Publicly Verifiable and Secrecy Preserving Periodic Auctions

Hisham S. Galal and Amr M. Youssef

Concordia Institute for Information Systems Engineering,
Concordia University, Montréal, Quebéc, Canada

Abstract. In lit markets, all the information about bids and offers in the order book is visible to the public. With this transparency, traders can discover prices and adjust their strategies accordingly. On the other hand, submitting a bulk order by a financial institution will have a significant impact on the market price. Therefore, financial institutions prefer trading on dark pools, which hide order books, to avoid potential losses from negative market impact. However, the lack of transparency hurts price discovery, results in poor execution of trades, and promotes illicit behaviors such as front-running. Hence, several financial regulations have limited trading on dark pools. Subsequently, periodic auctions, which are considered regulation-compliant alternatives to dark pools, have witnessed a surge in trading volumes. Unfortunately, similar to dark pools, there are no guarantees that the operators will neither exploit their exclusive access to the order book nor incorrectly compute the market-clearing price. In this paper, we build a publicly verifiable and secrecy preserving periodic auction protocol to address these challenges. We utilize aggregate Bulletproofs to prove the ordering on a vector of commitments. To alleviate the burden on traders' computation resources and achieve public verifiability, the protocol delegates the verification of the operator's work to a smart contract. We evaluate the protocol's performance, and the results show that it is practical and feasible to deploy.

Keywords: Periodic Auction, Zero-Knowledge Arguments, Blockchain

1 Introduction

Investors use financial exchanges to trade equities and securities. Generally, an exchange is a continuous double-sided auction between buyers and sellers [1]. It records all outstanding limit orders in an order book. A *limit* order consists of a unit price, a quantity of an asset, and a direction to indicate whether it is a *bid* by a buyer or an *offer* by a seller. If the order book is transparent and accessible to the public, the exchange is known as a *Lit market*. On the contrary, a *dark pool*, which is favored by financial institutions, is an exchange that hides its order book from traders [1].

To understand the main benefit of dark pools, it is worth considering the problem institutional investors face in lit markets. Suppose that Bob is an institutional investor who uses a lit market to buy one million USD worth of an

arbitrary asset. The sellers will notice Bob's bid. Hence, they anticipate the increased demand and react by moving their offers to higher prices so that they can gain higher revenue. As a result, Bob will have a hard time trying to fill his order. Thus, he has to either accept the loss in buying the full volume at higher prices, or divide the quantity into smaller batches and buy at different prices. Although the latter approach may seem better, it still incurs high fees and commissions paid to the exchange. Therefore, it is much more convenient for Bob to trade on a dark pool where the market impact will be minimal.

While dark pools provide a better trading platform for financial institutions, they have several issues. Most importantly, they hurt price discovery and put traders on other exchanges at a disadvantage. Furthermore, the lack of transparency could result in poor execution of trades or abuses such as front-running. Conflicts of interest are also a possibility since the operator could trade against pool clients. The U.S. Securities and Exchange Commission has found numerous violations and fined some dark pool operators [2–4]. Accordingly, several recent financial regulations, such as Europe's MiFID-II [5], call for limiting trades on dark pools. Interestingly, post enforcing MiFID-II, periodic auctions, which are considered regulation-compliant alternatives to dark pools, have witnessed a surge in the size of executed trading volumes.

In periodic auctions, the operator matches orders periodically, rather than continuously. Initially, traders submit orders privately to the operator. The submission phase ends at a random time. Next, the operator determines the market clearing price (MCP) and market clearing volume (MCV). Essentially, traders trust the operator to correctly calculate these values since they do not have access to the order book. To counter-balance this trust, regulators must audit the operator's work to reveal malicious behavior. However, the audit process is prohibitively expensive, and it might also be infrequent.

One way to remove trust requirements and reduce costly audits is to utilize the blockchain technology. Clearly, with the advent of the blockchain and Bitcoin [6], mutually distrusting parties can finally make transactions without relying on a trusted third party. Furthermore, complex types of transactions beyond simple payment transfers have quickly emerged due to the rich capabilities of smart contracts on blockchains such as Ethereum [7]. Smart contracts are pieces of data and code deployed on the blockchain. The consensus layer ensures that they execute precisely as their code dictates. Hence, a smart contract can act as a public trusted judge that resolves disputes and verifies the correctness of transactions.

The contributions in this paper are summarized below:

1. We build a protocol to prove that the committed values for a given vector of commitments are in descending order.
2. We utilize the above protocols to build a publicly verifiable and secrecy preserving periodic auction protocol.
3. We implemented a basic prototype to assess the protocol's performance, and released its source code on Github[1].

[1] `https://github.com/Anonymousub/PeriodicAuction`

2 Related Work

Thrope and Parkes [1] proposed a protocol for continuous double-sided auctions. Initially, each trader sends a price, a quantity, and a direction encrypted by the operator's public key of a homomorphic encryption scheme to a bulletin board. Then, the operator decrypts the orders and tries to match them. Once a match is found, the operator executes the matched orders and publishes them in history. The main drawback of this protocol is its heavy computation burden on the operator since it requires ranking all orders and generating proofs of correctness after the execution of every matched order.

Jutla [8] presented a secure five-party computation protocol for periodic auctions. A small number of brokers and a regulatory authority run the protocol. The auction starts with traders sending limit orders to brokers. Next, brokers run the protocol to find MCP and settle matched orders. In each round of this protocol, the regulatory authority must audit extensive computation to ensure the correctness of MCP, which renders the protocol impractical.

Galal and Youssef [9–11] proposed three constructions to build sealed-bid auctions on Ethereum. In the first construction, they utilize Pedersen commitment scheme and an interactive zero-knowledge range argument with high-cost transactions and limited scalability. The second construction uses zkSNARK, which improves the protocol scalability due to the constant proof size and verification cost. However, it requires a trusted setup to generate the proving and verification keys. Finally, the third construction utilizes Intel SGX as a trusted execution environment to determine the auction winner in a full privacy-preserving way with high performance. However, Intel SGX technology is not mature technology yet, and it faces multiple attacks that compromise its security.

Cartlidge *et al.* [12] utilized SCALE-MAMBA, a multi-party computation (MPC) framework, to emulate a trusted third party. The authors designed three constructions to assess the feasibility of using MPC in stock markets. They argue that it is not practical yet to run continuous double-sided auctions. On the other hand, the periodic auctions and volume matching constructions show promising results. Although this protocol provides strong secrecy, it requires a heavy preprocessing phase in addition to the inherent highly interactive communications between parties.

3 Preliminaries

3.1 Assumptions and Notations

Throughout the paper, an adversary $\mathcal{A}$ is a probabilistic interactive Turing Machine that runs in a polynomial time in the security parameter λ. Let $\mathbb{G}$ be a cyclic group of prime order p with generators g and h. Let $\mathbb{Z}_p^*$ denote $\mathbb{Z}_p \backslash \{0\}$, and $x \leftarrow_{\$} \mathbb{Z}_p^*$ denote uniform sampling of an element from $\mathbb{Z}_p^*$. We represent vectors by bold font, e.g. $\boldsymbol{a}$ is a vector with elements $(a_1, \ldots, a_n)$. Finally, let $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ denote a cryptographic collision resistant hash function.

3.2 ElGamal Encryption

We utilize ElGamal encryption scheme where messages are encoded in the exponent. It consists of the following algorithms:

- $(x, y) \leftarrow \mathcal{K}(\mathbb{G}, p, g)$: it samples a secret key $x \leftarrow_{\$} \mathbb{Z}_p$, and generates a public key $y = g^x$.
- $c \leftarrow \texttt{Enc}_y(m, r)$: it encrypts a message $m \in \mathbb{Z}_p$ using a blinding factor $r \leftarrow_{\$} \mathbb{Z}_p$ by the public key y, and outputs a ciphertext $c = (c_1, c_2) = (g^r, g^m y^r)$.
- $g^m \leftarrow \texttt{Dec}_x(c)$: it decrypts the ciphertext c by the secret key x, and outputs $g^m \leftarrow c_2 \cdot c_1^{-x}$. One needs to brute-force the discrete log of g^m to recover m which is affordable for small values (e.g., when m is a 32-bit value).

3.3 Pedersen Commitment

Pedersen commitment [13] is a non-interactive commitment scheme that has perfectly hiding and computationally binding properties. It consists of the following algorithms:

- $X \leftarrow \texttt{Com}(x, r)$: it commits to a message $x \in \mathbb{Z}_p$ using blinding factor $r \leftarrow_{\$} \mathbb{Z}_p$, and outputs $X = g^x h^r$.
- $\{\top/\bot\} \leftarrow \texttt{Vfy}(X, x, r)$: it verifies whether X commits to x with blinding factor r, and outputs $\top$ on success, otherwise, it outputs $\bot$.

Pedersen commitments are additively homomorphic. For instance, given the commitments $\texttt{Com}(x_1, r_1)$ and $\texttt{Com}(x_2, r_2)$, one can compute $\texttt{Com}(x_1 + x_2, r_1 + r_2) = \texttt{Com}(x_1, r_2)\texttt{Com}(x_2, r_2)$ without knowing the committed values.

3.4 Zero-Knowledge Proof of Knowledge

A zero-knowledge proof of knowledge is a protocol that allows a prover to convince a verifier that a certain statement holds without revealing any information beyond that fact. An argument is a proof which only holds if the prover is computationally bounded and certain computational hardness assumptions hold. We consider arguments consisting of three interactive algorithms $(\texttt{Setup}, \mathcal{P}, \mathcal{V})$ running in probabilistic polynomial time. The $\texttt{Setup}$ algorithm takes 1^λ as input, and produces a common reference string (CRS) denoted by Σ_{cce}. The transcripts produced by $\mathcal{P}$ and $\mathcal{V}$ when interacting on inputs s and t is denoted by $tr \leftarrow \langle \mathcal{P}(s), \mathcal{V}(t) \rangle$. We write $\langle \mathcal{P}(s), \mathcal{V}(t) \rangle = b$ to denote whether the verifier accepts $b = 1$, or rejects $b = 0$. Let $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$ be a ternary polynomial-time decidable relation. We call w a witness for a statement x if $(\sigma, x, w) \in \mathcal{R}$. Additionally, we define the CRS-dependent language

$$\mathcal{L}_\sigma = \{x | \exists w : (\sigma, x, w) \in \mathcal{R}\}$$

as the set of statements x that have a witness w in $\mathcal{R}$. Zero-knowledge arguments have the following three properties.

1. *Completeness*: the verifier will always accept a proof generated by an honest prover.

2. *Soundness*: the verifier will not accept a false proof except with a negligible probability.
3. *Zero-knowledge*: the verifier does not learn any information about the witness from the transcripts exchanged with the prover.

Fiat-Shamir heuristic [14] can transform the interactive triple $(\texttt{Setup}, \mathcal{P}, \mathcal{V})$ to non-interactive zero-knowledge (NIZK) proof in the random oracle model.

3.5 Zero-Knowledge Range Proof

A zero-knowledge range proof allows a prover to convince a verifier that a committed value falls within a given range. One of the recent constructions is Bulletproofs [15], which has a short logarithmic proof size $O(log(n))$ in the bit-width of the range. The proof generation and verification times scale linearly with n. More importantly, it does not require a trusted setup. In particular, given a commitment $X = g^x h^r \in \mathbb{G}$ for a witness $x \in \mathbb{Z}_p$, Bulletproofs allows a prover to generate the following NIZK argument:

$$\{(g, h \in \mathbb{G}, X; x, r) : X = g^x h^r \wedge x \in [0, 2^n - 1]\}$$

Bulletproofs allows the generation of an efficient aggregate argument for a vector of commitments. Specifically, given a vector of m commitments, the aggregate argument is smaller than the total size of m simple arguments. We refer to the protocol generating aggregate range arguments as $\texttt{BP} = (\texttt{Setup}, \mathcal{P}, \mathcal{V})$, which consists of the following probabilistic polynomial-time algorithms:

1. $\sigma \leftarrow \texttt{BP.Setup}(1^\lambda, n, m)$: it takes λ as the security parameter, n as the range bit-width, and m as the vector cardinality; and outputs Σ_{cce} as the CRS.
2. $\pi \leftarrow \texttt{BP}.\mathcal{P}(\sigma, \boldsymbol{X}, \boldsymbol{x}, \boldsymbol{r})$: it takes a vector of commitments $\boldsymbol{X}$ along with the opening vectors $\boldsymbol{x}$ and $\boldsymbol{r}$; and generates an argument π to prove

$$\{(\boldsymbol{X}; \boldsymbol{x}, \boldsymbol{r}) : X_i = g^{x_i} h^{r_i} \wedge x_i \in [0, 2^n - 1]\}$$

3. $\{\top/\bot\} \leftarrow \texttt{BP}.\mathcal{V}(\sigma, \boldsymbol{X}, \pi)$: it returns $\top$ if it accepts π; otherwise, it returns $\bot$.

3.6 Evaluator-Prover Model

The evaluator-prover (EP) model [16] provides a practical framework for secrecy preserving proofs of correctness. Involved parties secretly submit input values $(x_1, \ldots, x_n)$ to the EP entity. The EP privately computes a function $y = f(x_1, \ldots, x_n)$, outputs the value y, and generates a proof of the correctness. Parties accept the result on successful verification of the proof of correctness. The EP model is *secrecy preserving* if the proof does not reveal any information about the inputs beyond what is implied by the result.

Note that the EP model does not maintain *strong secrecy* [17], which mandates that the EP cannot disclose information about the inputs. However, the notion of *secrecy preserving* is still useful in the context of periodic auctions. More specifically, at the end of each round, information about the MCP and MCV are published, which gives more hints about the inputs. Hence, the main requirement here is to ensure that the operator cannot exploit this information to its advantage. In particular, the operator must not have access to the submitted orders until the end of the submission phase.

4 Basic Protocols

In this section, we present two zero-knowledge arguments protocols that are utilized to build the periodic auction.

4.1 Zero-Knowledge Proof of Consistent Commitment Encryption

We design an honest-verifier zero-knowledge Σ_{cce}, which is shown in Fig. 1, to prove that an ElGamal ciphertext hides the committed value of a Pedersen commitment. To motivate the need for this protocol, suppose that Alice has sent Bob a commitment $X = g^x h^r$ for an arbitrary value x. Later, she reveals the committed value x to Bob by encrypting it in a ciphertext $c = (c_1, c_2) = (g^r, g^x y^r)$ using Bob's public key y with the same blinding factor r. Alice wants to convince Carol that she has encrypted the committed value x in the ciphertext c using Bob's public key. More precisely, Alice wants to generate the following argument:$\{(g, h, c, X, y; x, r) : c_1 = g^r \wedge c_2 = g^x y^r \wedge X = g^x h^r\}$

$\mathcal{P}(c, X, y; x, r)$		$\mathcal{V}(c, X, y)$
$x', r' \leftarrow\$ \mathbb{Z}_p$		
$a_1 = g^{r'}, a_2 = g^{x'} y^{r'}$		
$A = g^{x'} h^{r'}$	$\xrightarrow{a_1, a_2, A}$	
$z_1 = x' + xe$	$\xleftarrow{e \leftarrow\$ \mathbb{Z}_p}$	
$z_2 = r' + re$	$\xrightarrow{z_1, z_2}$	Accept if and only if
		$g^{z_2} = a_1 c_1^e$
		$g^{z_1} y^{z_2} = a_2 c_2^e$
		$g^{z_1} h^{z_2} = A X^e$

Fig. 1: Σ_{cce} protocol

We utilize Fiat-Shamir heuristic to convert the Σ_{cce} protocol into NIZK argument by using a hash function to get the challenge $e \leftarrow H(\boldsymbol{c}, X, y, a_1, a_2, A)$. We define the following two $\mathcal{PPT}$ algorithms for this protocol:

1. $\pi \leftarrow \Sigma_{cce}.\mathcal{P}(c, X, y, x, r)$. It generates a proof π to prove that the ciphertext $\boldsymbol{c}$ is an encryption of the opening value x for the commitment X.
2. $\{0, 1\} \leftarrow \Sigma_{cce}.\mathcal{V}(c, X, \pi)$. It returns 1 if it has successfully verified the proof π for a ciphertext $\boldsymbol{c}$ and a commitment X; otherwise, it returns 0.

4.2 Zero-Knowledge Argument of Ordering

We build a protocol `ProveOrder` to prove that the committed values for a vector of Pedersen commitments are in descending order without revealing any infor-

mation beyond that fact, as shown in Fig. 2. More specifically, given a vector of commitments $\boldsymbol{X}$ of size $m+1$ to a vector of elements $\boldsymbol{x}$ in $[0, 2^n-1]$, we say that $\boldsymbol{x}$ is in descending order if the differences between successive elements x_i, x_{i+1} are non-negative values. Furthermore, since Pedersen commitments are additively homomorphic, one can compute the commitments vector $\hat{\boldsymbol{X}}$ to the differences between successive elements x_i, x_{i+1} given their commitments X_i, X_{i+1}. Now, we can utilize aggregate Bulletproofs to prove that the commitments in $\hat{\boldsymbol{X}}$ are commitments to elements in the range $[0, 2^n - 1]$. Note that we can also prove ascending order by simply reversing the elements in the vectors.

$\texttt{ProveOrder}.\mathcal{P}(\sigma, \boldsymbol{X}; \boldsymbol{x}, \boldsymbol{r})$		$\texttt{ProveOrder}.\mathcal{V}(\sigma, \boldsymbol{X})$
$\hat{\boldsymbol{X}} = (X_i \cdot X_{i+1}^{-1})_{i=1}^m$		
$\hat{\boldsymbol{x}} = (x_i - x_{i+1})_{i=1}^m$		
where $x_i' \in [0, 2^n - 1]$		
$\hat{\boldsymbol{r}} = (r_i - r_{i+1})_{i=1}^m$		
$\pi \leftarrow \texttt{BP}.\mathcal{P}(\sigma, \boldsymbol{X}', \boldsymbol{x}', \boldsymbol{r}')$		
	$\xrightarrow{\quad\pi\quad}$	
		$\hat{\boldsymbol{X}} = (X_i \cdot X_{i+1}^{-1})_{i=1}^m$
		Accept if and only if:
		$\texttt{BP}.\mathcal{V}(\sigma, \hat{\boldsymbol{X}}, \pi) = 1$

Fig. 2: `ProveOrder` Protocol

By default, this protocol inherits the completeness and zero-knowledge properties of Bulletproofs [15]. To ensure the soundness, we have a condition on the value of 2^n. Specifically, as the operation $x_i - x_{i+1}$ is carried out in $\mathbb{Z}_p$, then, the condition $2^n < \frac{p}{2}$ must hold to ensure that negative differences do not fall in the range $[0, 2^n - 1]$.

It is worth mentioning that the implementation of Fiat-Shamir heuristic can compromise the security of Σ_{cce} and `ProveOrder` protocols. More precisely, these protocols are susceptible to *replay* attacks by the adversary when they are used with blockchain. For example, the adversary can replay an arbitrary trader's proof to the smart contract without knowing any witness, yet her proof will be successfully accepted. To prevent this attack, we include the *address* of the transaction sender as one of the inputs to the hash function that computes the verifier challenges. Consequently, the adversary's proof will be rejected because the verifier challenges computed by the smart contract will be different from those computed for the replayed proof.

5 Periodic Auction Protocol Design

5.1 System Model

In this protocol, there are three entities, namely, traders, an operator, and a smart contract. The operator and traders interact indirectly through the smart contract using their accounts on the blockchain.

1. Traders are the buyers and sellers who want to exchange their assets through the auction.
2. An operator is the EP entity that privately receives orders and evaluates the MCP and MCV, and proves their correctness to the smart contract.
3. A smart contract publicly verifies the zero-knowledge proofs submitted by traders and the operator, as well as serves as a secure bulletin-board.

5.2 High-Level Flow of the Protocol

The operator deploys the smart contract and initializes it by a set of public parameters. In Appendix A, we show the pseudocode for the smart contract. Each operation performed by the traders or the operator results in a piece of data and zero-knowledge proof, which will be submitted to the smart contract. The smart contract verifies the proof, and upon success, it stores the associated data. A single round of the periodic auction protocol consists of the following three phases:

1. Traders commit to their orders, and utilize Bulletproofs to generate an aggregate range proof.
2. Traders encrypt their orders by the operator's public key, and utilize Σ_{cce} protocol to prove the consistency between ciphertext and commitments.
3. The operator does the following:
   (a) Access price and quantity values in orders.
   (b) Determine the MCP and MCV.
   (c) Generate proof of correctness for MCP and MCV.

6 Protocol Design

6.1 Smart Contract Deployment and Parameters Setup

The protocol starts by the operator Alice generating the public parameter σ by running the setup algorithm of Bulletproofs for a security parameter λ, a bit-width n, and number of commitments m. Then, she generates a key-pair x, y as the secret and public keys for ElGamal encryption scheme, respectively. Additionally, she defines the time-window of each phase by the vector $\boldsymbol{t}$.

$$\begin{aligned} \sigma &\leftarrow \texttt{BP.Setup}(1^\lambda, n, m) \\ x, y &\leftarrow \mathcal{K}(\mathbb{G}, p, g), \\ \boldsymbol{t} &= (t_1, t_2, t_3) \end{aligned}$$

Next, she deploys the smart contract and initializes it with the parameters $(\sigma, y, \boldsymbol{t})$ and locks a fixed collateral deposit $\mathcal{D}$.

6.2 Phase One: Submission of Orders

Traders submit their orders before the block-height t_1. For example, a trader Bob wants to buy v units of the auctioned asset at a price u. He creates his order as follows:

$$\begin{aligned}
&\boldsymbol{r} \leftarrow_{\$} \mathbb{Z}_p^2 \\
&U \leftarrow Com(u, r_1) \\
&V \leftarrow Com(v, r_2) \\
&\pi \leftarrow \texttt{BP}.\mathcal{P}(\sigma, (U, V), (u, v), \boldsymbol{r})
\end{aligned}$$

First, he creates the commitments U and V for the price and quantity, respectively. Subsequently, the trader generates an aggregate range proof π to assert that the price and quantity values are within the range, i.e. $u, v \in [0, 2^n - 1]$. It is worth mentioning that in the prototype, this phase uses a different Bulletproof setup where $m = 2$ since there are two commitments only. Finally, he sends a transaction that includes the parameters $(\texttt{dir}, U, V, \pi)$ where `dir` indicates whether the order is a bid or an offer.

Upon receiving the transaction, the smart contract checks whether the current block-height is less than t_1, and the transaction has a collateral deposit $\mathcal{D}$. Then, it verifies the aggregate range proof π for the commitments U and V. Finally, it stores the commitments in either the list of `Bids` or the list of `Offers` based on the value of `dir`.

It is worth mentioning that front-running has a little impact in periodic auctions in contrast to continuous mainly because orders will be settled at a single MCP regardless of orders sequence. Still, this protocol provides protection against front-running for three main reasons. First, the commitments U and V are perfectly hiding. Second, the aggregate range proof π is zero-knowledge, hence, π does not reveal any information about the witness u and v beyond the fact that they are in range $[0, 2^n - 1]$. Third, there is an idle period between the first and second phases to consider the possibility of revealing orders on minor blockchain forks that will be discarded.

6.3 Phase Two: Revealing Orders

Traders utilize ElGamal encryption to reveal their orders to Alice before the block-height t_2. Therefore, Bob retrieves Alice's public key y from the smart contract and encrypts the opening values (u, r_1) and (v, r_2) as follows:

$$\begin{aligned}
&c_u \leftarrow \texttt{Enc}_y(u, r_1),\ \pi_u \leftarrow \Sigma_{cce}.\mathcal{P}(c_u, U, y, u, r_1) \\
&c_v \leftarrow \texttt{Enc}_y(v, r_2),\ \pi_v \leftarrow \Sigma_{cce}.\mathcal{P}(c_v, V, y, v, r_2)
\end{aligned}$$

Then, he utilizes Σ_{cce} protocol to generate the proofs π_u and π_v to prove that the ciphertext c_u and c_v encrypt the opening values (u, r_1) and (v, r_2) of commitments U and V using Alice's public key y, respectively. Subsequently, he sends a transaction which includes the parameters (c_u, c_v, π_u, π_v).

Initially, the smart contract checks if the transaction is sent within the right time window between t_1 and t_2. Then, it searches for the commitments (U, V) corresponding to transaction `sender` in either `Bids` or `Offers`. Subsequently,

it verifies the proofs π_u and π_v. Alice can monitor the transactions submitted to the smart contract during this phase to recover the ciphertext c_u and c_v. In practice, Alice can efficiently retrieve the ciphertext by listening to *events* triggered on the smart contract.

6.4 Phase Three: Matching Orders

At the beginning of this phase, Alice instructs the smart contract to find unrevealed orders, remove them, and penalize their owners. Accordingly, she has access to the price and quantity values of revealed orders. She performs the following tasks to determine the MCP and MCV before block-height t_3:

1. Sort the bids descendingly and the offers ascendingly by price.
2. Compute the cumulative quantity in bids and offers.
3. Finds the MCP that clears the highest cumulative quantity, i.e. MCV.
4. Send the MCP and MCV along with proofs of correctness to the smart contract.

She can generate proof of correctness by creating an order with the MCP and MCV values. Then, she inserts that order in the sorted lists of bids and offers consisting of prices and cumulative quantities. Finally, she utilizes `ProveOrder` protocol to prove the sort on price and cumulative quantity commitments. Note that cumulative quantity commitments can be easily computed since Pedersen commitments are additively homomorphic.

Let $\boldsymbol{B}$ and $\boldsymbol{S}$ denote the lists of bids and offers where the numbers of orders in each list are M and N, respectively. Each order in $\boldsymbol{B}$ and $\boldsymbol{S}$ is encoded as a tuple $(U, V, V_c, u, r_1, v, r_2, v_c, r_c)$ of price, quantity, and cumulative quantity commitments and their opening values. Note that, at the beginning, V_c, v_c, r_c are empty. Alice performs the first task as follows:

$$\texttt{Sort}(\boldsymbol{B}, \texttt{DESCENDING}),\ \texttt{Sort}(\boldsymbol{S}, \texttt{ASCENDING})$$

The `Sort` function sorts the elements in the input list according to the specified criteria on the price values. For example, the elements in $\boldsymbol{B}$ and $\boldsymbol{S}$ are relocated such that:

$$\forall i \in [1, M-1],\ B_i.u > B_{i+1}.u$$
$$\forall j \in [1, N-1],\ S_j.u < S_{j+1}.u$$

Next, for each order in $\boldsymbol{B}$ and $\boldsymbol{S}$, she computes the cumulative quantities as:

$$\forall i \in [1, M],\ B_i.(V_c, v_c, r_c) \leftarrow \boldsymbol{B}.(\prod_{k=1}^{i} V_k, \sum_{k=1}^{i} v_k, \sum_{k=1}^{i} r_{2,k})$$
$$\forall j \in [1, N],\ S_j.(V_c, v_c, r_c) \leftarrow \boldsymbol{S}.(\prod_{k=1}^{j} V_k, \sum_{k=1}^{j} v_k, \sum_{k=1}^{j} r_{2,k})$$

Subsequently, she finds the intersection range between prices in $\boldsymbol{B}$ and $\boldsymbol{S}$. Then, for this range, take the middle point as MCP denoted by p, and the lowest

cumulative quantity as MCV denoted by l. After that, she generates an order $\mathcal{M}$ with commitments to p and l as follows:

$$\begin{aligned} P &\leftarrow \texttt{Com}(p, 0),\ L \leftarrow \texttt{Com}(l, 0) \\ \mathcal{M} &= (P, 0, L, p, 0, 0, 0, l, 0) \end{aligned}$$

Note that, the blinding values in commitments of $\mathcal{M}$ are set to zero as we want the commitments to be binding only. Moreover, p and l will be posted on the smart contract eventually, we just need them in commitment form to be utilized in the `ProveOrder` protocol. Finally, she inserts $\mathcal{M}$ in both $\boldsymbol{B}$ and $\boldsymbol{S}$ while preserving the ordering:

$$\boldsymbol{B}.\texttt{Insert}(\mathcal{M}),\ \boldsymbol{S}.\texttt{Insert}(\mathcal{M})$$

Now, Alice utilizes `ProveOrder` protocol to prove the correctness of MCP p and MCV l as follows:

$$\begin{aligned} \pi_1 &\leftarrow \texttt{ProveOrder}.\mathcal{P}(\boldsymbol{B}.(\boldsymbol{U}, \boldsymbol{u}, \boldsymbol{r_1}), \texttt{DESCEND}) \\ \pi_2 &\leftarrow \texttt{ProveOrder}.\mathcal{P}(\boldsymbol{B}.(\boldsymbol{V_c}, \boldsymbol{v_c}, \boldsymbol{r_c}), \texttt{ASCEND}) \\ \pi_3 &\leftarrow \texttt{ProveOrder}.\mathcal{P}(\boldsymbol{S}.(\boldsymbol{U}, \boldsymbol{u}, \boldsymbol{r_1}), \texttt{ASCEND}) \\ \pi_4 &\leftarrow \texttt{ProveOrder}.\mathcal{P}(\boldsymbol{S}.(\boldsymbol{V_c}, \boldsymbol{v_c}, \boldsymbol{r_c}), \texttt{ASCEND}) \\ \boldsymbol{\pi} &= (\pi_1, \pi_2, \pi_3, \pi_4) \end{aligned}$$

In the smart contract, the indices of orders in `Bids` and `Offers` depend entirely on their arrival time. Hence, Alice creates two positioning vectors $\boldsymbol{\chi}$ and $\boldsymbol{\gamma}$ that will be used by the smart contract as proxies to access `Bids` and `Offers` in their sorted order, respectively. Finally, Alice sends a transaction which contains the parameters $(p, l, \boldsymbol{\chi}, \boldsymbol{\gamma}, \boldsymbol{\pi})$ to the smart contract.

The smart contract checks that the transaction is sent by Alice between block-heights t_2 and t_3. Then, it checks whether $p, l \in [0, 2^n - 1]$. After that, it appends the order $\mathcal{M}$ in `Bids` and `Offers`. Finally, it verifies the proofs $\boldsymbol{\pi}$ before accepting and storing p and l.

Upon successful verification, the smart contract refunds the collateral deposits to Alice and owners of unsettled orders. On the other hand, the smart contract keeps the collateral deposits of owners of executed orders locked for the settlement of the assets exchange phase off-chain. Conversely, if the verification was not successful or the Alice failed to send the proofs $\boldsymbol{\pi}$ before block-height t_3, then the smart contract slashes her deposit and refunds the traders.

7 Performance Evaluation

In this section, we evaluate the performance measurement of the proposed protocol and assess its feasibility.

7.1 Evaluation

We report the measurements of the two main building blocks that constitutes the periodic auction protocol, namely, Σ_{cce} and `ProveOrder` protocols n Table 1. The proof size is measured by the number of elements in $\mathbb{G}$ and $\mathbb{Z}_p$. For

the verifier, we report the number of elliptic curve operations required to verify proofs.

Table 1: Performance of `ProveOrder` and Σ_{cce} protocol.

Performance	#	`ProveOrder`	Σ_{cce} Protocol
Proof size	$\mathbb{G}$	$2(log_2(n) + log_2(m)) + 4$	3
	$\mathbb{Z}_p$	5	2
Verifier operations	`mul`	$11 + 2n + m$	8
	`add`	$7 + 2n + m$	5

In Ethereum, the point addition and point multiplication operations cost 150 and 6000 gas, respectively. Hence, we can measure the transaction gas cost of verifying the submitted proofs. In Fig.3, we report the performance measurements: proof size, prover time, and gas cost of proof verification by the smart contract with respect to the total number of orders for the transactions: `SubmitOrder`, `RevealOrder`, and `ClearMarket`. Obviously, the transaction `SubmitOrder` and `RevealOrder` have constant measurements as opposed to `ClearMarket` transaction which scale linearly with the number of orders.

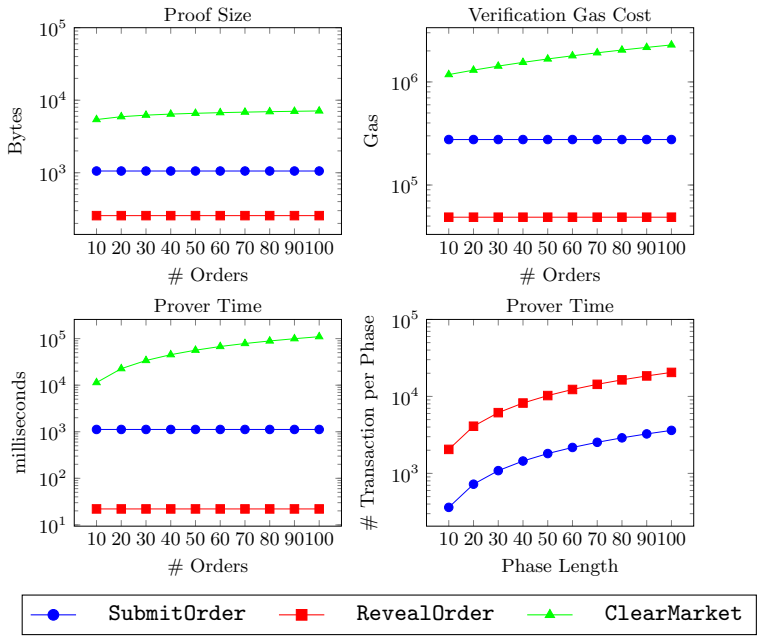

Fig. 3: Performance measurements of the periodic auction protocol

The current block gas limit on Ethereum is roughly 10M gas. Hence, we can estimate the number of transactions that fit in a single block. More importantly, we can estimate the theoretical number of `SubmitOrder` and `RevealOrder` transactions that the smart contract can receive during the first and second phases for different phase lengths as shown in Fig 3. The `SubmitOrder` transaction incurs the cost of verifying `ProveOrder` proof where $n = 16$ and $m = 2$. Similarly, the `RevealOrder` transaction incurs the cost of verifying Σ_{cce} proof. Accordingly, the transaction cost of `SubmitOrder` and `RevealOrder` are roughly 276150 and 48750 gas, respectively. Note that, in practice, the gas cost for each transaction is higher since there are additional operations involving data access and control flow.

Furthermore, we can estimate the highest number of orders that can be processed by a single `ClearMarket` transaction before exceeding the 10M gas block limit. Typically, the `ClearMarket` transaction requires verification of two `ProveOrder` proofs for M bids and two `ProveOrder` proofs for N offers. For convenience, assume that we have an equal number of bids and offers $M = N$, hence, the `ClearMarket` transaction incurs the verification cost of four `ProveOrder` proofs of M commitments. Accordingly, the `ClearMarket` transaction can theoretically process up to ≈ 728 orders before exceeding the block gas limit. Certainly, in practice, this number is lower due to the gas cost associated with operations other than proof verification.

8 Conclusion

We presented publicly verifiable secrecy preserving periodic auction protocol. The protocol depends on two zero-knowledge proofs, namely, proof of consistent commitment encryption and proof of ordering. Furthermore, we implemented a prototype and evaluated its performance to assess its feasibility. Based on the result, we believe that the periodic auction protocol is a feasible and secure alternative to dark pools.

References

1. Thorpe, C., Parkes, D.C.: Cryptographic securities exchanges. In: International Conference on Financial Cryptography and Data Security. pp. 163–178. Springer (2007)
2. Sec charges citigroup for dark pool misrepresentations. `https://www.sec.gov/news/press-release/2018-193` (2018)
3. Sec charges itg with misleading dark pool subscribers. `https://www.sec.gov/news/press-release/2018-256` (2018)
4. Barclays, credit suisse charged with dark pool violations. `https://www.sec.gov/news/pressrelease/2016-16.html` (2016)
5. Markets in financial instruments directive ii. `https://www.esma.europa.eu/policy-rules/mifid-ii-and-mifir` (2018)
6. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
7. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper 151, 1–32 (2014)
8. Jutla, C.S.: Upending stock market structure using secure multi-party computation. IACR Cryptology ePrint Archive 2015, 550 (2015)

9. Galal, H.S., Youssef, A.M.: Verifiable sealed-bid auction on the Ethereum blockchain. In: Financial Cryptography and Data Security. pp. 265–278. Springer (2019)
10. Galal, H.S., Youssef, A.M.: Succinctly verifiable sealed-bid auction smart contract. In: Data Privacy Management, Cryptocurrencies and Blockchain Technology. pp. 3–19. Springer (2018)
11. Galal, H.S., Youssef, A.M.: Trustee: Full privacy preserving vickrey auction on top of Ethereum. In: Financial Cryptography and Data Security Workshop on Trusted Smart Contracts, to appear. Springer (2019)
12. Cartlidge, J., Smart, N.P., Talibi Alaoui, Y.: MPC joins the dark side. In: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security. pp. 148–159. ACM (2019)
13. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual International Cryptology Conference. pp. 129–140. Springer (1991)
14. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM conference on Computer and communications security. pp. 62–73 (1993)
15. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315–334. IEEE (2018)
16. Micali, S., Rabin, M.O.: Cryptography miracles, secure auctions, matching problem verification. Communications of the ACM 57(2), 85–93 (2014)
17. Parkes, D.C., Thorpe, C., Li, W.: Achieving trust without disclosure: Dark pools and a role for secrecy-preserving verification. In: Proceedings of the Third Conference on Auctions, Market Mechanisms and Their Applications (2015)

Appendix A Smart Contract Pseudocode

```
function SubmitOrder(dir, U, V, π)
    require(msg.value = D)
    require(msg.blockNumber < t_1)
    require(BP.V(σ_2, (U, V), π))
    if dir = BUY then
        Bids[msg.sender] ← (U, V)
    else
        Offers[msg.sender] ← (U, V)
    end if
end function
function RevealOrder(c_u, c_v, π_u, π_v)
    require(t_1 < msg.blockNumber < t_2)
    (U, V) ← FindOrder(msg.sender)
    require((U, V) ≠ NULL)
    require(Σ.V(c_u, U, y, π_u))
    require(Σ.V(c_v, V, y, π_v))
    emit RevealEvent(msg.sender, c_u, c_v)
```

```
end function
function ClearMarket(p, l, χ, γ, π)
    require(t_2 < msg.blockNumber < t_3)
    require(msg.sender = Alice)
    RemoveUnrevealedOrders()
    require(p ∈ [0, 2^n − 1] ∧ l ∈ [0, 2^n − 1])
    O = (g^p, g^l)
    Bids[Alice] ← O
    Offers[Alice] ← O
    Relocate(Bids, χ)
    Relocate(Offers, γ)
    U_1 ← Bids.U
    V_1 ← CumulativeQuantity(Bids.V)
    U_2 ← Offers.U
    V_2 ← CumulativeQuantity(Offers.V)
    require(ProveOrder.V(σ, U_1, π_1))
    require(ProveOrder.V(σ, V_1, π_2))
    require(ProveOrder.V(σ, U_2, π_3, ASCEND))
    require(ProveOrder.V(σ, V_2, π_4))
    Store p, l
end function
```