A Blockchain-based Iterative Double Auction Protocol Using Multiparty State Channels

TRUC D. T. NGUYEN and MY T. THAI, University of Florida

Although the iterative double auction has been widely used in many different applications, one of the major problems in its current implementations is that they rely on a trusted third party to handle the auction process. This imposes the risk of single point of failures, monopoly, and bribery. In this article, we aim to tackle this problem by proposing a novel decentralized and trustless framework for iterative double auction based on blockchain. Our design adopts the smart contract and state channel technologies to enable a double auction process among parties that do not need to trust each other, while minimizing the blockchain transactions. In specific, we propose an extension to the original concept of state channels that can support multiparty computation. Then, we provide a formal development of the proposed framework and prove the security of our design against adversaries. Finally, we develop a proof-of-concept implementation of our framework using Elixir and Solidity, on which we conduct various experiments to demonstrate its feasibility and practicality.

CCS Concepts: • **Security and privacy** → **Software and application security**; **Distributed systems security**; • **Computer systems organization** → *Peer-to-peer architectures*;

Additional Key Words and Phrases: Iterative double auction, blockchain, state channel, trustless

ACM Reference format:
Truc D. T. Nguyen and My T. Thai. 2021. A Blockchain-based Iterative Double Auction Protocol Using Multiparty State Channels. *ACM Trans. Internet Technol.* 21, 2, Article 39 (March 2021), 22 pages.
https://doi.org/10.1145/3389249

39

1 INTRODUCTION

Blockchain, the technology that underpins the great success of Bitcoin [18] and various other cryptocurrencies, has incredibly emerged as a trending research topic in both academic institutes and industries associations in recent years. With great potential and benefits, the blockchain technology promises a new decentralized platform for the economy such that the possibility of censorship, monopoly, and single point of failures can be eliminated [26]. The technology, in its simplest form, can be seen as a decentralized database or digital ledger that contains append-only data blocks where each block comprises valid transactions, timestamp, and the cryptographic hash of the previous block. By design, a blockchain system is managed by nodes in a peer-to-peer network and operates efficiently in a decentralized fashion without the need of a central authority. Specifically, it enables a trustless network where participants of the system can settle transactions without having to trust each other. With the aid of the smart contracts technology, a blockchain system

Authors' addresses: T. D. T. Nguyen and M. T. Thai (corresponding author), University of Florida, Gainesville, Florida, 32611; emails: truc.nguyen@ufl.edu, mythai@cise.ufl.edu.

1533-5399/2021/03-ART39 $15.00
https://doi.org/10.1145/3389249

can enable a wide range of applications that go beyond financial transactions [29]. In the context of blockchain, *smart contracts* are defined as self-executing and self-enforcing programs that are stored on chain. They are intended to facilitate and verify the execution of terms and conditions of a contract within the blockchain system. By employing this technology, applications that previously require a trusted intermediary can now operate in a decentralized manner while achieving the same functionality and certainty. For that reason, blockchain and smart contracts together have inspired many decentralized applications and stimulated scientific research in diverse domains [1, 2, 4, 14, 19, 20, 22].

An auction is a market institution in which traders or parties submit bids that can be an offer to buy or sell at a given price [9]. A market can enable only buyers, only sellers, or both to make offers. In the latter case, it is referred as a two-sided or *double auction*. A double auction process can be one-shot or iterative (repeated). The difference between them is that an iterative double auction process has multiple, instead of one, iterations [21]. In each iteration, each party submits a bid illustrating the selling/buying price and supplying/demanding units of resource. This process goes on until the market reaches Nash Equilibrium (NE). In practice, the iterative double auction has been widely used for decentralized resource allocations among rational traders, especially for divisible resources, such as energy trading [8, 14], mobile data offloading [11], or resource allocation in autonomous networks [12]. In these applications, in each iteration, players submit their individual bids, respectively, to an auctioneer who later calculates to determine the resource allocation with respect to the submitted bids. However, current implementations of double auction systems require a centralized and trusted auctioneer to regulate the auction process. This results in the risk of single point of failures, monopoly, and bribery.

Although many research work have tried to develop a trading system combining the iterative double auction and blockchain [14, 28], nonetheless, they still need a trusted third-party to handle the auction process. In this work, we leverage blockchain and smart contracts to propose a general framework for iterative double auction that is completely *decentralized* and *trustless*. We argue that, due to the low throughput of blockchain, a naive and straightforward adoption of blockchain smart contracts to eliminate the trusted third-party would result in significantly high latency and transaction fees. To overcome this problem, we adopt the *state channel* technology [6] with extension to support computation among more than two parties so we can enable efficient off-chain execution of decentralized applications without changing the trust assumption. Specifically, we propose a double auction framework operating through a *state channel* that can be coupled to existing double auction algorithms to run the auction process efficiently.

To demonstrate the feasibility and practicality of our solution, we develop a proof-of-concept implementation of the proposed solution. This proof-of-concept is built based on our novel development framework that can be used to deploy any distributed protocols using blockchain and state channels, which is also introduced in this article. Based on the proof-of-concept, we conduct experiments and measure the performance in various aspects; the results suggest that our proposed solution can carry out a double auction process on blockchain that is both time- and cost-saving with a relatively small overhead.

Contributions. We summarize our contributions as follows:

- We introduce a novel decentralized and trustless framework for iterative double auction based on blockchain. With this framework, existing double auction algorithms can efficiently run on a blockchain network without suffering the high latency and fees of on-chain transactions.
- We enhance the state channel technology, which is currently limited to two participants, to support multiparty computation. Based on this enhancement, we present a formal

development of our solution, in which we develop a Universally Composable (UC)-style model [3] for the double auction protocol and prove the security properties of our design using the simulation-based UC framework.
- To validate our proposed solution, we develop a proof-of-concept implementation of the framework to demonstrate its feasibility. For this implementation, we also introduce a novel development framework for distributed computing using blockchain and state channel. The framework is developed using the Elixir programming language and the system can be deployed on an Ethereum blockchain [7].

Organization. The remainder of this article is organized as follows: The related work is summarized in Section 2. Section 3 discusses the integration of Blockchain and double auction to establish some security goals for the system. We first present a straw-man design and then provide a high-level view of our framework. Then, we provide formal security definitions and specifications with a detailed security analysis of our framework in Section 4. Section 5 presents the proof-of-concept implementation along with the system evaluation. Section 6 concludes the article.

2 RELATED WORK

Double auction based on blockchain. As blockchain is an emerging technology, there have been many research works addressing double auction with blockchain. Recently, Thakur et al. [27] published a paper on distributed double auction for peer-to peer-energy trading. The authors use the McAfee mechanism to process the double auction on smart contracts. In Reference [17], the authors presented BlockCloud, which is a service-centric blockchain architecture that supports double auction. The auction model in this work uses a trade-reduction mechanism. However, the double auction mechanism in these work is one-shot and is only applicable to single-unit demands. For applications such as energy or wireless spectrum allocation, these models greatly limit users' capability to utilize the products [25].

In References [14] and [28], the authors propose blockchain-based energy trading using double auction. The auction mechanism is implemented as an iterative process that can be used for divisible goods. Although the system presented in these papers employs blockchain, the double auction process is still facilitated by a central entity. The blockchain is only used for settling payments. Our work is fundamentally different, as we aim to design a framework that can regulate the iterative double auction process in a decentralized and trustless fashion.

State channel. Although there have been many research efforts on payment channels [5, 15, 16], the concept of state channel has only emerged in recent years. As payment channel is limited to payment transactions, state channel is a generalization of payment channel in which users can execute complex smart contracts off-chain while still maintaining the trustless property. Instead of executing the contracts on-chain and having all the transactions validated by every blockchain nodes, the state channel technologies allow users to update the states off-chain with the option to raise disputes on-chain. Thus, it offers an efficient solution to address the scalability issue of blockchain systems. Dziembowski et al. [6] is the first work that presents formal specifications of state channels. However, the authors did not develop any proof-of-concept implementation to validate their protocol.

One problem with the original concept of state channels is that they only support execution between two parties, which is not applicable to our scenario, since we are dealing with a system of multiple parties. For that reason, based on the work in Reference [6], we extend the state channel technology to a multiparty state channel that can support computation among multiple users. Our extension also provides additional functionalities to handle dynamic changes of system participants. Based on this multiparty state channel, we design our double auction framework and analyze its security properties in the UC model.

3 DOUBLE AUCTION WITH BLOCKCHAIN

In this section, we formally define the double auction model that is used in this work. Moreover, beginning with a straw-man design, we present the high-level design of our framework and its security goals.

3.1 Auction Model

We consider a set of parties that are connected to a blockchain network. We divide the set of parties into a set $\mathcal{B}$ of buyers who require resources from a set $\mathcal{S}$ of sellers. These two sets are disjoint. The demand of a buyer $i \in \mathcal{B}$ is denoted as d_i, and the supply of a seller $j \in \mathcal{S}$ is denoted as s_j. In this work, we adopt the auction model proposed in Reference [30], which elicits hidden information about parties to maximize social welfare, as a general iterative double auction process that converges to a Nash Equilibrium (NE).

A bid profile of a buyer $i \in \mathcal{B}$ is denoted as $b_i = (\beta_i, x_i)$ where β_i is the buying price per unit of resource and x_i is the amount of resource that i wants to buy. Likewise, a bid profile of a seller $j \in \mathcal{S}$ is denoted as $b_j = (\alpha_j, y_j)$ where α_j is the selling price per unit of resource and y_j is the amount of resource that j wants to supply.

The auction process consists of multiple iterations. At an iteration k, the buyers and sellers submit their bid profiles $b_i^{(k)}$ and $b_j^{(k)}$, respectively, to the auctioneer. Then, a double auction algorithm will be used to determine the best response $b_i^{(k+1)}$ and $b_j^{(k+1)}$ for the next iteration. This process goes on until the auction reaches NE, at which the bid demand and supply (x_i, y_j) will converge to an optimal value that maximizes the social welfare. For an example of such algorithm, refer to Reference [30]. The pseudo code for a centralized auctioneer is presented in Algorithm 1.

ALGORITHM 1

$k \leftarrow 1$
while not NE **do**
  Receive bid profiles $b_i^{(k)}$ and $b_j^{(k)}$ from buyers and sellers
  Compute best responses $b_i^{(k+1)}$ and $b_j^{(k+j)}$
  Send $b_i^{(k+1)}$ and $b_j^{(k+1)}$ back to sellers and buyers
  k ← k + 1
end while

3.2 Straw-man Design

In this section, we present a design of the trading system. The trading mechanism must meet the following requirements:

(a) Decentralized: the auction process is not facilitated by any central middleman
(b) Trustless: the parties do not have to trust each other.
(c) Non-Cancellation: parties may attempt to prematurely abort from the protocol to avoid payments. These malicious parties must be financially penalized.

Based on the requirements, we will first show a straw-man design of the system that has some deficiencies in terms of latency and high transaction fee. Then, we will propose a trading system using state channels to address those problems.

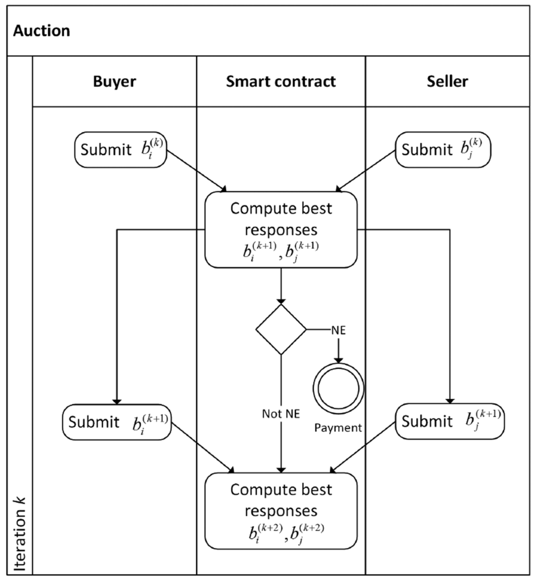


Fig. 1. Auction phase.

In this system, we deploy a smart contract to the blockchain to regulate the trading process. Prior to placing any bid, all parties must make a deposit to the smart contract. If a party tries to cheat by prematurely aborting the trading process, he or she will lose that deposit and the remaining parties will receive compensation. Therefore, the deposit deters parties from cheating. At the end of the trading process, these deposits will be returned to the parties.

In this straw-man design, the auction process will be executed on-chain, that is, the smart contract will act as an auctioneer and thus will execute Algorithm 1. As the auction process consists of multiple iterations, the system will follow the activity diagram in Figure 1 at each iteration.

At an iteration k, all buyers and sellers submit their bids $b_i^{(k)}$ and $b_j^{(k)}$, respectively, to the smart contract. To avoid unresponsiveness, a timeout is set for collecting bids. Should any parties fail to meet this deadline, the system considers that they aborted the process.

The smart contract then determines the best response $b_i^{(k+1)}$ and $b_j^{(k+j)}$ for buyers and sellers, respectively, until the trading system reaches NE. This design, however, has two main disadvantages:

(1) Transaction latency: Each message exchanged between parties and the smart contract is treated as a blockchain transaction, which takes time to get committed.
(2) High computational complexity on the smart contract means that the blockchain will require high transaction fees.

These disadvantages come from the fact that a transaction in a blockchain network has to be confirmed by all the validators. In other words, with this design, the buyers and sellers are having the entire blockchain to process their auction, which is very inefficient and, thus, not practical. In the following section, we will propose another design to overcome these issues.

3.3 Blockchain with Multiparty State Channels

As the double auction process involves multiple iterations, state channel [6] is a proper solution to address the deficiencies of the straw-man design. Instead of processing the auction on-chain,

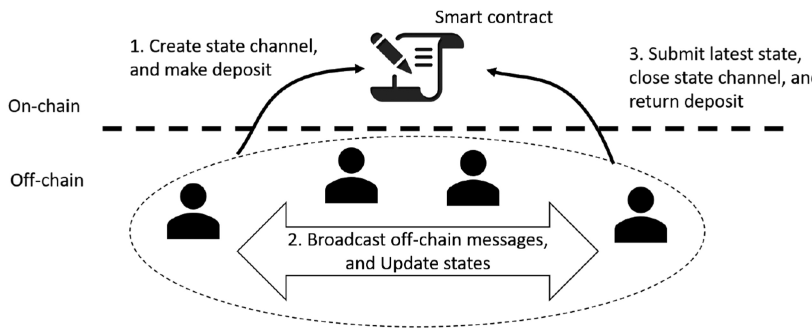


Fig. 2. Multiparty state channel: overview.

the parties will be able to update the states of the auction off-chain. Whenever something goes wrong (e.g., some parties try to cheat), the users always have the option of referring back to the blockchain for the certainty of on-chain transactions. Since the original concept of state channel only supports the computation between two parties, in this work, we propose an extension to support multiparty computation that can work with the double auction. This section illustrates a high-level view of how we can conduct a double auction using the multiparty state channel.

In the same manner as the straw-man design, the parties deploy a smart contract to the blockchain. However, this smart contract does not regulate the auction process, but instead acts as a judge to resolve disputes. The parties must also make a deposit to this contract prior to the auction. Figure 2 illustrates the overview of the operations in the multiparty state channel.

After deploying the smart contract, the parties can now begin the auction process in a *state channel*. At each iteration k of the auction, we define two operations: (1) collecting bids and (2) determining the best responses. Denoting the set of parties as $\mathcal{P} = \mathcal{B} \cup \mathcal{S} = \{p_1, p_2, \ldots, p_n\}$, in the first operation, each party broadcasts a blockchain transaction containing its bid $b_{p_i}^{(k)}$ to all other parties. Note that this transaction is a *valid* blockchain transaction and it is only broadcasted locally among the parties. Upon receiving that transaction, each party has to verify the transaction's signature. After this operation, each party now has the bid profiles of all other parties.

Then, we move to the second operation of determining the best response. A party will be chosen to compute the best responses; in fact, it does not matter who will execute this computation, because the results will later be verified by other parties. Therefore, this party can be chosen randomly or based on the amount of deposit to the smart contract. Let p_k be the one who carries out the computation at iteration k, G_k be the result that consists of the best response $b_{p_i}^{(k+1)}$ for each party p_i, and p_k will broadcast a blockchain transaction containing G_k to all other parties. Upon receiving this transaction, each party has to verify the result G_k, then signs it and broadcasts another transaction containing G_k to all other parties. This action means that the party agrees with G_k. After this step, each party will have G_k together with the signatures of all parties.

When the auction process reaches NE, a party will send the final G_k together with all the signatures to the smart contract. The smart contract then verifies the G_k and if there is no dispute, the state channel is closed. Finally, the payment will be processed on-chain and the smart contract refunds the initial deposit to all parties. The entire process is summarized in the sequence diagram in Figure 3.

As can be seen, the blockchain is invoked only two times and thus saves tons of transaction fees compared to the straw-man design. Moreover, as the transactions are not sent to the blockchain, the latency is only limited by the communication network among the parties. We can also see that the bid profiles are only known among the involving parties, not to the entire blockchain,

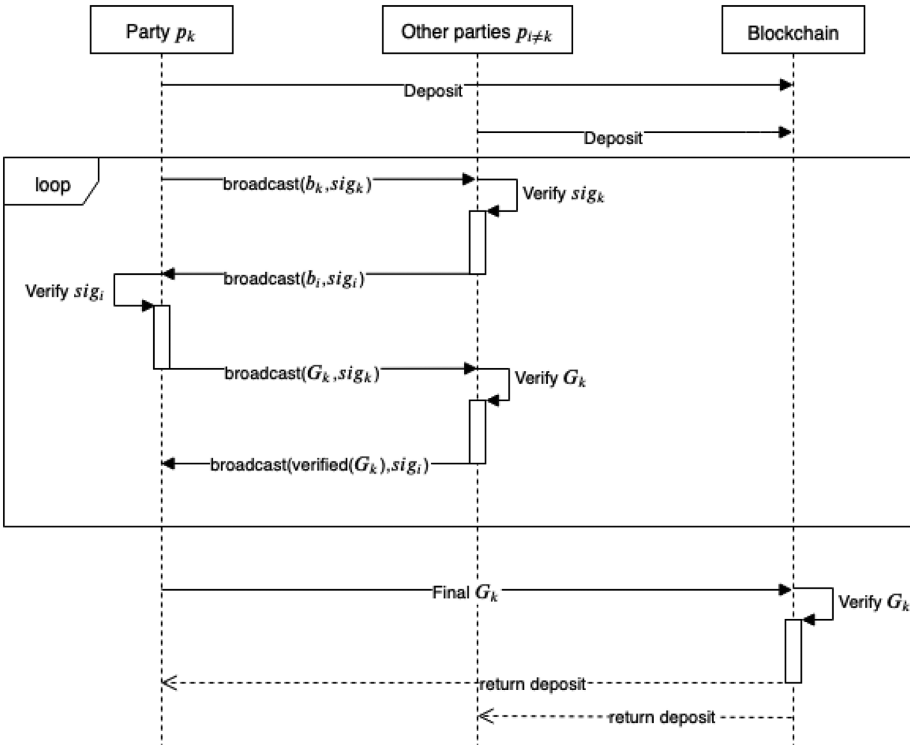


Fig. 3. Sequence diagram of the double auction process. Here, we single out one party p_k to elucidate the interactions among parties.

thus enhances the privacy. In this section, we only provide a very abstract workflow of the system without considering the security and privacy. In the following sections, we provide more detailed operations of the proposed protocol as well as how we ensure the system is secured.

3.4 Security and Privacy Goals

Before we present the formal development of our solution, we establish the threat model as well as security and privacy goals for our system. In this work, we consider a computationally efficient adversary who can corrupt any subset of parties. By corruption, the attacker can take full control over a party, which includes acquiring the internal state and all the messages destined to that party. Moreover, it can send arbitrary messages on the corrupted party's behalf.

With respect to the adversarial model, we define the security and privacy notions of interest as follows:

- Unforgeability: We use the ECDSA signature scheme, which is believed to be unforgeable against chosen message attack [13]. This signature scheme is currently being used by the Ethereum blockchain [29].
- Non-Repudiation: Once a party has submitted a bid, the system assures that he or she must not be able to deny having made the relevant bid.
- Public Verifiability: All parties can be publicly verified if they have been following the auction protocol correctly.

- Robustness: The auction process can tolerate invalid bids and dishonest participants who are not following the auction protocol correctly.
- Input independence: Each party does not see others' bid before committing to their own.
- Liveness: In an optimistic case when all parties are honest, the computation is processed within a small amount of time (off-chain messages only). When some parties are corrupted, the computation is completed within a predictable amount of time.

4 DOUBLE AUCTION USING MULTIPARTY STATE CHANNELS

In this section, we describe the ideal functionality of our system that defines how a double auction process is operated using the multiparty state channel technology. We show that the ideal functionality achieves the security goals. Afterwards, we present the design of our protocol that realizes the ideal functionality. Finally, a detailed security proof in the UC framework is given.

4.1 Security Model

The entities in our system are modeled as interactive Turing machines that communicate with each other via a secure and authenticated channel. The system operates in the presence of an adversary $\mathcal{A}$ who, upon corruption of a party p, seizes the internal state of p and all the incoming and outgoing packets of p.

4.1.1 Assumptions and Notation. We denote $\mathcal{P} = \mathcal{B} \cup \mathcal{S} = \{p_1, p_2, \ldots, p_n\}$ as the set of n parties. We assume that $\mathcal{P}$ is known before opening the state channel and $|\mathcal{P}| \geq 2$. The blockchain is represented as an append-only ledger $\mathcal{L}$ that is managed by a global ideal functionality $\mathcal{F}_\mathcal{L}$ (such as Reference [6]). The state of $\mathcal{F}_\mathcal{L}$ is defined by the current balance of all accounts and smart contracts' state and is publicly visible to all parties. $\mathcal{F}_\mathcal{L}$ supports the functionalities of adding or subtracting one's balance. We also denote $\mathcal{F}(x)$ as retrieving the current value of the state variable x from an ideal functionality $\mathcal{F}$.

We further assume that any message destined to $\mathcal{F}_\mathcal{L}$ can be seen by all parties (in the same manner as blockchain transactions are publicly visible). For simplicity, we assume that all parties have enough funds in their accounts for making deposits to the smart contract. Furthermore, each party and the ideal functionality will automatically discard any messages originated from a party that is not in $\mathcal{P}$ or the message's signature is invalid.

4.1.2 Communication. In this work, we assume a synchronous communication network. We define a *round* as a unit of time corresponding to the maximum delay needed to transmit an off-chain message between a pair of parties. Any modifications on $\mathcal{F}_\mathcal{L}$ and smart contracts take at most $\Delta \in \mathbb{N}$ rounds; this Δ reflects the fact that updates on the blockchain are not instant but can be completed within a predictable amount of time. Furthermore, each party can retrieve the current state of $\mathcal{F}_\mathcal{L}$ and smart contracts in one round.

4.1.3 Commitment Scheme. One cryptographic primitive that is used in our model is the commitment scheme. A commitment scheme consists of two following algorithms (Com, Vrf):

- $Com(m, r)$: given a message m, random nonce r, returns commitment c
- $Vrf(c, R)$: given a commitment c, and $R \triangleq (m, r)$, where m is a message, r is a random nonce, returns 1 iff $c = Com(m, r)$, otherwise returns 0.

We assume that there is no adversary $\mathcal{A}$ that can generate a commitment c and the tuples (m, r), (m', r'), such that $c = Com(m, r) = Com(m', r')$. Simply speaking, a party cannot alter the value after they have committed to it.

Functionality $\mathcal{F}_{\mathcal{L}}$

Store a vector $(x_1, x_2, ..., x_n)$ that denotes the balance of n parties.

Adding and subtracting balances.

- On **input** $update(p_i, s)$:
  (1) If $s \geq 0$, set $x_i = x_i + s$
  (2) If $s < 0$ and $x_i \geq -s$, set $x_i = x_i + s$
  (3) Otherwise, reply with an $error()$ message and stop.

Fig. 4. Ledger's functionality $\mathcal{F}_{\mathcal{L}}$.

4.2 Ideal Functionality

First, we define the ledger's ideal functionality $\mathcal{F}_{\mathcal{L}}$. Based on Section 4.1, the $\mathcal{F}_{\mathcal{L}}$ supports adding and subtracting one's balance, hence, we give the corresponding definition in Figure 4.

The formal definition of the ideal functionality $\mathcal{F}_{auction}$ is presented in Figure 5. As can be seen, it supports the following functionalities:

- Open channel
- Determine best response
- Revocation
- Close channel

As indicated in Reference [6], a state channel should be able to guarantee the consensus on creation and closing. The state channel creation is initiated by receiving a $create()$ message from a party. The functionality then waits for receiving $create()$ from all other parties within $1 + (n - 1)\Delta$ rounds. If this happens then the functionality removes a deposit from each party's balance on the blockchain. Since all parties have to send the $create()$ message, we achieve the consensus on creation.

Each iteration k of the double auction process starts with receiving the $best\_response(k)$ message from a party. Then all parties must submit a commitment of their bids. After that, all parties must submit the true bid that matches with the commitment they sent before. Any party that fails to submit in time or does not submit the true bid will be eliminated from the double auction process. With the commitment step, one party cannot see the other parties' bid prior to placing his or her own bid; this satisfies the *Input independence*.

When one party fails to behave honestly, it will be eliminated from the auction process and will not receive the deposit back. A party can voluntarily abort an auction process by sending a $revoke()$ message and it will receive the deposit back. Then, the auction can continue with the remaining parties. Therefore, the functionality satisfies the *Robustness*. Moreover, a malicious party cannot delay the execution of the protocol for an arbitrary amount of time, because after timeout, the execution still proceeds. In the best case, when everyone behaves honestly and does not terminate in the middle of the auction process, the computation is processed within $O(1)$ rounds, otherwise, $O(\Delta)$ rounds. Thus, the *Liveness* is satisfied.

In the end, the state channel begins its termination procedure upon receiving a $close()$ message from a party. Next, it awaits obtaining the $close()$ messages from the remaining parties within $1 + (|\mathcal{P}| - 1)\Delta$ rounds. If all the parties are unanimous in closing the state channel, then the functionality returns the deposit back to all parties' account. Otherwise, the state channel remains open. As all parties have to send the $close()$ message to close the state channel, we achieve the consensus on closing.

Functionality $\mathcal{F}_{auction}$

Open channel.

- On **input** $create()$ from p_i

(1) For each party p_j, $j \neq i$, wait to receive $create()$. If not receiving after $1 + (n-1)\Delta$ rounds then stop.
(2) Otherwise, instruct $\mathcal{F}_{\mathcal{L}}$ to remove a deposit from each of the party's account on the blockchain within Δ rounds.
(3) $channel = created$

Determine best response.

- On **input** $best\_response(k)$ from p_i
  - Commitment:
    (1) For each party p_i, wait until receiving $C_i^{(k)} = Com(b_i^{(k)}, r_i^{(k)})$ from p_i where $b_i^{(k)}$ is the bid and $r_i^{(k)}$ is a random nonce.
    (2) If any party p_i fails to submit the commitment within 1 round, remove p_i from $\mathcal{P}$ then stop.
  - Reveal and compute:
    (1) For each party p_i, wait until receiving $R_i^{(k)} = (b_i^{(k)}, r_i^{(k)})$ from p_i.
    (2) If any party p_i fails to submit within 1 round or $Vrf(C_i^{(k)}, R_i^{(k)}) = 0$, remove p_i from $\mathcal{P}$ then stop.
    (3) Compute best response G_k based on $\mathbf{b} = \{b_i^{(k)} | p_i \in \mathcal{P}\}$
    (4) Send G_k to all parties in 1 round.

Revocation.

- On **input** $revoke()$ from p_i: within Δ rounds

(1) remove p_i from $\mathcal{P}$, add the deposit to p_i's balance on the blockchain

Close channel.

- On **input** $close()$ from p_i: if $p_i \notin \mathcal{P}$ then stop. Otherwise:

(1) Within $1 + (|\mathcal{P}| - 1)\Delta$ rounds, wait for receiving $close()$ from p_j, $j \neq i$
(2) If fails to receive then stop.
(3) Else, within Δ rounds, add the deposit to every party p_i's balance on the blockchain, where $p_i \in \mathcal{P}$
(4) $channel = \perp$

Fig. 5. Ideal functionality $\mathcal{F}_{auction}$.

4.3 Protocol for Double Auction State Channel

This section discusses in detail the double auction protocol based on state channel that realizes the $\mathcal{F}_{auction}$. The protocol includes two main parts: (1) a Judge contract and (2) Off-chain protocol.

Judge contract. The main functionality of this contract is to regulate the state channel and handle disputes. Every party is able to submit a *state* that everyone has agreed on to this contract. However, the contract only accepts the state with the highest version number. Once a party submits a state G, the contract will wait for some deadline T for other parties to raise disputes. See Figure 6 for the functionality of the Judge contract. Note that the contract $\mathcal{F}_{Judge}$ has a state variable *channel* that indicates whether the channel is opened or not. If the channel is not opened ($channel = \perp$), the three functionalities "State submission," "Revocation," and "Close channel" cannot be executed.

Contract $\mathcal{F}_{Judge}$

Open channel.

- On **input** $create()$ from p_i:
  - For each party p_j, $j \neq i$, wait to receive $create()$. If not receiving after Δ rounds then stop.
  - Otherwise:
    * $channel = created$.
    * instruct $\mathcal{F}_{\mathcal{L}}$ to remove a deposit from each of the party's account on the blockchain within Δ rounds.
    * Initialize $bestVersion = -1$, $state = \emptyset$, $flag = \perp$, the set of parties $\mathcal{P}$

State submission.

- On **input** $state\_submit(p_r, v, G, proof)$ from p_i

(1) if $p_r \neq \perp$, wait for $(|\mathcal{P}| - 2)\Delta$ rounds. Then, if it receives $state\_submit(p'_r, v', G', proof')$, such that $p'_r = p_r$, from all parties except p_r and p_i, then remove p_r from $\mathcal{P}$
(2) if $v \leq bestVersion$ then stop.
(3) Verify the signatures of $\mathcal{P}$ on G and verify the state G using $proof$. If failed then stop
(4) $bestVersion = v$
(5) $state = G$
(6) $flag = dispute$
(7) Set $flag = \perp$ after a deadline of T rounds unless $bestVersion$ is changed.

Revocation.

- On **input** $revoke()$ from p_i: within Δ rounds

(1) remove p_i from $\mathcal{P}$, instruct $\mathcal{F}_{\mathcal{L}}$ to add the deposit to p_i's balance on the blockchain

Close channel.

- On **input** $close()$ from p_i: if $p_i \notin \mathcal{P}$ then stop. Otherwise:

(1) If $flag = dispute$ then stop.
(2) Within $1 + (|\mathcal{P}| - 1)\Delta$ rounds, wait for receiving $close()$ from p_j, $j \neq i$
(3) If fails to receive then stop.
(4) Else, within Δ rounds, add the deposit to every party p_i's balance on the blockchain, where $p_i \in \mathcal{P}$
(5) $channel = \perp$

Fig. 6. Judge contract.

In the same manner, if the channel is already opened ($channel = created$), then the functionality "Open channel" cannot be executed.

As the contract always maintains the valid state on which all parties have agreed (by verifying all the signatures), we can publicly verify if all parties are following the protocol. A dishonest party who attempts to submit an outdated state will be detected, as the smart contract is public; that state would be then overwritten by a more recent state. When the state channel is closed, the contract is now holding the latest state with the final bids of all the parties, and by the immutability of blockchain, no bidder can deny having made the relevant bid. Therefore, this contract satisfies the *Non-Repudiation* and *Public Verifiability* goals.

Off-chain protocol. In this section, we present the off-chain protocol that operates among parties in a double auction process. In the same manner as $\mathcal{F}_{auction}$, the protocol consists of four parts: (1) Create state channel, (2) Determine best response, (3) Revocation, and (4) Close state channel.

Protocol : Create state channel

Party p_i: On **input** $create()$ from environment

(1) Send $create()$ to $\mathcal{F}_{Judge}$ and wait for $1 + (n-1)\Delta$ rounds.

Party $p_{j \neq i}$: Upon p_i sends $create()$ to $\mathcal{F}_{Judge}$

(2) Send $create()$ to $\mathcal{F}_{Judge}$ and wait for $(n-2)\Delta$ rounds.

For each party:

(3) If $\mathcal{F}_{Judge}(channel) = created$ then outputs $created()$ to the environment.

Fig. 7. Protocol: Create state channel.

Protocol: $Submit(p_i, p_r, v, G)$

Party p_i

(1) If $p_r = \perp$ and $v \leq \mathcal{F}_{judge}(bestVersion)$ then stop.
(2) Otherwise, construct $proof$ for G and send $state\_submit(p_r, v, G, proof)$ to $\mathcal{F}_{Judge}$ in Δ rounds.
(3) Wait until $\mathcal{F}_{judge}(flag) = \perp$ then stop.

Party $p_{j \neq i}$: On $\mathcal{F}_{judge}(flag) = dispute$

(4) If the latest valid state G_k has $k > \mathcal{F}_{judge}(bestVersion)$ then construct $proof$ for G_k and send $state\_submit(\perp, k, G_k, proof)$ to $\mathcal{F}_{Judge}$ in Δ rounds.
(5) Wait until $\mathcal{F}_{judge}(flag) = \perp$ then stop.

Fig. 8. Protocol: *Submit*.

Figure 12 illustrates the connections among these parts as well as the execution order of the protocol.

First, to create a new state channel, the environment sends a message $create()$ to one of the parties. Let us denote this initiating party as p_i. The detailed protocol is shown in Figure 7. p_i will send a $create()$ message to the smart contract $\mathcal{F}_{Judge}$, which will take Δ rounds to get confirmed on the blockchain. As this message is visible to the whole network, any $p_{j \neq i}$ can detect this event and also send a $create()$ message to $\mathcal{F}_{Judge}$. To detect this event, each p_j needs to retrieve the current state of blockchain, which takes 1 round, and as there are $n-1$ parties p_j, p_i has to wait $1 + (n-1)\Delta$ rounds. If all parties agree on creating the state channel, this process will be successful and the channel will be opened. After that, the smart contract will take a deposit from the account of each party.

When parties run into dispute, they will have to resolve on-chain. In specific, the procedure $Submit()$ as shown in Figure 8 allows any party to submit the current state to the smart contract. However, as stated above, $\mathcal{F}_{Judge}$ only considers the valid state that has the highest version number. In this procedure, we also define a $proof$ of a state G. Based on the algorithm used for double auction, this $proof$ is anything that can verify whether the calculation of G in an iteration is correct or not. For example, $proof$ can be all the valid bids in that iteration. When any party submits a state, the $\mathcal{F}_{Judge}$ will raise the state variable $flag = dispute$. Upon detecting this event, other parties can submit their states if they have higher version numbers. After a deadline of T rounds, if none of the parties can submit a newer state, then $\mathcal{F}_{Judge}$ will set $flag = \perp$ to conclude the dispute period. Furthermore, we also note that this procedure also supports eliminating any

Protocol: Determine best response

Party p_i: On **input** *best_response*(k) from the environment

(1) Broadcast $C_i^{(k)} = Com(b_i^{(k)}, r_i^{(k)})$ to other parties and wait for 1 round. Then go to step 4.

Party $p_{j \neq i}$: On **input** $C_i^{(k)}$ from p_i

(2) Broadcast $C_j^{(k)} = Com(b_j^{(k)}, r_j^{(k)})$ to other parties and wait for 1 round.

(3) If there exists a party $p_{l \neq j}$ such that it doesn't receive $C_l^{(k)}$ then execute $Submit(p_l, k-1, G_{k-1})$ and stop.

Party p_i

(4) If there exists a party p_j such that it doesn't receive $C_j^{(k)}$ then execute $Submit(p_j, k-1, G_{k-1})$ and stop.

(5) Broadcast $R_i^{(k)} = (b_i^{(k)}, r_i^{(k)})$ to other parties and wait for 1 round. Then go to step 8.

Party $p_{j \neq i}$: On **input** $R_i^{(k)}$ from p_i

(6) Broadcast $R_j^{(k)} = (b_j^{(k)}, r_j^{(k)})$ to other parties and wait for 1 round.

(7) If there exists a party $p_{l \neq j}$ such that it doesn't receive $R_l^{(k)}$ or $Vrf(C_l^{(k)}, R_l^{(k)}) = 0$ then execute $Submit(p_l, k-1, G_{k-1})$ and stop.

Party p_i

(8) If there exists a party p_j such that it doesn't receive $R_j^{(k)}$ or $Vrf(C_j^{(k)}, R_j^{(k)})$ then execute $Submit(p_j, k-1, G_{k-1})$ and stop.

(9) Compute best response G_k, $sig_{p_i}^{G_k} = sign_{p_i}(G_k)$ and broadcast $best\_response(G_k, sig_{p_i}^{G_k})$ to other parties. Wait for 1 round then go to step 13.

Party $p_{j \neq i}$: On **input** $best\_response(G_k, sig_{p_i}^{G_k})$ from p_i

(10) Verify G_k

(11) If G_k is not correct then execute $Submit(p_i, k-1, G_{k-1})$ and stop.

(12) Otherwise, let $sig_{p_j}^{G_k} = sign_{p_j}(G_k)$ and broadcast $verified(G_k, sig_{p_j}^{G_k})$ to other parties. Then wait for 1 round.

For each party:

(13) If there exists a party $p_{l \neq i}$ such that it doesn't receive $verified(G_k, sig_{p_l}^{G_k})$ then execute $Submit(p_l, k-1, G_{k-1})$ and stop.

Fig. 9. Protocol: Determine best response.

dishonest party that does not follow the protocol by setting the parameter p_r to that party. This function requires a unanimous agreement among the remaining honest parties. If a party p_i wants to eliminate a party p_r, then it will need other parties, except p_r, to call the $Submit()$ protocol to remove p_r from $\mathcal{P}$.

Next, in Figure 9, we present the protocol for determining the best response, which only consists of off-chain messages if all the parties are honest. In each iteration k, this process starts when the environment sends $best\_response(k)$ to a party p_i. Again, this does not violate the trustless property, since p_i can be any party chosen at random. First, p_i broadcasts the commitment $C_i^{(k)}$ of its bid, which only takes one round, since this is an off-chain message. Other parties upon receiving

Protocol : Revocation

Party p_i: On **input** *revoke*() from the environment

(1) Send *revoke*() to $\mathcal{F}_{Judge}$ in Δ rounds then stop.

Party $p_{j \neq i}$: On changes of $\mathcal{F}_{judge}(\mathcal{P})$

(2) Update the local $\mathcal{P}$ with $\mathcal{F}_{judge}(\mathcal{P})$

Fig. 10. Protocol: Revocation.

Protocol : Close state channel

Party p_i: On **input** *close*() from environment

(1) If $\mathcal{F}_{Judge}(flag) = dispute$ then stop.
(2) Send *close*() to $\mathcal{F}_{Judge}$ and wait for $1 + (|\mathcal{P}| - 1)\Delta$ rounds. Then go to step 4

Party $p_{j \neq i}$: Upon p_i sends *close*() to $\mathcal{F}_{Judge}$

(3) Send *close*() to $\mathcal{F}_{Judge}$ and wait for $(|\mathcal{F}_{auction}(\mathcal{P})| - 2)\Delta$ rounds.

For each party:

(4) Wait for Δ rounds and check if $\mathcal{F}_{Judge}(channel) = \perp$ then outputs *closed*() to the environment.

Fig. 11. Protocol: Close state channel.

this message will also broadcast their commitments. Then, p_i proceeds to broadcast the opening $R_i^{(k)}$ of its bid and hence, other parties upon receiving this $R_i^{(k)}$ also broadcast their openings. If any party refuses to send their bids or sends an invalid bid, other parties will call the *Submit* procedure to eliminate that dishonest party from the auction process. Thus, that party will lose all the deposit. In practice, one may consider refunding a portion of deposit back to that party. To achieve this, we only need to modify the first line of the functionality "State submission" in $\mathcal{F}_{Judge}$ to return a portion of deposit to p_r.

During the auction process, some parties may want to abort the auction process. To avoid losing the deposit, they must use the Revocation protocol described in Figure 10 to send a *revoke*() message to $\mathcal{F}_{Judge}$. In this case, they will get the deposit back in full and be removed from the set $\mathcal{P}$. Other parties upon detecting this operation also update their local $\mathcal{P}$ to ensure the consistency.

Finally, Figure 11 illustrates the protocol for closing the state channel. One technical point in this protocol is that we must check whether there is any ongoing dispute. If so, then we must not close the channel. In the same way of opening the channel, a party p_i also initiates the request by sending a message *close*() to the smart contract. Upon detecting this event, other parties may also send *close*(). If all parties agreed on closing the channel, they will get the deposit back.

4.4 Security and Privacy Analysis

In this section, we prove the security of our solution in the UC model. We denote $\text{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ as the outputs of the environment $\mathcal{E}$ when interacting with the parties running the protocol π and the adversary $\mathcal{A}$. From Reference [15], we have the following definition:

Definition 4.1 (UC-security). A protocol π UC-realizes an ideal functionality $\mathcal{F}$ if for any adversarial $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{E}$ the outputs $\text{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ and $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.

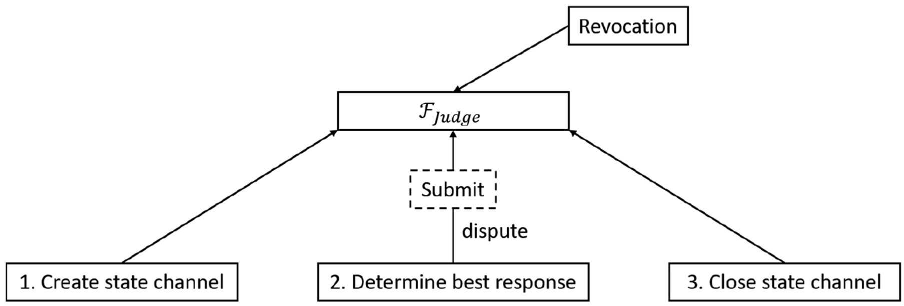


Fig. 12. Connections and execution order of the functionalities of the proposed protocol. The "Revocation" functionality can be triggered any time between "Create state channel" and "Close state channel." Any dispute occurred during the "Determine best response" requires a call to the "Submit" functionality.

The main goal of this analysis is to prove that the off-chain protocol UC realizes the ideal functionality $\mathcal{F}_{auction}$ by constructing a simulator $\mathcal{S}$ in the ideal world that translates every attacker in the real world, such that the two worlds are indistinguishable. To achieve that, we need to ensure the consistency of timings, i. e., the environment $\mathcal{E}$ must receive the same message in the same round in both worlds. Furthermore, in any round, the internal state of each party must be identical between the two worlds, which will make $\mathcal{E}$ unable to perceive whether it is interacting with the real world or the ideal one.

Per Canetti [3], the strategy for proving the UC-security is constructing the simulator $\mathcal{S}$ that handles the corrupted parties and simulates the $(\mathcal{F}_{judge}, \mathcal{F}_{\mathcal{L}})$-hybrid world while interacting with $\mathcal{F}_{auction}$. The simulator will maintain a copy of the hybrid world internally so it can turn every behavior in the hybrid world into an indistinguishable one in the real world. We further assume that upon receiving a message from a party, the ideal functionality $\mathcal{F}_{auction}$ will leak that message to the simulator. For simplicity, we do not elaborate these operations when constructing the simulator. Since $\mathcal{S}$ locally runs a copy of the hybrid world, $\mathcal{S}$ knows the behavior of the corrupted parties and the messages sent from $\mathcal{A}$ to $\mathcal{F}_{Judge}$, therefore, $\mathcal{S}$ can instruct the $\mathcal{F}_{auction}$ to update the ledger $\mathcal{L}$ in the same manner as the hybrid word.

In specification of the off-chain protocol, the protocol Submit in Figure 8 is called as a subroutine of the protocol Determine best response in Figure 9. Hence, we first define a simulator for the protocol Submit by proving the following lemma:

Lemma 4.2. *Under the assumptions given in Section 4.1, we can construct a simulator for the protocol Submit in the ideal world such that the view of $\mathcal{E}$ remains the same in both the ideal world and the $(\mathcal{F}_{judge}, \mathcal{F}_{\mathcal{L}})$-hybrid world.*

Proof. Let p_i be the party that calls the $Submit()$ protocol, we define $S\_Submit()$ as the simulator of the protocol $Submit()$. If p_i is corrupted, then upon p_i sends $state\_submit(p_r, v, G, proof)$ to $\mathcal{F}_{Judge}$

(1) $S\_Submit()$ waits for Δ rounds. If $p_r = \perp$ then stop.
(2) Otherwise, $S\_Submit()$ waits for $(|\mathcal{P}| - 2)\Delta$ rounds
(3) If all parties $p_{j \neq \{i,r\}}$ send $state\_submit(p_r, v, G, proof)$ to $\mathcal{F}_{Judge}$ then instruct $\mathcal{F}_{auction}$ to remove p_r from $\mathcal{P}$.

If $p_{j \neq i}$ is corrupted, then $S\_Submit()$ also updates its $\mathcal{P}$ in the same round as the real world if $\mathcal{F}_{Judge}$ updates $\mathcal{P}$.

Since the protocol $Submit()$ can potentially change the internal state of the $\mathcal{F}_{judge}$ and the parties by removing a party from $\mathcal{P}$, the $S\_Submit()$ ensures that $\mathcal{F}_{auction}$ also performs the same operation in the same round. Hence, the view of both worlds are consistent. □

Finally, we prove the following theorem:

Theorem 4.3. *Under the assumptions given in Section 4.1, the proposed off-chain protocol UC-realizes the ideal functionality $\mathcal{F}_{auction}$ in the $(\mathcal{F}_{judge}, \mathcal{F}_{\mathcal{L}})$-hybrid model.*

Proof. We provide the description of $\mathcal{S}$ for each of the functionalities as follows:

Open channel. Let p_i be the party that initiates the request. We inspect the following cases:

- *p_i is corrupted:* Upon p_i sends $create()$ to $\mathcal{F}_{Judge}$
  (1) $\mathcal{S}$ waits for Δ rounds
  (2) Then sends $create()$ to $\mathcal{F}_{auction}$ to make sure that $\mathcal{F}_{auction}$ receives $create()$ in the same round as $\mathcal{F}_{Judge}$. Then wait for $1 + (n-1)\Delta$ rounds
  (3) if $\mathcal{F}_{auction}(channel) = created$ then sends $created()$ to $\mathcal{E}$ on behalf of p_i.
- *$p_{j \neq i}$ is corrupted:* Upon p_i sends $create()$ to $\mathcal{F}_{auction}$
  (1) $\mathcal{S}$ waits for Δ rounds
  (2) If p_j sends $create()$ to $\mathcal{F}_{Judge}$ then $\mathcal{S}$ sends $create()$ to $\mathcal{F}_{auction}$ and wait for $(n-2)\Delta$ rounds
  (3) if $\mathcal{F}_{auction}(channel) = created$ then sends $created()$ to $\mathcal{E}$ on behalf of p_j

In all cases above, according to Figure 7, p_i or p_j will output $created()$ to $\mathcal{E}$ if $\mathcal{F}_{auction}(channel) = created$. Hence, $\mathcal{S}$ also outputs $created()$ in the same round. Therefore, the environment $\mathcal{E}$ receives the same outputs in the same round in both worlds.

Close channel. Let p_i be the party that initiates the request. We inspect the following cases:

- *p_i is corrupted*: Upon p_i sends $close()$ to $\mathcal{F}_{Judge}$
  (1) if $\mathcal{F}_{judge}(flag) = dispute$ then stop. Otherwise, $\mathcal{S}$ waits for Δ rounds
  (2) Then sends $close()$ to $\mathcal{F}_{auction}$ to make sure that $\mathcal{F}_{auction}$ receives $close()$ in the same round as $\mathcal{F}_{Judge}$. Then wait for $1 + (\mathcal{F}_{auction}(|\mathcal{P}|) - 1)\Delta$ rounds
  (3) Wait for another Δ round and check if $\mathcal{F}_{auction}(channel) = \perp$ then sends $close()$ to $\mathcal{E}$ on behalf of p_i.
- *$p_{j \neq i}$ is corrupted:* Upon p_i sends $close()$ to $\mathcal{F}_{auction}$
  (1) $\mathcal{S}$ waits for Δ rounds
  (2) If p_j sends $close()$ to $\mathcal{F}_{Judge}$ then $\mathcal{S}$ sends $close()$ to $\mathcal{F}_{auction}$ and wait for $(|\mathcal{F}_{auction}(\mathcal{P})| - 2)\Delta$ rounds
  (3) Wait for another Δ round and check if $\mathcal{F}_{auction}(channel) = \perp$ then sends $closed()$ to $\mathcal{E}$ on behalf of p_j.

The indistinguishability in the view of $\mathcal{E}$ between the two worlds holds in the same manner as *Open channel.*

Revocation. Let p_i be the party that initiates the request. We inspect the following cases:

- *p_i is corrupted*: Upon p_i sends $revoke()$ to $\mathcal{F}_{Judge}$
  (1) $\mathcal{S}$ waits for Δ rounds
  (2) Then sends $revoke()$ to $\mathcal{F}_{auction}$ to make sure that $\mathcal{F}_{auction}$ receives $revoke()$ in the same round as $\mathcal{F}_{Judge}$.
- *$p_{j \neq i}$ is corrupted:* Upon p_i sends $revoke()$ to $\mathcal{F}_{auction}$
  (1) If p_j updates the local $\mathcal{P}$ then $\mathcal{S}$ also updates its $\mathcal{P}$.

In both cases, $\mathcal{S}$ ensures that the messages exchanged between the entities are identical in both worlds. Moreover, since $\mathcal{P}$ is updated according to the real world, thus the internal state of the each party are also identical. Therefore, the view of $\mathcal{E}$ between the two worlds are indistinguishable.

Determine best response. Based on Lemma 4.2, we define $S\_Submit()$ as the simulator of the protocol $Submit()$ in the ideal world. Let p_i be the party that calculates the best responses. In each iteration k, we inspect the following cases:

- *p_i is corrupted:* Upon p_i broadcasts $C_i^{(k)}$ to other parties
  1. Send $C_i^{(k)}$ to $\mathcal{F}_{auction}$ and wait for 1 round.
  2. If $\mathcal{F}_{auction}$ removes any party then stop. If p_i executes the $Submit()$ then $\mathcal{S}$ also calls the $S\_Submit()$ in the same round.
  3. Otherwise, if p_i broadcasts $R_i^{(k)}$ to other parties then $\mathcal{S}$ sends $R_i^{(k)}$ to $\mathcal{F}_{auction}$ and waits for 1 round. Else, stop.
  4. If $\mathcal{F}_{auction}$ removes any party then stop. If p_i executes the $Submit()$ then $\mathcal{S}$ also calls the $S\_Submit()$ in the same round. Otherwise, wait for 1 round
  5. Receive G_k from $\mathcal{F}_{auction}$ and wait for 1 round.
  6. If p_i executes the $Submit()$ then $\mathcal{S}$ also calls the $S\_Submit()$ in the same round. Otherwise, stop.
- *$p_{j\neq i}$ is corrupted:* Upon p_i sends $best\_response(k)$ to $\mathcal{F}_{auction}$
  1. Wait until p_i sends $C_i^{(k)}$ to $\mathcal{F}_{auction}$, then forwards that $C_i^{(k)}$ to p_j in the same round.
  2. If p_j broadcasts $C_j^{(k)}$ to other parties then $\mathcal{S}$ sends $C_j^{(k)}$ to $\mathcal{F}_{auction}$. Else, execute $S\_Submit()$ to eliminate the party that made p_j refuse to broadcast and stop.
  3. Wait for 1 round. If p_i sends $R_i^{(k)}$ to $\mathcal{F}_{auction}$, then forwards that $R_i^{(k)}$ to p_j in the same round. Otherwise, stop.
  4. If p_j broadcasts $R_j^{(k)}$ to other parties then $\mathcal{S}$ sends $R_j^{(k)}$ to $\mathcal{F}_{auction}$. Else, execute $S\_Submit()$ to eliminate the party that made p_j refuse to broadcast and stop.
  5. Wait for 1 round, if $\mathcal{S}$ does not receive G_k from $\mathcal{F}_{auction}$ then stop. Otherwise, $\mathcal{S}$ forwards that G_k to p_j.
  6. If p_j executes the $Submit()$ then $\mathcal{S}$ also calls the $S\_Submit()$ in the same round. Otherwise, stop.

Since the messages exchanged between any entities are exact in both worlds, the indistinguishability in the view of $\mathcal{E}$ between the two worlds holds. □

5 IMPLEMENTATION AND EVALUATION

In this section, we present an evaluation of our proposed double auction framework by running some experiments on a proof-of-concept implementation. Before that, we introduce our novel development framework for distributed computing on state channels.

5.1 Development Framework for Distributed Computing Based on State Channels

To the best of our knowledge, this is the first work that builds a functioning programming framework that can be used to deploy any distributed protocols using blockchain and state channels. We use the Elixir programming language to implement the protocol for the participating parties that run on Erlang virtual machines (EVM). The smart contract for the state channel is implemented using the Solidity programming language, which is the official language for realizing smart contracts on Ethereum. Hence, the whole system can be deployed on an Ethereum blockchain.

The Elixir implementation of the participating parties is designed around the Actor programming model [10]. This model realizes *actors* as the universal primitive of concurrent computation that use message-passing as the communication mechanism. In response to a message, an actor can make local decisions, send messages to other actors, or dynamically generate more actors. Since actors in Elixir are "location transparent," when sending a message, whether the recipient actor is on the same machine or on another machine, the EVM can manage to deliver the message in both cases. In our framework, each of the parties is treated as a separate actor, and they communicate with each other using asynchronous message-passing mechanism. Furthermore, for the parties to interact with the smart contract, we leverage a remote procedure call protocol encoded in JSON (JSON-RPC) provided by Ethereum. The actors also listen for triggered events (e.g., the channel is opening) from the smart contract and carry out appropriate action.

To develop a distributed computing system using our framework, one would only have to define the following:

(1) The state of the system at each iteration.
(2) The operation to be executed at each iteration.
(3) The off-chain messages to be exchanged among the parties of the system.
(4) The rule to resolve disputes.

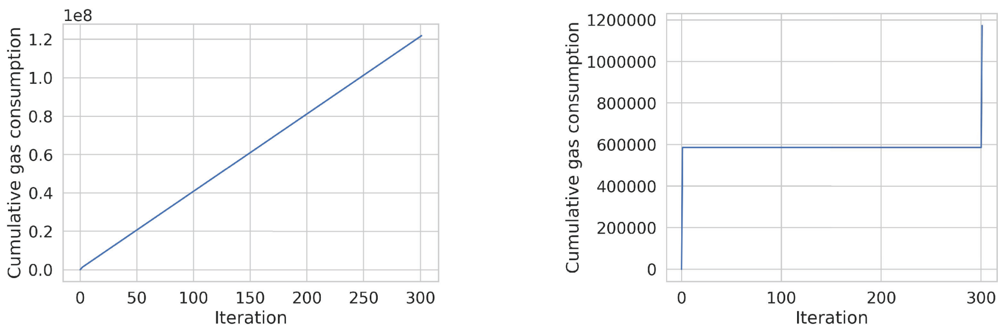
To demonstrate the feasibility of this development framework, we use it to develop a proof-of-concept implementation of the proposed double auction protocol in the next section.

5.2 Proof-of-concept Implementation of Double Auction

We create a proof-of-concept implementation of our proposed double auction protocol using our development framework, as illustrated above. Our implementation of the Judge contract closely resembles the protocol from Figure 6; it is developed in the Truffle framework [24] and later deployed on an Ethereum testnet managed by Ganache [23]. Our main goal was to illustrate the feasibility and practicality of our off-chain protocols when interacting with the Judge smart contracts. As regards the commitment scheme, we used the SHA-256 hash function with a random string of 64 characters.

One crucial evaluation criteria when working with smart contracts on the Ethereum blockchain are costs incurred by transaction fees. In Ethereum, these fees are calculated using a special unit called "gas," the fee is paid by the sender of a transaction to the miner that validates that transaction. Transactions in an Ethereum blockchain can provide inputs to and execute smart contract's functions. The amount of gas used for each transaction is determined by the amount of data it sends and the computational effort that it will take to execute certain operations. In addition, depending on the exchange rate between gas and ETH (Ethereum's currency), we can determine the final cost in ETH. In practice, this exchange rate is decided by the person who issues transactions depending on how much they want to prioritize their transactions in the mining pool. As we are using the Ganache testnet, in our calculation, we use the default gas price of 1 gas = 2×10^{-8} Ether.

To deploy the Judge contract, we have to pay 3,387,400 gas, which corresponds to 0.0677 ETH. Using an exchange rate of 1 ETH = 152.49 USD (as of Nov. 2019), the deployment of the contract costs approximately 10.3 USD. We also note that our proof-of-concept implementation did not aim to optimize the deployment cost, since we implement every functionality on a single smart contract. To mitigate this cost, we would only need to transfer all functionalities of the Judge contract into an external library. Hence, this library only requires one-time deployment; instances of the Judge contract can refer to this library and re-use its functionalities. Therefore, we can save much of the deployment cost, since we do not have to re-deploy all the functionalities with each



(a) Straw-man design

(b) Using state channel

Fig. 13. Cumulative gas consumption over time.

instance of the Judge contract. Thus, in the following section, we disregard this deployment cost when evaluating the performance of our system.

5.3 Experiments

For the experiments, we run the system on machines equipped with an Intel Core i7-8550U CPU 1.8 GHz and 16 GB of RAM. The mean latency for transmitting a 32-byte message from one party to another is about 101.2 ms. We assume that, in the beginning, the parties of the system collected the public keys of each other. Moreover, each party also created an account on the Ethereum testnet. Since this is a one-time setup procedure, we do not take it into consideration when evaluating our system. To receive events from the Judge contract, each party periodically checks for triggered events from the Judge contract every 100 ms. To illustrate the double auction process, we follow the setup of a numerical experiment in Reference [30] that consists of 10 parties in total, in which 6 are buyers and 4 are sellers. The auction process takes a total of 300 iterations to reach NE.

First, we implemented the straw-man design as described in Section 3.2. Figure 13(a) shows the cumulative gas consumption over time. As can be seen, the gas consumption increases linearly with the iterations. This happens because each iteration requires parties to send on-chain transactions to the smart contract. In sum, the straw-man design used 121,913,080 gas, which corresponds to 2.44 ETH. This means the design incurs a cost of about 372 USD to carry out the double auction process. Furthermore, if we consider the Ethereum's block time of 15 seconds per block, it could take over 75 minutes to complete the process (assuming that the waiting time in the mining pool is negligible).

Next, we run the double auction on our proof-of-concept implementation. If all the parties unanimously agree to create the state channel, they need to issue 10 on-chain transactions that cost about 0.0117 ETH (each party needs to send 1 transaction to the smart contract). For closing the state channel, they need another 10 transactions that cost about 0.0109 ETH. The whole double auction process is executed off-chain and does not cost any gas or ETH. In Figure 13(b), which shows the cumulative gas consumption over time, we can observe that only the first and last iteration incurs some amount of gas to create and close the state channel, respectively. As a result, the proposed protocol only incurs 20 on-chain transactions and 1,133,420 gas. With comparison to the straw-man design, we reduced the amount of transactions and gas by 99%. Additionally, regarding the nominal cost, our solution requires only 0.0226 ETH, which is about 3 USD, to conduct the auction process. Table 1 summarizes the costs of execution of our proposed system.

Table 1. The Estimated Costs for Executing the Transactions as Well as the Message Complexity of the Proposed Double Auction Protocol When Running with 10 Parties

	# on-chain tx	Gas	ETH	# off-chain message
Create state channel	10	586,180	0.0117	0
Double auction process	0	0	0	81,000
Close state channel	10	547,240	0.0109	0

For a more detailed analysis, we look into other scenarios and measure some aspects of the system. One scenario to consider is when a party becomes dishonest and attempts to close the state channel with an outdated state. To do that, the dishonest party has to submit the outdated state using the *state_submit* function of the Judge contract, which costs about 52,845 gas. After that, another party would submit a more recent state that can overwrite that outdated state, which costs another 52,845 gas. Consequently, the cost for settling a disagreement would be about 0.0021 ETH for both the dishonest and honest party.

Another scenario that is worth looking into is the dynamic leave of parties. The two cases supported by our system are (1) eliminating dishonest parties and (2) voluntarily aborting the auction process. In the former case, when eliminating a party via the *state_submit* function, it requires nine on-chain transactions that cost a total of 0.0122 ETH. In the latter case, using the Revocation protocol, only the leaving party needs to issue a transaction that costs about 0.0011 ETH.

To analyze the execution cost, we focus on the cost of computing digital signatures and message complexity, since these are additional work that the parties have to perform apart from computing best responses. As in the protocol design, in each iteration, each party has to sign the current state and broadcast it to all other parties. A state of an iteration is defined as the list of parties' best responses and it is about 80 bytes (each party's bid profile is 8 bytes). The time it takes to sign the state is about 1.4 ms and to verify the signature is about 0.8 ms. As for broadcasting bids among parties, a commitment of 32 bytes (using SHA-256 hash function) is sent and later accompanied by the opening of 72 bytes (including the bid and the random string). This incurs a total of 270 off-chain messages that are transmitted per iteration with the total size of 23 KB. If we consider the whole auction process that involves 300 iterations, the total size of the messages transmitted among the parties is only 6.9 MB. This emphasizes the practicality of our proposed solution because of low transmission overhead.

Finally, we measure the end-to-end latency of the auction process. For the creating and closing state channel, each will take one block time (i.e., about 15 seconds in Ethereum). The 300 iterations of determining best responses takes only about 8.3 minutes. To test the scalability, we observe how the system latency changes when we increase the number of parties. For a fair comparison, we fix the number of iterations for the double auction process to 300. In Figure 14(a), we show the end-to-end latency with 10, 40, 70, and 100 parties. The total time to create and close the state channel remains the same, because each operation only needs one block time as a block in Ethereum can store about 380 transactions. When we increase the number of parties to 40, the system needs 833.4 seconds to finish the auction process. The rise in the latency comes from the fact that the size of the state at each iteration increases (from 80 bytes to 320 bytes), hence, it takes longer to transmit the state as well as to compute the digital signatures. However, we note that with 40 parties, this latency is only 1.6 times as much as the latency with 10 parties. Likewise, when we increase the number of parties by 9 times (to 100 parties), the latency only increases by 2.9 times. Therefore, we can see that the proposed solution is able to scale with the number of parties.

To better capture the performance of our system under heavy load, Figure 14(b) illustrates the end-to-end delay with thousands of trading parties over 300 iterations. For creating and closing state channel, it would take 6, 12, 16, 22, and 28 block time with 1,000, 2,000, 3,000, 4,000, and 5,000

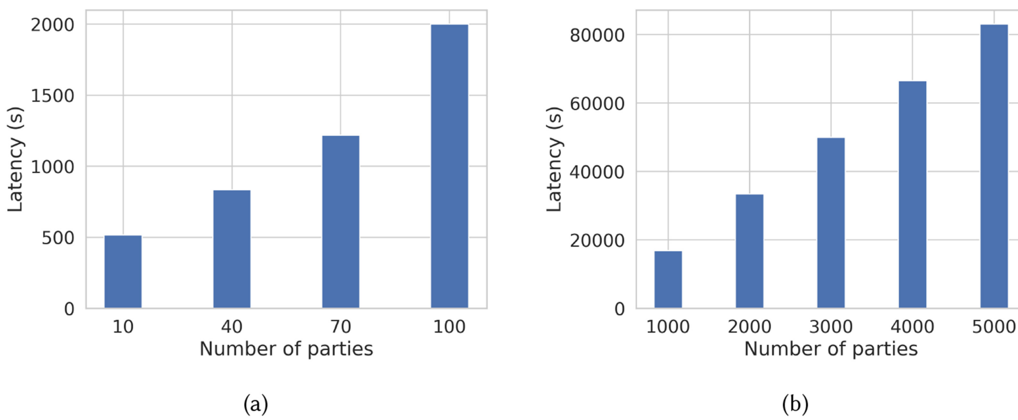


Fig. 14. End-to-end latency.

parties, respectively. As can be seen, the latency increases linearly with the number of parties. Under this load, the size of the state reaches 40,000 bytes at 5,000 parties, and the end-to-end latency is dominated by the time it takes to transmit the state at each iteration. Therefore, the performance of our system is only limited by the underlying communication channel.

In summary, the proof-of-concept implementation has demonstrated that our solution is both feasible and practical; all the functionalities work according to the design in Section 4. With comparison to the straw-man design, our implementation has resulted in a gigantic saving of money and time. Moreover, as the proposed solution induces a relatively small overhead, it is able to support a high number of trading parties.

6 CONCLUSION

In this article, we have proposed a novel framework based on blockchain that enables a complete decentralized and trustless iterative double auction. That is, all parties can participate in the auction process without having to rely on an auctioneer, and they do not have to trust one another. With an extension of the state channel technology, we were able to specify a protocol that reduces the blockchain transactions to avoid high transaction fee and latency. We have provided a formal specification of the framework, and our protocol was proven to be secured in the UC model. Finally, we have developed a proof-of-concept implementation and performed experiments to validate the feasibility and practicality of our solution.

REFERENCES

[1] Nurzhan Zhumabekuly Aitzhan and Davor Svetinovic. 2016. Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams. *IEEE Trans. Depend. Sec. Comput.* 15, 5 (2016).

[2] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. 2016. MedRec: Using blockchain for medical data access and permission management. In *Proceedings of the International Conference on Open and Big Data (OBD'16)*. IEEE, 25–30.

[3] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the IEEE International Conference on Cluster Computing*. IEEE, 136–145.

[4] Thang N. Dinh and My T. Thai. 2018. AI and blockchain: A disruptive integration. *Computer* 51, 9 (2018), 48–53.

[5] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2019. Perun: Virtual payment hubs over cryptocurrencies. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'19)*. IEEE, 106–123.

[6] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. 2018. General state channel networks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*. ACM, New York, NY, 949–966. DOI: https://doi.org/10.1145/3243734.3243856

[7] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014), 1–32.

[8] M. Nazif Faqiry and Sanjoy Das. 2016. Double-sided energy auction in microgrid: Equilibrium under price anticipation. *IEEE Access* 4 (2016), 3794–3805.

[9] D. Friedman. 1993. The double auction market institution: A survey. In *The Double Auction Market: Institutions, Theories, and Evidence*. Routledge, 3–25.

[10] C. Hewitt, P. Bishop, and R. Steiger. 1973. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*. 235–245.

[11] George Iosifidis, Lin Gao, Jianwei Huang, and Leandros Tassiulas. 2013. An iterative double auction for mobile data offloading. In *Proceedings of the 11th International Symposium and Workshops on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt'13)*. IEEE, 154–161.

[12] George Iosifidis and Iordanis Koutsopoulos. 2010. Double auction mechanisms for resource allocation in autonomous networks. *IEEE J. Select. Areas Commun.* 28, 1 (2010).

[13] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Sec.* 1, 1 (2001), 36–63.

[14] Jiawen Kang, Rong Yu, Xumin Huang, Sabita Maharjan, Yan Zhang, and Ekram Hossain. 2017. Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using consortium blockchains. *IEEE Trans. Industr. Inf.* 13, 6 (2017), 3154–3164.

[15] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and privacy with payment-channel networks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 455–471.

[16] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. 2019. Sprites and state channels: Payment networks that go faster than lightning. In *International Conference on Financial Cryptography and Data Security*. Springer, 508–526.

[17] Zhongxing Ming, Shu Yang, Qi Li, Dan Wang, Mingwei Xu, Ke Xu, and Laizhong Cui. 2018. Blockcloud: A blockchain-based service-centric network stack. https://www.block-cloud.io/blockcloudtechnicalwhitepaper.pdf.

[18] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008). https://bitcoin.org/bitcoin.pdf.

[19] Lan N. Nguyen, Truc D. T. Nguyen, Thang N. Dinh, and My T. Thai. 2019. OptChain: Optimal transactions placement for scalable blockchain sharding. In *Proceedings of the IEEE 39th International Conference on Distributed Computing Systems (ICDCS'19)*. IEEE, 525–535.

[20] Truc D. T. Nguyen, Hoang-Anh Pham, and My T. Thai. 2018. Leveraging blockchain to enhance data privacy in IoT-based applications. In *Proceedings of the International Conference on Computational Social Networks*. Springer, 211–221.

[21] Simon Parsons, Marek Marcinkiewicz, Jinzhong Niu, and Steve Phelps. 2006. *Everything you Wanted to Know About Double Auctions, but were Afraid to (Bid or) ask*. Technical Report. Department of Computer and Information Science, Brooklyn College. http://www.sci.brooklyn.cuny.edu/~parsons/projects/mech-design/publications/cda.pdf.

[22] Muhammad Saad, Victor Cook, Lan Nguyen, My T. Thai, and Aziz Mohaisen. 2019. Partitioning attacks on bitcoin: Colliding space, time, and logic. In *Proceedings of the IEEE 39th International Conference on Distributed Computing Systems (ICDCS'19)*. IEEE.

[23] Truffle Suite. 2016. Ganache. Retrieved from https://www.trufflesuite.com/ganache.

[24] Truffle Suite. 2016. Sweet Tools for Smart Contracts. Retrieved from https://www.trufflesuite.com/.

[25] Yu-e Sun, He Huang, Kai Xing, Zhili Chen, Jianying Zheng, Hongli Xu, and Liusheng Huang. 2014. SPRITE: A novel strategy-proof multi-unit double auction scheme for spectrum allocation in ubiquitous communications. *Pers. Ubiq. Comput.* 18, 4 (2014), 939–950.

[26] Melanie Swan. 2015. *Blockchain: Blueprint for a New Economy*. O'Reilly Media, Inc.

[27] Subhasis Thakur, Barry P. Hayes, and John G. Breslin. 2018. Distributed double auction for peer to peer energy trade using blockchains. In *Proceedings of the 5th International Symposium on Environment-friendly Energies and Applications (EFEA'18)*. IEEE, 1–8.

[28] Jian Wang, Qianggang Wang, and Niancheng Zhou. 2018. A decentralized electricity transaction mode of microgrid based on blockchain and continuous double auction. In *Proceedings of the IEEE Power & Energy Society General Meeting (PESGM'18)*. IEEE, 1–5.

[29] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ether. Proj. Yell. Pap.* 151 (2014), 1–32.

[30] Suli Zou, Zhongjing Ma, and Xiangdong Liu. 2017. Resource allocation game under double-sided auction mechanism: Efficiency and convergence. *IEEE Trans. Automat. Contr.* 63, 5 (2017), 1273–1287.

Received November 2019; revised February 2020; accepted March 2020