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

Fabrice Benhamouda*[1], Angelo DeCaro†[1], Shai Halevi‡[1], Tzipora Halevi§[2], Charanjit jutla¶[1], Yacov Manevich‖[1], and Qi Zhang**[1]

[1]IBM Research
[2]Brooklyn College

October 22, 2018

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 IPOs. Fabric is a platform for "blockchain for business," providing a robust mechanism for a set of "peers" to jointly maintain a distributed immutable ledger. As with any blockchain, using *confidential data* on the ledger is a challenge, and we use cryptographic secure multiparty computation (MPC) to address this challenge. The use of secure MPC to enable confidential data on blockchains was proposed before, but this approach only now begins to find its way to real-life systems. In particular, ours is the first attempt to integrate it in a complex blockchain environment such as Fabric.

To enable this approach, we integrated mechanisms into Fabric to let the peers access local information such as their respective keys, and also send messages to each other while executing smart contracts. We added to Fabric a library that implements the required cryptographic tools, and made it accessible from the smart contract. We then 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.

1 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" [13]. 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

*fabrice.benhamouda@normalesup.org
†adc@zurich.ibm.com
‡shaih@alum.mit.edu
§thalevi@nyu.edu
¶csjutla@us.ibm.com
‖yacovm@il.ibm.com
**q.zhang@ibm.com

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 (Some are described in [18, 10, 3]). One example is 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.

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 access to the cleartext data requires having the secret key. 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 [15, 7], 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., [18, 10, 3]), 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 [1] (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 implemented a system for IPO trading. IPO trading 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 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 a unique ID, the ticker (a.k.a., stock symbol), a description, the number of available shares, and the date and time of the sale. Then, brokerage firms can record IPO orders on the ledger on behalf of investors. Each order includes the IPO ID, an order ID,[3] and the order details in encrypted form. (Specifically, 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, as well as the share allocation (which investor gets how many shares). All of these details are written to the ledger (in the clear), and the IPO is marked as "sold."

[1] Recording the transaction on the ledger 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] In our actual system, the order ID includes the investor ID for simplicity. The system can be easily extended to ensure anonymity of the investors as follows: the brokerage firms use randomly-generated ID and keep in a separate private database the mapping between order IDs and investors.

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 will need an out-of-band "supervision" of the trusted party at all times to ensure that no collusion or price fixing occurs, negating most/all the utility that can be derived from 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, but not quite the same as, the one described in [3]: One difference is that the entities with the secrets in [3] 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 (i.e., the brokerage firms). In our solution, the brokerage firms 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 novel ideas in its combination of additive secret-sharing, GMW-style protocol [7] for comparing integers, and a new offline/online method for converting between binary and mod-N shares of secrets.

In order to integrate secure computation (and our protocol in particular) into Fabric, we provide the chaincode at the different peer with access to their secret keys, as well as the ability to communicate with each other. For the former we used the construct of "decorators" in Fabric, and for the latter 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 following: Section 2 discusses related work; Section 3 describes the design of our blockchain based IPO trading system. Details of the MPC protocol used in the system are described in Section 4, performance measurements listed in Section 5, and conclusions are in Section 6.

2 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 any 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 prove to everyone else that it did so correctly. For example, in a setting where participants have accounts with secret balance on the ledger, a participant wishing to buy a $100-item can use ZKP to prove that its balance is greater than $100. This approach is used for example in the Zcash currency [16], that supports 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) [14]

use simple ZKPs in conjunction with additive homomorphic commitments to manipulate secret data on the ledger. For example, two users whose secret account balances are encrypted with additively homomorphic commitments, can agree privately (off chain) on a price of an item. The first user can then subtract this amount from her balance and add it to the balance of the other user (using homomorphism), and prove to everyone (using ZKP) that the amount added to the second balance is equal to the amount subtracted from the first. But note that the transaction amount itself must be fully known to the first party, this combination of ZKP and additively homomorphic commitments is still not strong enough to compare two secret values. This combination of ZKP and additive homomorphism, however, is not strong enough to compare two secret values, for example.

Solidus [5] is a system for confidential transactions on public blockchains, aiming to hide not only the details of the different transactions but also 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 [10] 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 very many parties).

Enigma [18, 19] 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. Some comparison between on-chain and off-chain secure-MPC can be found in [3].

Calcatopia [11] and Tangram [17] 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 seems to be using only two-party secure-computation in their examples.

3 Integration in Hyperledger Fabric

Our IPO trading system is integrated into Fabric. Below we briefly discuss some relevant aspects of Fabric, then elaborate on our integration work.

3.1 Basics of Hyperledger Fabric

This subsection introduces three critical components in Fabric: application chaincode, system chaincode, and peer to peer communication. more details on Fabric itself can be found in [1]. Also note that, in Fabric, it is the client's responsibility to invoke new transactions by contacting one or more peers and sending them a transaction proposal to be endorsed, and all the invoked peers will see exactly the same transaction details.

Application chaincode. The *application chaincode* is the program code that implements the application logic, which runs during the endorsement of transactions. It is the central part of a distributed application in Fabric and may be written by an *untrusted developer*. Currently, the programming languages supported for chaincode are Go and JavaScript (running in Node.js runtime). A chaincode needs first to be *installed* by an administrator on the peers that are supposed to execute it, 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, to be run for endorsing it.

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

of the actual language in which the chaincode is implemented.

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. Fabric can also leverage Go plugins to allow a peer's administrator to embed new functionality by adding SCCs as needed (though we did not use plugins in this project).

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:

- Point to point: These messages are exchanged between peers on both sides of a connection, and are never forwarded further. The communication layer exposes an API to higher layers of the peer that attaches the sender peer's enrollment certificate for every message sent to the peer, and in this way - an application layer inside the peer can employ this mechanism as well and doesn't need to implement its own authentication mechanism to prevent message spoofing. The *CommSCC* in our study uses this point-to-point messaging ability in order to send messages securely to peers denoted by the caller chaincode.

- Gossiped: These messages are disseminated via a gossip protocol: when sending a message, it is first signed by the peer's private key corresponding to the enrollment certificate and sent to random k_1 peers, and also periodically k_2 peers are selected to pull messages. The membership view of a peer (the peers a peer knows about) is built via this mechanism by having peers disseminate their enrollment certificates and messages attesting for their endpoints, as well as channel membership messages for each channel, that contains channel meta-data information such as the ledger height of the peer. This mechanism is also how blocks are disseminated between peers, with the slight modification that the peers do not sign the blocks but the ordering service nodes do.

3.2 Our Modifications

System overview. Our system includes a few organization, each one owning a peer (cf. Footnote 2). The system ensures the confidentiality of the order 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 depict a scenario with four organizations, but that number can vary. For a specific IPO, each investor can place their orders via their brokerage firm's web server, which acts as a client for the blockchain and encrypts its investor's orders and stores the encrypted data into the blockchain ledger. When the time for IPO trading arrives, the organizations collaborate with each other via an MPC protocol to compute the clearing price. This computation is done without the decrypted 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.

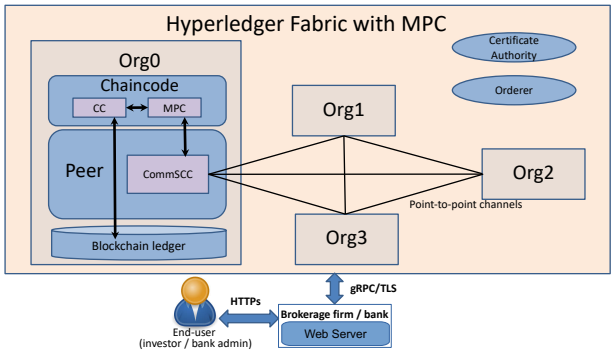

Figure 1: Architecture overview of blockchain based IPO trading system with organizations Org0, Org1, Org2, Org3.

We strove to address these issue without drastic changes to either Fabric's codebase or the underlying secure-MPC libraries. For this we leveraged Fabric's pluggability and existing mechanisms. The details are as follows.

Communication channels. The default endorsement process in Fabric is stand-alone: the chaincode instance at every peer gets the relevant state from the ledger, runs on its own, and returns the intended changes that needs 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 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. Specifically, CommSCC provides the following primitives:

- **Send**: Sends a payload to a remote peer. The chaincode must uniquely identify the remote peer by a predicate, and the SCC buffers the message internally and sends back an acknowledgment to the chaincode.
- **Receive**: The SCC returns a message from the intended source peer if such message has been received and is waiting in the internal buffer, or waits a specified amount of time for reception of a message from the intended source peer.

CommSCC is made available to all chaincodes, access can be restricted to avoid abuses, if needed.

One 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 a simple solution where peer addresses are specified either in configuration files or as argument to the chaincode (as part of the transaction proposal), this solution requires no code modifications to Fabric.

Private Data. The default execution mode for application chaincode in Fabric is that different peers run identical copies of the code. In our case we need some specialization of the peers, at least enough to give the different peers their secret keys and tell each peer how to contact the other peers 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, in addition to what was supplied by the transaction from the client. We implemented a new *chaincode input decorator*, that loads the peer's secret from the file system, and, each time a chaincode is invoked, it injects the peer's local data (including keys) into the inputs passed to the chaincode.

Because the decorator is essentially customized code, it can be perform additional access control as well. For example, the decorator can pass the peer's secret only to specific chaincodes, or only when the chaincode is invoked by a specific user.

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 4 over the publicly available GMW-MPC implementation due to Seung-Geol Choi [6], which is written in C++, but had to modify and improved 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.

- 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.)

For the pre-computation and use of correlated randomness as described in Section 4, we also implemented additional protocols beyond just GMW-MPC.

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 [12] that allows calling C++ code from other languages. Specifically, SWIG was added to the containers running the application chaincode.

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 client (who prepares the transaction, i.e., a brokerage firm acting on behalf of an investor) supplying all the encrypted content. This means that it encrypted any content that should only be seen by peer j using peer j's public key. In addition, the client 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.

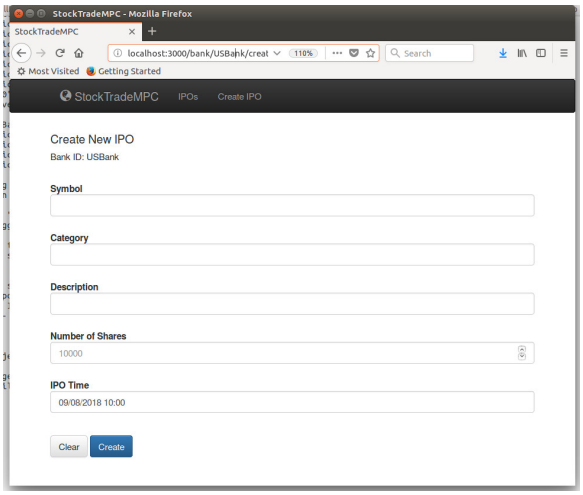

Figure 2: Creating a new IPO

3.3 User Interface

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

When creating a new IPO, an administrator at the bank logs into the bank's web server and fills a form with all the public IPO parameters. Namely they specify a unique ID, the stock ticker symbol, category, description, creation date, sale date and time,and the number of available shares for sale. The IPO creation screen is shown in Fig. 2. The IPO information is then recorded on the ledger in the clear.

When an investor accesses its account at a broker-

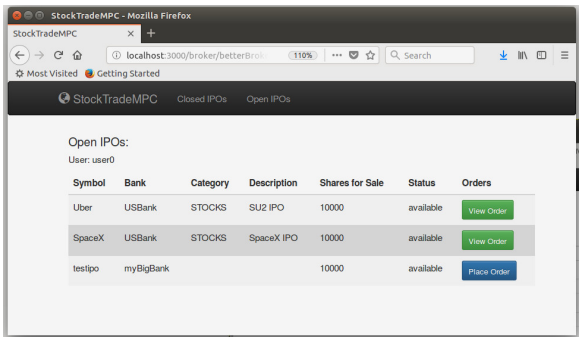

Figure 3: List of open IPOs

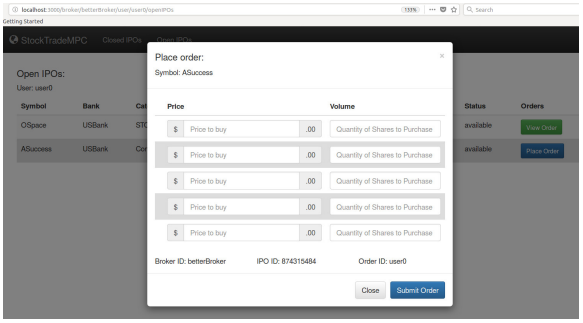

Figure 4: Submitting a new order

age firm, it see a list all the open IPOs that are available for sale, and can submit an order for any of them (or view its order if it was already submitted). This is illustrated in Fig. 3. When submitting an order, the investor in effect specifies a complete histogram, saying how many shares he/she is willing to buy at each price point. This is done by specifying a set of price/volume points, as depicted in Fig. 4. When the investor submits the order, the brokerage web-server translates it into a complete histogram. then it The brokerage firm web server then shares the histogram among all the peers (as described in Section 4), 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. (For example when clicking on the "View Orders" button in Fig. 3.)

Finally, a bank administrator will activate the sell-IPO process from the bank web site. This is done by

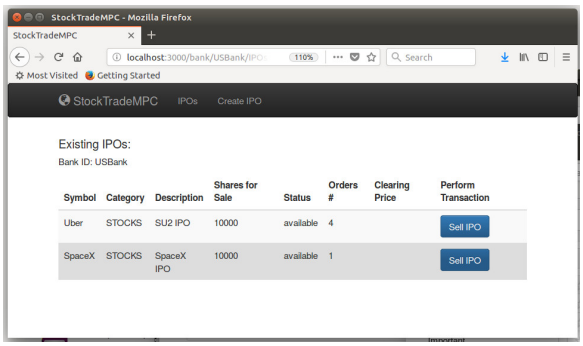

Figure 5: Selling an IPO

clicking the "Sell IPO" button as seen in Fig. 5. The bank web server will then invoke the sell-IPO transaction, which will run the protocol from Section 4.

4 Cryptographic Protocol for IPO Trading

In this section we describe our cryptographic design in some detail: its functionality, the protocol steps, and its security properties. Some aspects of this protocol are novel, in particular its combination of additive secret-sharing (allowing easy additions of shared data) with a comparison procedure following the GMW style of protocols [7], and offline/online design. While this integer add-and-compare functionality was well studied (cf. [4] for example), in this work we describe novel ideas for implementing it more efficiently, especially with regards to pre-computation (aka "correlated randomness generation"). As the combination of additions and comparisons that we implement is quite common in many auction settings, our techniques should also hold in general for other auctions.

4.1 The Clearing-Price Function

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. One can also consider a reserve price, i.e. a lower

bound on the clearing price, but for ease of exposition we will let that be zero. Investors submit *orders* to the system (via a brokerage firm), and then the system computes a clearing price, as well as share allocation (who gets how many shares). An order is a complete histogram, $\mathsf{order}_i : [1\,..\,P] \to \mathbb{N}$, with $\mathsf{order}_i(p)$ the number of shares that investor i wants to buy at price point p. [4] 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

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

where $\mathsf{Vol}(p) = \sum_{i=1}^{n} \mathsf{order}_i(p)$ is the total demand of shares at price p. [5] 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 [2] and further economic advantages of a truly sealed-bid auction for stock-markets is discussed in [9].

To mitigate such issues, our approach is *splitting the trust*: Instead of a single trusted auctioneer, we have a small set of organizations (four of them by default). Each organization owns a peer on the blockchain. The system is set such that no single organization 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.

[4] We assume rational investors, so the order_i's are all non-increasing functions: the higher the price, the lower the number of shares. Furthermore, price points are assumed to be integers in $[1\,..\,P]$.

[5] This is not the only way to set the IPO price, but for simplicity we stick in this work to only that formula.

4.2 The Protocol

We note that our solution 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 solution infeasible in practice. Instead, in our solution the investors *secret-share* their orders among all the peers, and the peers jointly compute the clearing price from their shares. For simplicity, in this initial work we adopt the *semi-honest security model*, where all peers are assumed to follow the protocol correctly. The security goal in this model is preventing the peers from learning anything about the secret orders from the correct protocol execusion. The protocol proceeds as follow:

Step 1: sharing the orders. The function that we compute, i.e., the clearing price from Eq. (1), requires many integer additions (to compute $\mathsf{Vol}(p) = \sum_i \mathsf{order}_i(p)$ for every price-point p), and also comparisons with a publicly known number (to find the highest p s.t. $\mathsf{Vol}(p) \ge S$). The GMW secure-MPC protocol [7] forms the basis of our solution. The GMW protocol can be used to securely compute any function on confidential inputs owned by different parties without revealing anything but the output of the function. However, it requires the computed functions to be represented as a boolean circuits, and implementing the integer additions this way will be rather inefficient. (GMW-style protocol need as many rounds of communication as the depth of the circuit, which can then become very expensive in our case.) Instead, in our protocol we use additive secret sharing for the additions, then convert the sums to binary representation for the comparisons.

Let n be the number of investors and m the number of peers (where n could be large and $m = 4$ by default). For each price-point $p \in [1\,..\,P]$, each investor i uses linear secret-sharing to share the number $\mathsf{order}_i(p)$ among the m peers, modulo some large number N. Specifically, for each price point p, the investor chooses $m-1$ random integers $R_{i,1,p}, \ldots, R_{i,m-1,p} \in [0\,..\,N-1]$, and sets

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

(Note that the sum of the $R_{i,j,p}$'s equals $\mathsf{order}_i(p)$

modulo 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 to the j'th peer the shares $R_{i,j,p}$ for all the price points $p \in [1..P]$. [6] Each peer $j \in [1..m]$ thus gets $R_{i,j,p}$ for every investor $i \in [1..n]$ and every price point $p \in [1..P]$. (The resolution of the price points p is a parameter that we can adjust, in our implementation we use reslutions between \$1 and \$0.01.)

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 linearly among the m peers, namely $\mathsf{Vol}(p) = \sum_j V_{j,p} \pmod N$.

Step 3: comparisons. Next, the peers need to find the largest price p^* such that $\mathsf{Vol}(p^*) \geq S$. Recall that S is a known quantity and all the $\mathsf{Vol}(p)$ values are shared additively among them.[7] For that, 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 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^*.

The comparison protocol. The comparison protocol itself is essentially just the classical GMW protocol [7], applied to the comparison-to-S function. But GMW requires as input the *mod-2 sharing of the individual bits*. Namely, before we can use the GMW protocol to compare the k-bit integer $X = \mathsf{Vol}(p) = \sum_{\ell=0}^{k-1} X_\ell \cdot 2^\ell$ to S, we need the peers $j \in [1..m]$ to hold shares $A_{j,\ell} \in \{0,1\}$, such that $X_\ell = A_{1,\ell} \oplus \cdots \oplus A_{m,\ell}$ for all $\ell = 0, \ldots, k-1$.

As described above, however, the peers instead hold integers $V_{j,p} \in [0..N-1]$ such that $\mathsf{Vol}(p) = \sum_j V_{j,p} \pmod N$. Hence, before running each comparison protocol, we must first run a sub-protocol to convert the mod-N shares $V_{j,p}$'s to mod-2 shares of the bits $A_{j,\ell,p}$'s. We now show a novel way of performing this task, where most of the expensive part is shifted to a pre-computation phase (aided by the fact that the modulus N that we use is a power of two, $N = 2^k$).

Fix a particular pivot p, and from now on we drop the p subscript, writing V_j instead of $V_{j,p}$, etc. We also denote $X = \mathsf{Vol}(p)$ where X is a k-bit integer. In an offline pre-computation phase, each peer j chooses a random number $A_j \in [0..N-1]$, and let $A_{j,0}, \ldots, A_{j,k-1}$ be its binary representation (i.e., $A_j = \sum_{\ell=0}^{k-1} A_{j,\ell} \cdot 2^\ell$). The peers then run a classical GMW protocol, to compute a binary secret sharing of the sum $B = \sum_{j=1}^m A_j \mod N$. Since N is a power of two, then the reduction modulo N is done simply by ignoring the high order bits of B. At the end of this GMW protocol, the peers will have a mod-2 sharing of the bits of B, namely each peer j will have shares $B_{j,0}, ..., B_{j,k-1}$, such that

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

Note that the A_j's are completely independent of the shares V_j that we will later want to convert, hence this computation can be done off line, before the V_j's are known.

In the online phase, once each peer j computed its share V_j (by summing as above), it locally computes the k-bit difference $D_j := V_j - A_j \bmod N$, and just broadcasts D_j to all the other peers. (No secret-sharing required here – see below for why this is secure.) Each peer j, having received the differences $D_{j'}$ from all the peers $j' \in [1..m]$ (in-

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

[7] Essentially the same protocol can be used even when S is secret-shared among the peers rather than being public. But in our case it is publicly known.

cluding its own), locally compute the number $\Delta := (\sum_j D_j) - S \bmod N$, and we have

$$\Delta = \sum_j (V_j - A_j) - S = (\sum_j V_j) - (\sum_j A_j) - S \\ = X - S - B \pmod N.$$

In other words, we have $\Delta + B = X - S \pmod N$, where the peers all know Δ in the clear, and they have mod-2 secret sharing of the bits of B. The peers now run a classical GMW protocol to determine the high-order $(k-1)$-th bit of the sum of $\Delta+B$, and we claim that this bit is exactly the boolean value $(S > X)$.

To prove this claim, recall that we have $\Delta + B = X - S \pmod N$, and we know that $X = \mathsf{Vol}(p)$ (at the current pivot p) and the number of shares S are both smaller than $N/2$ (and N is a power of two). We prove that $S > X$ iff the $(k-1)$-th bit of $\Delta + B$ is zero.

First, suppose that $S \le X$, then $0 \le X - S \le X < N/2$, and hence also $(\Delta + B \bmod N) < N/2$, so the $(k-1)$-th bit bit of $(\Delta + B \bmod N)$ is zero. But $N = 2^k$, so it must be that the $(k-1)$-th bit of $\Delta+B$ is zero even without the mod-N reduction. In the other direction, if $S > X$ then $N > (X-S \bmod N) = X-S+N \ge N-S > 2^{k-1}$. Hence, $(\Delta+B \bmod N) > 2^{k-1}$, so the $(k-1)$-th bit of $(\Delta + B \bmod N)$ is one, and therefore so is the $(k-1)$-th bit of $\Delta+B$ without the mod-N reduction. ■

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 requested volume at the clearing price, $\mathsf{Vol}(p^*)$. the number of shares sold to each investor i is set at $\mathsf{alloc}_i := \mathsf{order}_i(p^*) \cdot S/\mathsf{Vol}(p^*)$ (with S the number of available shares). 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.

Note that by construction we have $\mathsf{Vol}(p^*) \ge S$, so investors get at most what they asked for at this price point. We typically expects $\mathsf{Vol}(p^*) = S$ (so investors get exactly what they asked for), since for any larger price $p > p^*$ we have $\mathsf{Vol}(p) < S$. But this is not guaranteed, and it could happen that $\mathsf{Vol}(p^*) > S$. We also 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 $\mathsf{Vol}(p^*) = S \cdot \mathsf{order}_i(p^*)/\mathsf{alloc}_i$.

4.3 Security of this Protocol

We now sketch the proof that the protocol is secure in the semi-honest model. (A complete proof is deferred to the long version of this report.) We need to show that the view of each peer (or even small collusion of upto $m-1$ peers) yields no more information than can be deduced from the inputs and outputs of the computation alone. First, by the very straightforward description of the additive secret-sharing, the view of any $m-1$ peers is random and independent of the actual orders $\mathsf{order}_i(p)$ of the i-th investor. Similar, randomness guarantees hold for other secret-sharings in the protocol, e.g. the secret-sharing of $B = \sum_j A_j$. The only other part in the protocol which does not use a classical GMW protocol is the revealing of $D_j = V_j - A_j \bmod N$ by each peer j.

Recall, V_j is the sum of secret shares of peer j, and this peer blinds the value V_j by the random A_j before "revealing it" to the other peers. But recall that A_j is completely random, and the security of the GMW protocol implies that the view of the other peers in the protocol for computing the sum $B = \sum_j A_j$ does not reveal anything (computationally) on A_j. Hence A_j is still random for the other peers, and therefore revealing $D_j = V_j - A_j \bmod N$ does not compromise the secrecy of V_j.

5 Evaluation

In this section, we evaluate the performance of our blockchain based IPO trading application. We run our test 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.

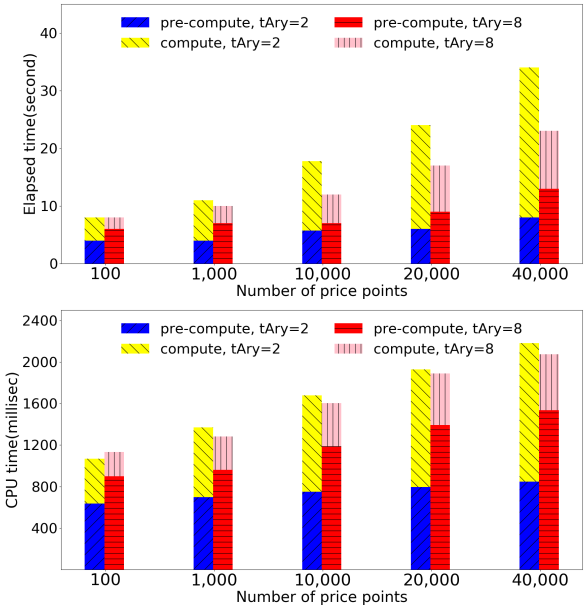

Figure 6: Elapsed and CPU time of running the IPO trading application on blockchain using MPC.

In Fig. 6 we show the total elapsed time of running the MPC protocol from Section 4 over Fabric, as well as the CPU time (ignoring communication). The figures describe two different instances of the protocol from Section 4, 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 23 seconds for 40000 price points (hence the time grows sub-linearly with the number of points).

In these figures we also report separately the running times of the offline vs. online phases, notice that almost all the time is spent during the offline phase. (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 almost 90% of the protocol time is spent on communication. 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.) We note that, when tAry is 8, as the number of price points increases from 100 to 40000, the total data transferred by the chaincode containers grows from 9.5MB to 98.5MB, which is a sub-linear growth.

6 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 an example, sending messages through *CommSCC* is an overhead that could be mitigated if the chaincode had a native API to send messages to executions taking place on other peers.

References

[1] 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 R. Oliveira, P. Felber, and Y. C. Hu, editors, *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, pages 30:1–30:15. ACM, 2018.

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

[3] 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)*, pages 357–363. IEEE, 2018. Long version available from `https://shaih.github.io/pubs/bhh18.html`.

[4] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In R. Dingledine and P. Golle, editors, *Financial Cryptography and Data Security*, pages 325–343, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. Long version at `https://ia.cr/2008/068`.

[5] E. Cecchetti, F. Zhang, Y. Ji, A. E. Kosba, A. Juels, and E. Shi. Solidus: Confidential distributed ledger transactions via PVORM. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 701–717. ACM, 2017.

[6] 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 O. Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 416–432. Springer, Heidelberg, Feb. / Mar. 2012.

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

[8] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

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

[10] 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*, pages 839–858. IEEE Computer Society Press, May 2016.

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

[12] Swig. `http://www.swig.org/`, 2018.

[13] D. Tapscott and A. Tapscott. *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. Penguin Books Canada, 2018.

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

[15] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, 1986.

[16] Zcash - all coins are created equal. `https://z.cash/`, 2016. Accessed Dec 2017.

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

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

[19] G. Zyskind, O. Nathan, and A. Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *CoRR*, abs/1506.03471, 2015.