)/\mathsf{alloc}_i$.

$V < S$. In other words, the desired output bit can be computed as $\mathrm{MSB}(V - S \bmod N)$.

Pre-Computation Stage. The peers begin by performing a small number of real OTs, then using OT extension [19], [20] to get sufficiently many random-OTs for all the computations that follow.

Next, each peer picks a random number modulo N, say the j-th peer picks A_j. The peers then run a GMW protocol to compute a bit-wise xor-sharing of the sum $B = \sum A_j \bmod N$ (evaluating a carry-save-adder binary circuit using some of the random-OTs). For each bit position $\ell \in [0..\nu - 1]$, denote the share of the ℓ'th bit of B held by the j'th peer by $B_{j,\ell}$, namely we have

$$\sum_{\ell=0}^{\nu-1} 2^\ell \Big(\bigoplus_{j=1}^{m} B_{j,\ell}\Big) \;=\; B \;=\; \Big(\sum_{j=1}^{\nu} A_j \bmod N\Big) \;.$$

On-Line Stage. In this stage there is a public input S, the j-th peer has private input $V_j \in \mathbb{Z}_N$, we denote $V = \sum_j V_j \pmod N$, and the goal is to output the indicator bit $V < S$. Instead of converting the additive shares V_j's to xor-sharing of V and run GMW, we use the A_j's and $B_{j,\ell}$'s from before to directly compute the result, as follows:

Each peer locally computes the difference $D_j := V_j - A_j \bmod N$, and just broadcasts D_j to everyone. Having received the differences D_j for all $j \in [1..m]$ (including its own), each peer computes $\Delta := (\sum_j D_j) - S \bmod N$ locally, and we have

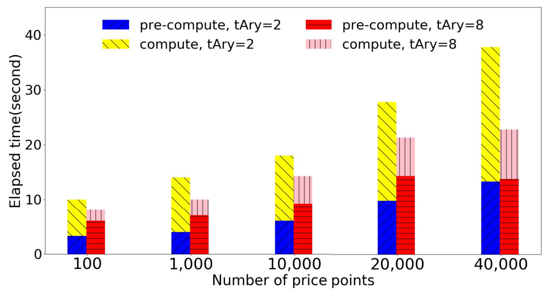
$$\Delta = \Big(\sum_j V_j\Big) - \Big(\sum_j A_j\Big) - S = V - S - B \pmod N \;.$$

In other words, we have $\Delta + B = V - S \pmod N$, where the peers all know Δ in the clear, and they have a bit-wise xor-sharing of B. Recall that the output bit we seek is $\mathrm{MSB}(V - S \bmod N) = \mathrm{MSB}(\Delta + B \bmod N)$, so all we need is to run the GMW protocol to compute the MSB of $\Delta + B \bmod N$. Since N is a power of two, then the MSB of $\Delta + B \bmod N$ is the ν'th bit of $\Delta + B$ (without mod N reduction), hence the peers just run GMW to compute the MSB in the integer addition circuit, using the random-OTs that they have from the pre-computation stage. This high-order bit can be computed using a carry-look-ahead circuit with at most $3n$ multiplication gates and depth of only $\log \nu$.

We note that the naive way of implementing the comparison we need consists of converting the additive shares V_j to an xor-sharing of the bits of their sum. It would use a similar circuit to above, except that instead of computing only the high order bit it would require computing all the bits, which takes at least twice as many multiplications (and likely much more). Hence the conversion would take $> 6\nu$ multiplications, followed by the same 3ν multiplications for the comparison, which is more than $3\times$ the cost of our protocol.

The following theorem about the security of the comparison protocol is proven in the full version.

Theorem 1 (Security of the comparison protocol.): The protocol from Section IV-C securely computes the comparison function in the stand-alone honest-but-curious model.



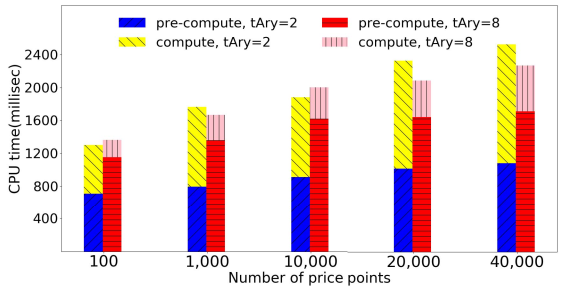


Fig. 2. Elapsed and CPU time of running the IPO trading application on blockchain using MPC.

V. Evaluation

We evaluated the performance of our system on a machine with Intel i7 6567U CPU at 3.30GHz, 16GB memory, and 500GB SSD disk. The software environment included Ubuntu 16.04 with Linux kernel 4.15.0 as the operating system, and our customized Hyperledger Fabric v1.1 with point-to-point channels and private data. In all of our experiments we ran Fabric with four organizations, each having a single peer (and every peer establishes three point-to-point channels to the others). We vary the number of price points in the orders from 100 to 40,000.

In Fig. 2 we show the total elapsed time of running the MPC protocol from Section IV over Fabric, as well as the CPU time (ignoring communication).The figures describe two different instances of the protocol from Section IV, one using binary search to find the clearing price (tAry=2) and the other using an 8-ary search (tAry=8). As we can see the difference between these variants is not large. When tAry is 8, the total time for running the protocol ranges from 8 seconds for 100 price points, to 22.75 seconds for 40000 price points. While when tAry is 2, the protocol run time varies from 13 seconds to 33.75 seconds (hence the time grows sub-linearly with the number of points in both cases).

Fig. 2 also reports separately the running times of the offline vs. online phases of the protocol. (Even though in our specific implementation we run both phases during endorsement, it is possible in principle to run the offline phase earlier, and then execute only the online phase during endorsement.)

Comparing the CPU time to the elapsed time shows that more than 90% of the protocol time is spent on communication (and even more during the online phase). This is due to the relatively high bandwidth required, and to the fact that our current implementation of communication channels over Fabric is far from being optimized. (This is an important future-work item.)

VI. Conclusions and Future Work

In this work we designed and implemented the clearing price mechanism for IPOs using secure-computation over the Hyperledger Fabric blockchain architecture. This work involved both a new cryptographic design for an efficient protocol to determine the clearing price, and a significant integration effort to be able to run this protocol over Fabric. While our design is more than fast enough for the IPO application, there is still a lot of room for optimization. For example, sending messages through *CommSCC* is an overhead. This overhead could be mitigated if the chaincode had a native API to send messages to executions taking place on other peers.

References

[1] D. Tapscott and A. Tapscott, *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. Penguin Books Canada, 2018. [Online]. Available: https://books.google.com/books?id=3EmitQEACAAJ

[2] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *IEEE Symposium on Security and Privacy Workshops*. IEEE Computer Society, 2015, pp. 180–184.

[3] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016, pp. 839–858.

[4] F. Benhamouda, S. Halevi, and T. Halevi, "Supporting private data on hyperledger fabric with secure multiparty computation," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 357–363, long version available from https://shaih.github.io/pubs/bhh18.html.

[5] A. C.-C. Yao, "How to generate and exchange secrets (extended abstract)," in *27th FOCS*. IEEE Computer Society Press, Oct. 1986, pp. 162–167.

[6] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *19th ACM STOC*, A. Aho, Ed. ACM Press, May 1987, pp. 218–229.

[7] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *EuroSys 2018*, R. Oliveira, P. Felber, and Y. C. Hu, Eds. ACM, 2018, pp. 30:1–30:15. [Online]. Available: http://doi.acm.org/10.1145/3190508.3190538

[8] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989.

[9] "Zcash - all coins are created equal," https://z.cash/, 2016, accessed Dec 2017.

[10] A. van Wirdum, ""confidential assets" brings privacy to all blockchain assets: Blockstream," Bitcoin Magazine, https://bitcoinmagazine.com/articles/confidential-assets-brings-privacy-all-blockchain-assets-blockstream/, April 2017.

[11] E. Cecchetti, F. Zhang, Y. Ji, A. E. Kosba, A. Juels, and E. Shi, "Solidus: Confidential distributed ledger transactions via PVORM," in *ACM CCS 17*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM Press, Oct. / Nov. 2017, pp. 701–717.

[12] G. Zyskind, O. Nathan, and A. Pentland, "Enigma: Decentralized computation platform with guaranteed privacy," *CoRR*, vol. abs/1506.03471, 2015. [Online]. Available: http://arxiv.org/abs/1506.03471

[13] D. C. Sánchez, "Raziel: Private and verifiable smart contracts on blockchains," Cryptology ePrint Archive, Report 2017/878, 2017, https://eprint.iacr.org/2017/878.

[14] R. Zou, "Tangrum: A next-generation decentralized computation platform," https://medium.com/tangrum/tangrum-a-next-generation-decentralized-computational-platform-e6851c85ebb1, 2018, accessed August 14, 2018.

[15] S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D. Rubenstein, "Secure multi-party computation of Boolean circuits with applications to privacy in on-line marketplaces," in *CT-RSA 2012*, ser. LNCS, O. Dunkelman, Ed., vol. 7178. Springer, Heidelberg, Feb. / Mar. 2012, pp. 416–432.

[16] "Swig," http://www.swig.org/, 2018.

[17] S. Benediktsdottir, "An empirical analysis of specialist trading behavior ar the new york stock exchange," FRB International Finance Discussion Paper (876), 2006.

[18] C. S. Jutla, "Upending stock market structure using secure multi-party computation," Cryptology ePrint archive: Report 2015/550, 2015.

[19] D. Beaver, "Correlated pseudorandomness and the complexity of private computations," in *28th ACM STOC*. ACM Press, May 1996, pp. 479–488.

[20] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *CRYPTO 2003*, ser. LNCS, D. Boneh, Ed., vol. 2729. Springer, Heidelberg, Aug. 2003, pp. 145–161.