Trustless Framework for Iterative Double Auction Based on Blockchain

Truc D. T. Nguyen(✉) and My T. Thai

Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611, USA
truc.nguyen@ufl.edu, mythai@cise.ufl.edu

Abstract. One of the major problems in current implementations of iterative double auction is that they rely on a trusted third party to handle the auction process. This imposes the risk of single point of failures and monopoly. In this paper, 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 trust each other, while minimizing the blockchain transactions. We provide a formal development of the framework and highlight the security of our design against adversaries.

Keywords: Blockchain · Iterative double auction · Trustless · State channel

1 Introduction

In recent years, following the great success of Bitcoin [1], the *blockchain* technology has emerged as a trending research topic in both academic institutes and industries associations. In a nutshell, blockchain can be seen as a decentralized database or digital ledger that contains append-only data blocks where each block is comprised of 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. In specific, it enables a trustless network where participants can transact although they do not trust one another. Moreover, a blockchain system may also employ the smart contracts technology to enable a wide range of applications that go beyond financial transactions [2]. In the context of blockchain, *smart contracts* are defined as self-executing and self-enforcing programs that are stored on chain. They are deployed to the blockchain system with publicly visible terms and conditions. Blockchain and smart contracts together have inspired many decentralized applications and stimulated scientific research in diverse domains [3–9].

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 [10]. A market can enable only

S. Chen et al. (Eds.): SecureComm 2019, LNICST 304, pp. 3–22, 2019.
https://doi.org/10.1007/978-3-030-37228-6_1

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 different between them is that an iterative double auction process has multiple, instead of one, iterations [11]. 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 in allocating divisible resources such as energy trading [7,12], mobile data offloading [13], or resource allocation in autonomous networks [14]. However, current implementations of double auction system require a centralized and trusted auctioneer to regulate the auction process. This results in the risk of single point of failures, monopoly, and privacy.

Although many research work has tried to develop a trading system combining the iterative double auction and blockchain [7,15], 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*. Although a naive mechanism to eliminate the trusted third-party is to implement the auctioneer in a blockchain smart contract, this results in high latency and transaction fees. To overcome this problem, we adopt the *state channel* technology [16] that enables the off-chain execution of smart contracts without changing the trust assumption.

Contribution. The key contribution of this work is the formal development of a novel decentralized and trustless framework for iterative double auction based on blockchain. With this framework, we are able to run existing double auction algorithms efficiently on a blockchain network without being suffered from the high latency of on-chain transactions. Specifically, we develop a Universally Composable (UC)-style model [17] for the double auction protocol and prove the security properties of our design using the simulation-based UC framework.

Organization. The remainder of this paper is organized in the following manners. We summarize the related work in Sect. 2. In Sect. 3, we discuss how blockchain and double auction can be combined. We will first present a straw-man design and then provide a high-level view of our framework. Section 4 presents the formal security definitions of our work. Then, we provide a formal specification of our framework in Sect. 5. Finally, Sect. 6 gives the concluding remarks.

2 Related Work

Double Auction Based on Blockchain. As blockchain is an emerging technology, there has been many research work addressing double auction with blockchain. Recently, Thakur et al. [18] 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 [19], 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 like energy or wireless spectrum allocation, these models greatly limit users' capability to utilize the products [20].

In [7] and [15], the authors propose blockchain-based energy trading using double auction. The auction mechanism is implemented as an iterative process which can be used for divisible goods. Although the system presented in these papers employ 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 has been many research effort on payment channels [21,22], the state channel has only emerged in recent years. State channel is a generalization of payment channel in which users can execute complex smart contracts off-chain while still maintains the trustless property. Dziembowski et al. [16] is the first work that present the formal specifications of state channels.

3 Double Auction with Blockchain

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 [23], which elicits hidden information about parties in order 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 $b_j^{(k)}$ 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. An example of such algorithm can be referred to [23]. 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 $b_j^{(k+j)}$ based on [23]
    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 which 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.

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 Fig. 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. In order 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 works, 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.

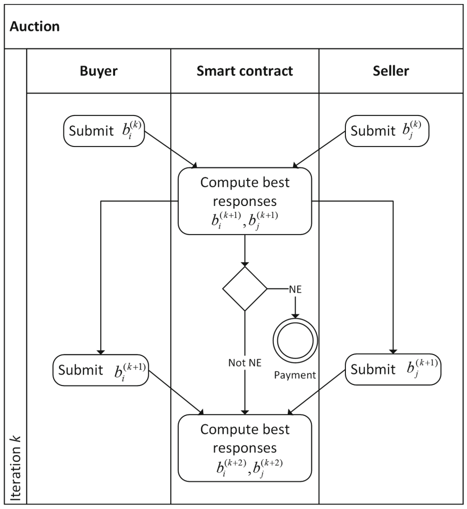

Fig. 1. Auction phase

2. High computational complexity on the smart contract which means that the blockchain will require high transaction fees.

In other words, the buyers and sellers are having the entire blockchain to process their auction.

3.3 Blockchain with State Channels

As the double auction process involves multiple iterations and has a fixed set of participants, state channel [16] is a proper solution to address the deficiencies of the straw-man design. Instead of processing the auction on-chain, 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.

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 operation in 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 $\mathcal{P} = \mathcal{B} \cup \mathcal{S} = \{p_1, p_2, ..., p_n\}$, 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,

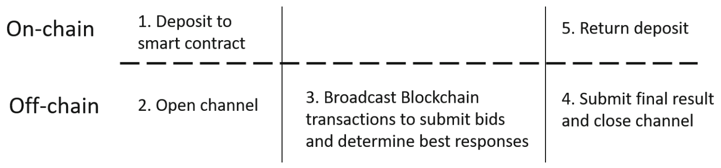
On-chain
1. Deposit to smart contract
5. Return deposit
Off-chain
2. Open channel
3. Broadcast Blockchain transactions to submit bids and determine best responses
4. Submit final result and close channel

Fig. 2. Double auction state channel

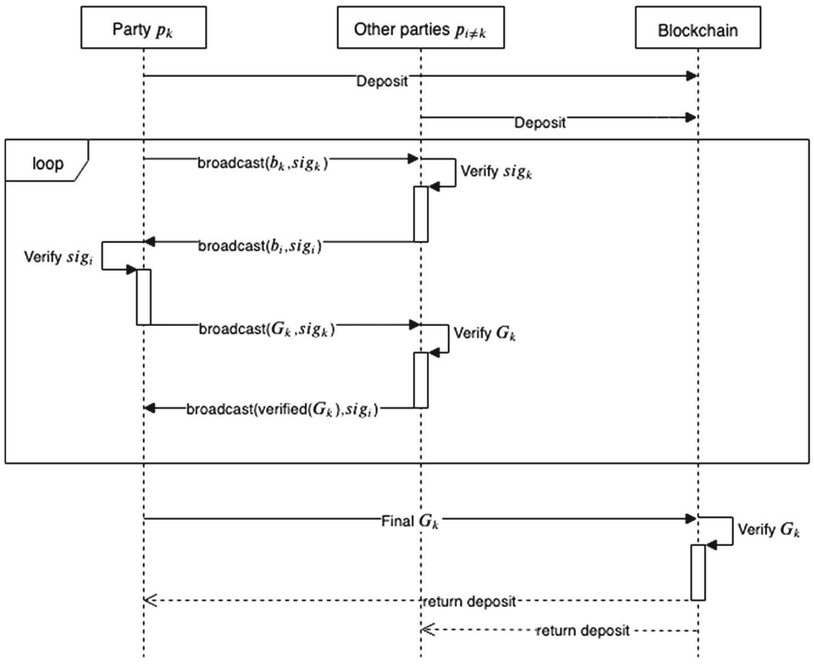

Fig. 3. Sequence diagram of the double auction process.

each party has to verify its 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, 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 Fig. 3.

As can be seen, the blockchain is invoked only two times and thus saves tons of transaction fees comparing 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, thus enhances the privacy.

4 Formal Definitions and Security Models

In this section, we describe the formal security definitions and models used in the construction of this framework. Before that, we establish some security and privacy goals for our system.

4.1 Security and Privacy Goals

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 a chosen message attack [24]. This signature scheme is currently being used in the Ethereum blockchain [2].
- Non-Repudiation: Once a bidder has submitted a bid, they must not be able to repudiate having made the relevant bid.
- Public Verifiability: All parties can be verified as having correctly followed the auction protocol.
- Robustness: The auction process must not be affected by invalid bids nor by participants not following the correct auction protocol.
- 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.

4.2 Our 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.

Assumptions and Notation. We denote $\mathcal{P} = \mathcal{B} \cup \mathcal{S} = \{p_1, p_2, ..., p_n\}$ 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 [16]). 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 fund 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.

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.

5 Double Auction State Channel

In this section, we describe the ideal functionality of our system that defines how a double auction process is operated using the state channel technology. Afterwards, we present the design of our protocol that realizes the ideal functionality.

5.1 Ideal Functionality

First, we define the ledger's ideal functionality $\mathcal{F}_{\mathcal{L}}$. Based on Sect. 4.2, the $\mathcal{F}_{\mathcal{L}}$ supports adding and subtracting one's balance, hence, we give the corresponding definition in Fig. 4.

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

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

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

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 which is a hash function of the bid and a random nonce. After that, all parties must submit the true bid that matches with the hash they sent before. Any party 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 which 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 advancing of the protocol to a great extent, 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, the functionality returns the deposit back to all parties' account.

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 $C_i^{(k)} = \mathcal{H}(b_i^{(k)}||r_i^{(k)})$ 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 $\mathcal{H}(R_i^{(k)}) \neq C_i^{(k)}$, remove p_i from $\mathcal{P}$ then stop.
    3. Compute best response G_k based on Algorithm 1
    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 $\mathcal{F}_{auction}$

5.2 Protocol for Double Auction State Channel

In this section, we will discuss in details the double auction protocol based on state channel that realizes the $\mathcal{F}_{auction}$. The protocol includes two main parts: (1) a Judge contact 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 Fig. 6 for the functionality of the Judge contract. Note that the contract $\mathcal{F}_{Judge}$ has a state variable *channel* which 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. 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. When the state channel is closed, the contract is now hold 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 π*.* In this section, we present the off-chain protocol π that operates among parties in a double auction process. In the same manner as $\mathcal{F}_{auction}$, the protocol π consists of four parts: (1) Create state channel, (2) Determine best response, (3) Revocation, and (4) Close state channel.

First, to create a new state channel, the environment sends a message $create()$ to one of the parties. Let's denote this initiating party as p_i. The detailed protocol is shown in Fig. 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, thus 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 Fig. 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, $\mathcal{F}_{Judge}$ will set $flag = \perp$ to conclude the dispute period. Furthermore, we also note that this procedure also supports eliminating any dishonest party that does not follow the protocol by setting the parameter p_r to that party. If a party p_i wants to eliminate a party p_r, it will need other parties, except p_r, to call the $Submit()$ procedure to remove p_r from $\mathcal{P}$.

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 $v \leq bestVersion$ the 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

Protocol π: 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 π: Create state channel

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

Party p_i

1. If $p_i = \perp$ 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. Procedure *Submit*

Next, in Fig. 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 this message will also broadcast their commitments. Then, p_i proceeds to broadcast the reveal $R_i^{(k)}$ of its bid and hence, other parties upon receiving this $R_i^{(k)}$ also broadcast their reveals. 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. In order to avoid losing the deposit, they must use the Revocation protocol described in Fig. 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, Fig. 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.

5.3 Security and Privacy Analysis

We denote $\text{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ as the outputs of the environment $\mathcal{E}$ when interacting with the adversary $\mathcal{A}$ and parties running the protocol π. From [17], we have the following definition:

Definition 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 $EXEC_{\pi,\mathcal{A},\mathcal{E}}$ and $EXEC_{\pi,\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.*

In this section, we will prove the following theorem:

Theorem 1. *Under the assumptions given in Sect. 4.2, the protocol π UC-realizes the ideal functionality $\mathcal{F}_{auction}$ in the $(\mathcal{F}_{judge}, \mathcal{F}_{\mathcal{L}})$-hybrid model.*

Proof. The main goal of this analysis is 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 messages exchanged between any entities as well as 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 Canneti [17], the proof strategy consists of 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 $\mathcal{F}_{auction}$. Hence, the simulator will maintain a copy of the hybrid world internally. 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 omit these operations from the description of the simulator. Since $\mathcal{S}$ locally runs a copy of the hybrid world, $\mathcal{S}$ knows the behavior of the

Protocol π: Determine best response

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

1. Broadcast $C_i^{(k)} = \mathcal{H}(b_i^{(k)} || r_i^{(k)})$ 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 $C_j^{(k)} = \mathcal{H}(b_j^{(k)} || r_j^{(k)})$ 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 $\mathcal{H}(R_l^{(k)}) \neq C_l^{(k)}$ 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 $\mathcal{H}(R_j^{(k)}) \neq C_j^{(k)}$ 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 π: Determine best response

Protocol π: 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 π: Revocation

Protocol π: 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 π: Close state channel

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. 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 analyze 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 Fig. 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 analyze 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 $created()$ 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 analyze 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. We define $S\_Submit()$ as the simulator of the procedure $Submit()$ in the ideal world. Let p_i be the party that calculates the best responses. In each iteration k, we analyze 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 $\mathcal{S}$ doesn't 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.

6 Conclusion

In this paper, 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 the aid of the state channel technology, we were able to reduce 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.

Acknowledgment. This paper is partially supported by DTRA HDTRA1-14-1-0055 and NSF CNS-1814614.

References

1. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
2. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Proj. Yellow Pap. **151**, 1–32 (2014)
3. Nguyen, L.N., Nguyen, T.D., Dinh, T.N., Thai, M.T.: OptChain: optimal transactions placement for scalable blockchain sharding. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 525–535. IEEE (2019)
4. Saad, M., Cook, V., Nguyen, L., Thai, M.T., Mohaisen, A.: Partitioning attacks on bitcoin: colliding space, time, and logic. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE (2019)
5. Azaria, A., Ekblaw, A., Vieira, T., Lippman, A.: MedRec: using blockchain for medical data access and permission management. In: International Conference on Open and Big Data (OBD), pp. 25–30. IEEE (2016)
6. Dinh, T.N., Thai, M.T.: AI and blockchain: a disruptive integration. Computer **51**(9), 48–53 (2018)
7. Kang, J., Yu, R., Huang, X., Maharjan, S., Zhang, Y., Hossain, E.: Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using consortium blockchains. IEEE Trans. Ind. Inform. **13**(6), 3154–3164 (2017)
8. Aitzhan, N.Z., Svetinovic, D.: Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams. IEEE Trans. Dependable Secure Comput. **15**(5), 840–852 (2016)
9. Nguyen, T.D.T., Pham, H.-A., Thai, M.T.: Leveraging blockchain to enhance data privacy in IoT-based applications. In: Chen, X., Sen, A., Li, W.W., Thai, M.T. (eds.) CSoNet 2018. LNCS, vol. 11280, pp. 211–221. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04648-4_18
10. Friedman, D.: The double auction market institution: a survey. Double Auction Market Inst. Theor. Evid. **14**, 3–25 (1993)
11. Parsons, S., Marcinkiewicz, M., Niu, J., Phelps, S.: Everything you wanted to know about double auctions, but were afraid to (bid or) ask (2006)
12. Faqiry, M.N., Das, S.: Double-sided energy auction in microgrid: equilibrium under price anticipation. IEEE Access **4**, 3794–3805 (2016)
13. Iosifidis, G., Gao, L., Huang, J., Tassiulas, L.: An iterative double auction for mobile data offloading. In: 2013 11th International Symposium and Workshops on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), pp. 154–161. IEEE (2013)
14. Iosifidis, G., Koutsopoulos, I.: Double auction mechanisms for resource allocation in autonomous networks. IEEE J. Sel. Areas Commun. **28**(1), 95–102 (2010)
15. Wang, J., Wang, Q., Zhou, N.: A decentralized electricity transaction mode of microgrid based on blockchain and continuous double auction. In: 2018 IEEE Power & Energy Society General Meeting (PESGM), pp. 1–5. IEEE (2018)
16. Dziembowski, S., Faust, S., Hostáková, K.: General state channel networks. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS 2018, pp. 949–966. ACM, New York (2018). https://doi.org/10.1145/3243734.3243856
17. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings 2001 IEEE International Conference on Cluster Computing, pp. 136–145. IEEE (2001)

18. Thakur, S., Hayes, B.P., Breslin, J.G.: Distributed double auction for peer to peer energy trade using blockchains. In: 2018 5th International Symposium on Environment-Friendly Energies and Applications (EFEA), pp. 1–8. IEEE (2018)
19. Ming, Z., et al.: Blockcloud: a blockchain-based service-centric network stack
20. Sun, Y.-E., et al.: SPRITE: a novel strategy-proof multi-unit double auction scheme for spectrum allocation in ubiquitous communications. Pers. Ubiquit. Comput. **18**(4), 939–950 (2014)
21. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 455–471. ACM (2017)
22. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: payment networks that go faster than lightning. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 508–526. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32101-7_30
23. Zou, S., Ma, Z., Shao, Y., Ran, L., Liu, X.: Efficient and dynamic double auctions for resource allocation. In: 2016 IEEE 55th Conference on Decision and Control (CDC), pp. 6062–6067. IEEE (2016)
24. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Secur. **1**(1), 36–63 (2001)