Auctionity Yellow Paper

Pascal Lafourcade[1], Mike Nopère[2], Daniela Pizzuti[2,3] and Étienne Roudeix[2]

[1]LIMOS, University Clermont Auvergne, Aubière, France
[2]Domraider, Aubière, France
[3]UNESP, Bauru, Brazil; ENSIMAG, Aubière, France

July 30, 2018

Abstract

Auctionity is a blockchain based English auction protocol. The components of the protocol and the exchanged messages for each action, as the creation of an auction or a bid, are described in this paper. Then, the desired security properties are presented.

Keywords: Blockchain, English Auction, Security.

1 Introduction

An auction is a method to sell products in which a seller proposes goods or services for sale, and bidders present the amount they are willing to pay for it. Auctions have been used since Antiquity, reportedly starting in Babylon as early as 500 BC. Today, both the range and the value of objects sold by this method have grown to astonishing proportions [Kri09]. Over the years, several kinds of auction have been invented. The most known is *English auction*, in which the bidder who offers the highest price wins the auction. *Vickery auction* is an auction type where the highest bid wins but the winner only pays the second-highest bid. *Dutch auction* is a mechanism where the seller sets up an initial price and the price is lowered until a bidder accepts the current price. *Sealed Bid auction* is a form of auction where bids are not public. Each bidder can bid only one value because all bidders simultaneously submit sealed bids and the highest bid wins the auction.

The easy access to the Internet, through computers and mobile devices, and the birth of modern cryptography in the 80's, made the use of digital systems to buy or sell products, a common practice. Following this trend, auctions also take place online, what is called *e-auctions*. The market of e-auctions is huge, as demonstrated by websites like eBay, which has more than 170 millions active buyers in 2018 [eba18]. E-auction systems often apply cryptographic mechanisms to be secure, but they use a centralized authority to manage all the transactions between sellers and bidders.

With Bitcoin [Nak09] and Ethereum [Woo18], the blockchain technologies are nowadays a key component of the modern digital world. Mainly, a blockchain is a distributed and decentralized ledger with some specific properties ensured by some cryptographic primitives. One of those properties is that it is not possible to change data stored in the blockchain without the consensus of the peers. This property is clearly a key feature to auctions based on the blockchain technology. Our goal is to design an English auction protocol that runs on a blockchain. This protocol is named Auctionity Protocol and the e-auction system based on it is called Auctionity.

Contributions. This paper has the following goals:

- Presenting *Auctionity*[1], our e-auction system based on the *Ethereum*[2] blockchain.
- Describing the different mechanisms and protocols used in Auctionity.
- Defining the relevant security properties for Auctionity.

Related Work. The closest work to our is STRAIN (Secure aucTions foR blockchAINs) proposed in [BK18]. In this paper, the authors propose a sealed bid auction based on blockchain and cryptographic primitives like Multi-Party Computation (MPC) and Zero-Knowledge Proofs (ZKP). They provide security proofs to guarantee bid confidentiality against fully-malicious parties. Our aim is different, since we want to design an English auction system where the bids are public. It leads us to a different protocol.

There exist several other e-auction protocols among [Bra06, NPS99, OM01, JS02, CPS07, PBDV02, Sak00, DDL15]. However, most of them do not use blockchain and aim at providing anonymity mechanisms for the bids. Here again, our aim is different.

Outline. In Section 2, we describe the structure of Auctionity. In Section 3, we provide the *Gas*[3] cost of our protocol. In Section 4, we give the security properties for Auctionity before concluding.

2 Auctionity Description

We start by presenting the general structure of Auctionity. Then, we introduce some notations. After, we present our Auctionity protocol.

2.1 General Structure

Auctionity uses Ethereum and it is composed of the following principals:

- Ethereum Live Net (ELNET or Γ): The Ethereum public blockchain, where the following Smart Contract (SC) is running:

[1] `https://www.auctionity.com` and `https://github.com/auctionity/auctionity`

[2] `https://ethereum.org`

[3] Gas is the pricing value required to execute a transaction on the Ethereum blockchain platform.

  - Deposit (D): A SC responsible for holding bidders deposit and processing withdrawal and payment demands on ELNET.

- Auctionity Network (ACNET or Ω): The private blockchain owned by Auctionity, where the following SCs are running:
  - Treasurer (T): A SC responsible for holding bidders deposit and processing withdrawal demands on ACNET.
  - Auction (A): A SC responsible for processing auctions. However, while the previous SCs are instantiated only once, for each auction there is an Auction SC instance, created by the seller.
- Oracle (O): The Auctionity server that transfers information between ELNET and ACNET.
- Bidder (B): A user that participates in an auction on ACNET.
- Seller (S): A user that auctions a product on ACNET.

In Figure 1, we show the exchanges between these principals in Auctionity. We notice that bidders and sellers directly interact with ELNET and ACNET, while ELNET and ACNET communicate between themselves via the Oracle, that, by allowing their communication, plays the role of a trusted third party.

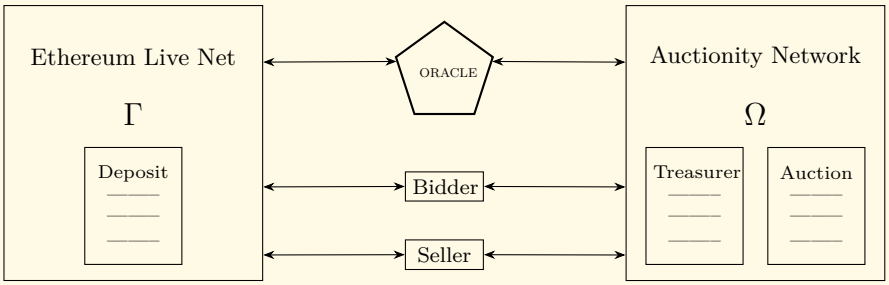

Figure 1: General Structure of Auctionity.

2.2 Notations

A user is denoted by u. It can be a seller, a bidder or the Oracle. Each user has a pair of ECDSA (Elliptic Curve Digital Signature Algorithm) keys, denoted by pk_u for the public key, and, sk_u for the secret key. Each Smart Contract or user has an address (ad_{SC} or ad_u) which is their identity on the system, that, in the case of users is based in their ECDSA public key and in the case of SCs is based in the address of the SC's creator and the value of her internal counter, denoted by co_u, at the moment of the contract creation. We also consider a hash function, denoted by $\mathrm{H}(m)$, and a signature algorithm, denoted by $\mathrm{Sig}_{sk_u}(m)$ for signing a message m with the secret key sk_u. The notation SIG_u represents that a message is sent with its signature, as shown bellow:

$$\mathrm{SIG}_u(m) = (m \parallel \mathrm{SIG}_{sk_u}(\mathrm{H}(m))$$

In Table 1, we list all the notations used to describe Auctionity.

Notation	*Description*
ACNET or Ω	Auctionity Network
ELNET or Γ	Ethereum Live Network
B	Bidder
O	Oracle
S	Seller
L	Current leader of an auction
W	Winner of an auction
A	Auction Smart Contract
D	Deposit Smart Contract
T	Treasurer Smart Contract
EVM	Ethereum Virtual Machine
SC	Smart Contract
AEV_u	Auction End Voucher
P_u	Parameters sent by u to Ω and Γ
WV_u	Withdraw Voucher
H	Hash function
Sig	Signature function
SIG	Message and the signature of its hash
ad_u	Address of u
am_u	Amount of u
co_u	Counter of u
de_u	Deposit balance of u
in_u	Information of u
m	Message
pk_u	Public key of u
sk_u	Secret key of u
ts_u	Timestamp of u

Table 1: Notations, where u is a user.

2.3 Auctionity Protocol

Interacting with Γ and Ω a user can bid, withdraw her deposit (amount the bidder has on Ω used as *Payment Guarantee*[4] and create an auction or end it (to obtain the payment locked on the Ω Treasurer SC). For each of those actions there is a protocol:

- Create Auction Protocol: It is a protocol that allows a seller to create a new instance of an Auction SC.
- Close Auction Protocol: It is a protocol that allows a seller to receive the winning amount of her auction.
- Bid Protocol: It is a protocol that allows a bidder to bid on an auction.

[4]Payment Guarantee is a functionality offered by the system that ensures sellers that they will receive the winning amount of their auctions. It is done thanks to the deposit made by bidders, that is blocked when they bid until another bid is accepted.

- Withdraw Deposit Protocol: It is a protocol that allows a bidder to get her deposit back to her account on ELNET.

In order to communicate with ELNET and ACNET, a user should respect the SC formalism. It is why the messages sent by u to ELNET and ACNET carry the following Ethereum parameters:

- co_u: It denotes the number of transactions sent by the sender. We notice that in [Woo18], this counter is called *nonce*.
- $gasPrice_u$: The price to be paid by u per gas unit.
- $gasLimit_u$: The limit of gas to be used for the transaction execution.
- $value_u$: The number of *Wei*[5] to be sent from u's account to the recipient or new contract.
- to_u: It is the recipient of the message sent by u. It corresponds to an address ad of a SC or of another user.
- $\mathrm{SIG}_{sk_u}(\mathrm{H}(m))$: The transaction signature, where m contains the previous parameters plus specific content of each message.

We use the notation P_u to represent the following set of parameters, sent by users in their messages to ACNET and ELNET:

$$P_u = (co_u \parallel gasPrice_u \parallel gasLimit_u \parallel value_u \parallel to_u) .$$

We are now ready to describe the four protocols of Auctionity.

2.4 Create Auction Protocol

To create an auction, a seller S sends a signed message to ACNET, with the Auction SC binary code, denoted by "$\mathrm{EVMCode}_{SC}$" and the auction information, denoted by *in*:

- *title*: Bit string chosen by S to be the title of her auction.
- *startAmount* : Value chosen by S to be the minimum bid amount of her auction.
- *startTime*: Date chosen by S to be start time of her auction.
- *endTime*: Date chosen by S to be the end time of her auction.
- *bidIncrement*: Value chosen by S to be the minimum bid increment a bidder will be able to make to her auction.
- *antiSnippingTriggerPeriod*: Time value chosen by S as the period of time before *endTime* during which the anti snipping is triggered. Its maximum value is 194 days, 4 hours, 20 minutes and 16 seconds, which corresponds to the maximum size of the type *uint24*, in seconds.

[5] Wei is the smallest money unit of Ethereum, which is equal to 10^{-18} Ether.

- *antiSnippingDuration*: Time value chosen by S to be the duration of the anti snipping. Its maximum value is 18 hours, 12 minutes and 16 seconds, which corresponds to the maximum size of the type *uint16*, in seconds.

The parameter *endTime* is stored as *originalEndTime*. The *endTime* is then updated as in the example below:

$originalEndTime = endTime =$ 07/08/2018 15:00:00,

$antiSnippingTriggerPeriod =$ 00:10:00,

$antiSnippingDuration =$ 30:00:00.

To trigger the anti snipping, a bid needed to be accepted between 07/08/2018 14:50:01 and 07/08/2018 15:00:00, inclusive. Supposing that a bid was accepted at 14:55:00, the following value would be updated:

$endTime =$ 07/08/2018 15:25:00.

The Create Auction Protocol is represented in Figure 2 and works as follows:

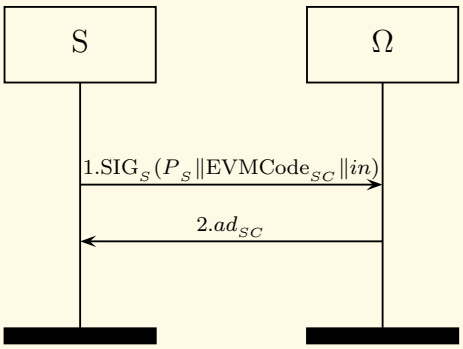

Figure 2: Create auction.

1. S → Ω: SIG$_S$(P_S ‖ EVMCode$_{SC}$ ‖ in). A seller S signs with her secret key the following parameters: P_S, EVMCode$_{SC}$ and in. Then, S sends it to ACNET. Finally, ACNET creates an Auction SC with these parameters.

2. Ω → S: ad_{SC}. ACNET sends to S, through ACNET's WebSocket[6], the address of her Auction SC as a confirmation that it has been created. At this point, with the Auction SC instance written on the blockchain, it is not possible to cancel the auction.

2.5 Close Auction Protocol

Anyone can request to close an auction. As ACNET can not emit an event by itself when the auction end time is reached, S is expected to call the function

[6]WebSocket is a protocol for the connection between a *http* client and a server. It is used by Auctionity because it allows Ethereum nodes to broadcast information to anyone who listens to it, so the users do not need to constantly interrogate the network.

that closes her auction. However, as in the case if only S could do it, S could "cancel" an auction by never ending it, this action can be taken by anybody. If the original end time, plus the triggered anti snipping period, is not reached, the end auction request cannot be performed.

When an end auction request is received by the Auction SC instance, an Auction End Voucher (AEV), that is a bit string issued by the Oracle with the auction results, as the winner address and winning amount, is added to the Auction SC. It is then submitted by the seller to ELNET in order to withdraw her auction winning amount, denoted by am_W where W is the bidder who won the auction.

The process of ending an auction is described in Figure 3 and works as follows:

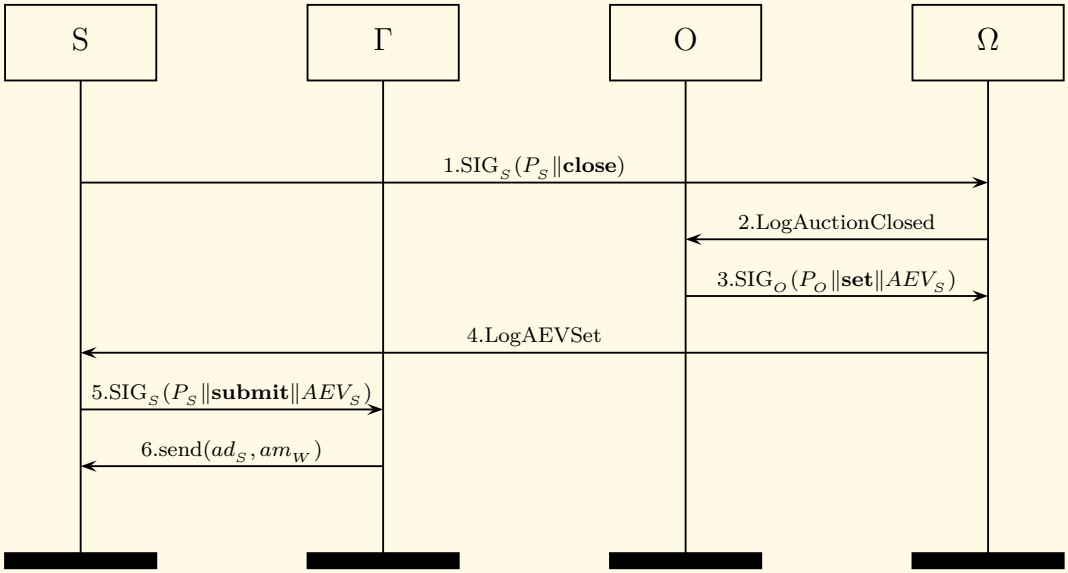

Figure 3: Close auction.

1. S → Ω: SIG$_S$(P_S ‖ **close**). Anyone can call the Auction SC function[7] **close** to close an auction, but we consider that this call is made by a seller S who created this auction. S signs with her secret key the following parameters: P_S and **close**. Then, S sends it to ACNET. Finally, as a result of the call of the Auction SC function **close**, ACNET emits the Ethereum event[8], LogAuctionClosed, which indicates that this Auction SC instance received a valid close auction request.

2. Ω → O: LogAuctionClosed. The Oracle O, listening to ACNET's WebSocket, gets the information of the Ethereum event, LogAuctionClosed, triggered by the Auction SC function **close**, of the Auction SC instance created by S on ACNET.

[7]SC function names are written in bold letters and variables are written in italic.

[8]An Ethereum event is an event emitted as result of a function computation. Every event is broadcasted through WebSockets.

3. O $\rightarrow \Omega$: $\text{SIG}_{O}(P_{O} \parallel \textbf{set} \parallel AEV_{S})$. O signs with its secret key the following parameters: P_{O}, **set** and AEV_{S}. Then, O sends it to ACNET. Finally, as a result of the call of the Auction SC function **set** with the parameter AEV_{S}, ACNET emits the Ethereum event LogAEVSet, which indicates that the AEV_{S} was set to S's Auction SC instance.

4. $\Omega \rightarrow$ S: LogAEVSet. S, listening to ACNET's WebSocket, gets the information of the Ethereum event, LogAEVSet, triggered by the Auction SC function **set**, of her Auction SC instance. Then, S gets her AEV_{S}.

5. S $\rightarrow \Gamma$: $\text{SIG}_{S}(P_{S} \parallel \textbf{submit} \mid AEV_{S})$. S, provided with her AEV_{S}, signs with her secret key the following parameters: P_{S}, **submit** and AEV_{S}. Then, S sends it to ELNET. Finally, as a result of the call of the Deposit SC function **submit**, with the parameter AEV_{S}, ELNET emits the Ethereum event LogAEVSubmitted, which indicates that ELNET received a valid AEV_{S} from S.

6. $\Gamma \rightarrow$ S: $\text{send}(ad_{S},\ am_{W})$ - The ELNET Deposit SC, provided with a valid AEV_{S}, uses the Solidity[9] function **send** to send the winning amount am_{W}, to S's address, denoted by ad_{S}.

2.6 Bid Protocol

Acting as a bidder, a user makes deposits on ELNET that are valid on ACNET thanks to the Oracle. The bidder is able to bid an amount am_{B} if it is smaller or equal her current deposit balance de_{B}. This is required by the Payment Guarantee feature in order to secure the payment to the seller.

The action of depositing (1 to 4) and bidding (5 and 6) that compose the Bid Procotol are outlined in Figure 4 and the steps for it are the following:

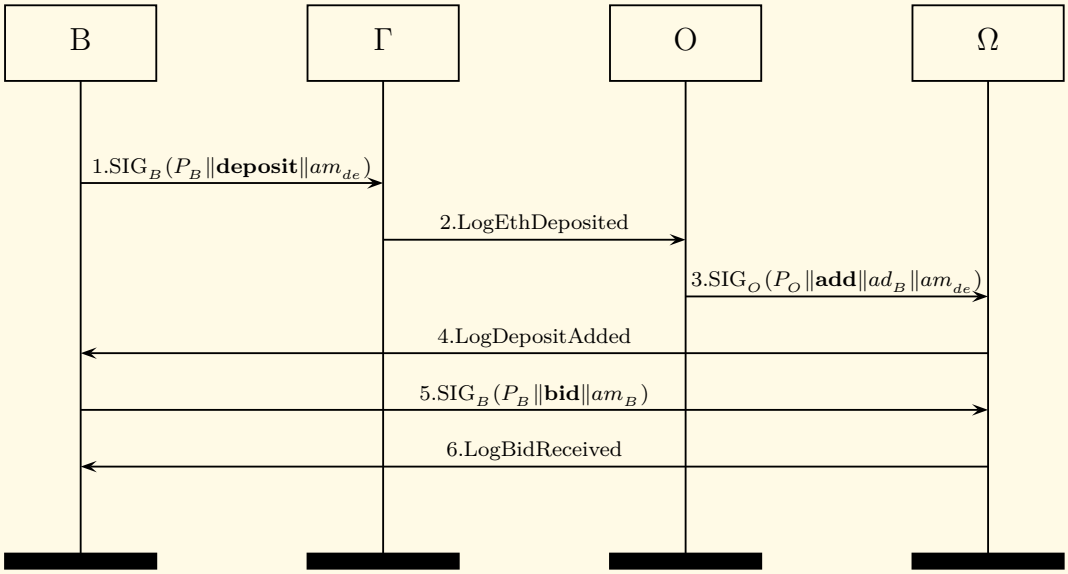

Figure 4: Bid.

[9]Solidity is a programming language for writing Ethereum Smart Contracts.

1. B $\rightarrow \Gamma$: $\text{SIG}_{B}(P_{B} \parallel \textbf{deposit} \parallel am_{de})$. A bidder B signs with her secret key the following parameters: P_{B}, **deposit** and am_{de}. Then, B sends it to ELNET. Finally, as a result of the call of the Deposit SC function **deposit**, with the parameter am_{de}, ELNET emits the Ethereum event LogEthDeposited, which indicates that a deposit of amount am_{de} was made by B and added to B's deposit balance, denoted by de_{B}.

2. $\Gamma \rightarrow$ O: LogEthDeposited. The Oracle O, listening to ACNET's WebSocket, gets the information of the Ethereum event, LogEthDeposited, triggered by the Deposit SC function **deposit**.

3. O $\rightarrow \Omega$: $\text{SIG}_{O}(P_{O} \parallel \textbf{add} \parallel ad_{B} \parallel am_{de})$. O signs with its secret key the following parameters: P_{O}, **add**, ad_{B} and am_{de}. Then, O sends it to ACNET. Finally, as a result of the call of the Treasurer SC function **add**, with the parameters ad_{B} and am_{de}, ACNET emits the event, LogDepositAdded, which indicates that the amount am_{de} was added to B's deposit balance on the Treasurer SC.

4. $\Omega \rightarrow$ B: LogDepositAdded. B, listening to ACNET's WebSocket, gets the information of the Ethereum event, LogDepositAdded, triggered by the Treasurer SC function **add**. Then, B gets the current balance of her deposit.

5. B $\rightarrow \Omega$: $\text{SIG}_{B}(P_{B} \parallel \textbf{bid} \parallel am_{B})$. B signs with her secret key the following parameters: P_{B}, **bid** and am_{B}. Then, B sends it to ACNET. Finally, as a result of the call of the Auction SC function **bid**, with the parameter am_{B}, ACNET emits the Ethereum event, LogBidReceived, which indicates that B's bid of amount am_{B} on an Auction SC instance ad_{SC} (the recipient of the message) was received by ACNET.

6. $\Omega \rightarrow$ B: LogBidReceived. B, listening to ACNET's WebSocket, gets the information of the Ethereum event, LogBidReceived, triggered by the function **bid** of the Auction SC instance ad_{SC}. Then, B gets the status of her bid (accepted or rejected).

Each Auction SC instance stores the address of the leader, denoted by ad_{L}, as leader is denoted by L, and its bid amount. The Auction SC is also responsible for checking the validity of each bid. This verification follows the algorithm given in Figure 5, where:

- $\text{Sig}_{sk_{B}}$ *matches* ad_{B}: Is the transaction signed with a secret key that corresponds to the bidder address (ad_{B})?

- $ts_{start} \leq ts_{cur} \leq ts_{end}$: Is the block timestamp (ts_{block}) bigger or equal to the *startTime* (ts_{start}) of the auction and smaller or equal to the *endTime* (ts_{end}) of the auction?

- $ad_{B} \neq ad_{S}$: Is the bidder address (ad_{B}) different of the sellers (ad_{S})?

- LogBidReceived $\neq \emptyset$: Is the auction bid history empty?

- $am_{B} > am_{min}$: Is the bid amount (am_{B}) bigger than the value of *minimumAmount* (am_{min}) and is it a multiple of the *bidIncrement* (am_{inc}), set by the seller?

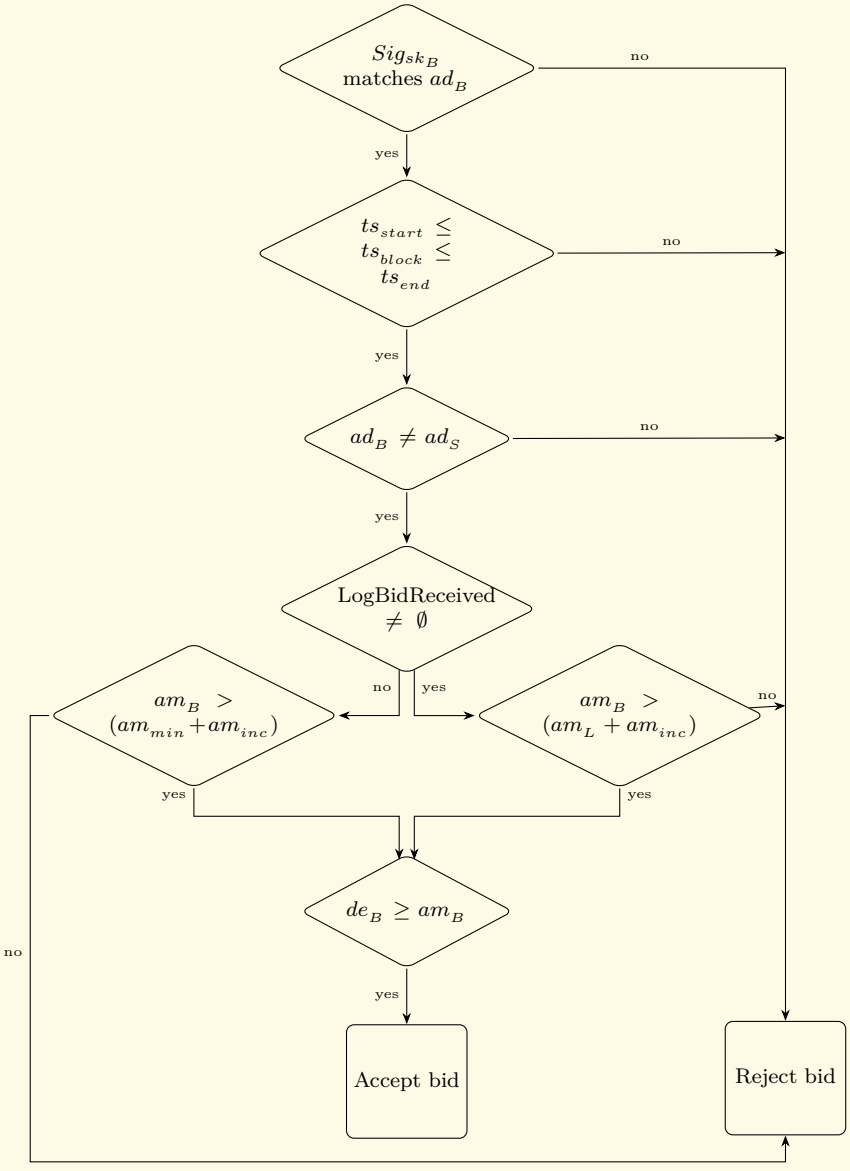

Figure 5: Checking the validity of a bid.

- $am_{_B} > am_{_L}$: Is the bid amount bigger than the value of *leaderAmount* (am_L) and is it a multiple of the *bidIncrement* (am_{inc}), set by the seller?
- $de_{_B} \geq am_{_B}$: Is the bidder deposit ($de_{_B}$) on ACNET bigger or equal the payment guarantee, which is equal to the bid amount, required by the auction?

At the end of the auction, the bidder address stored in the Auction SC variable $ad_{_L}$ is the winner, and the variable $am_{_L}$ is the winning amount.

$$\begin{aligned} ad_{_W} &= ad_{_L}, \\ am_{_W} &= am_{_L}. \end{aligned}$$

2.7 Withdraw Deposit Protocol

Another action made by a user as a bidder is to withdraw her deposit, which corresponds to have the Ether that she previously deposited but did not use, sent back to her address on ELNET. When a bidder wants to have her money back, she communicates with ACNET to request a Withdrawal Voucher ($WV_{_B}$), which is a bit string issued by the Oracle after getting the information of a new valid withdrawal request, wich means, of an amount smaller or equal to B's current balance, made to the Treasurer SC. It is then submitted by the bidder to ELNET in order to withdraw the amount $am_{_{wi}}$.

Figure 6 shows the withdrawing process that is described as follows:

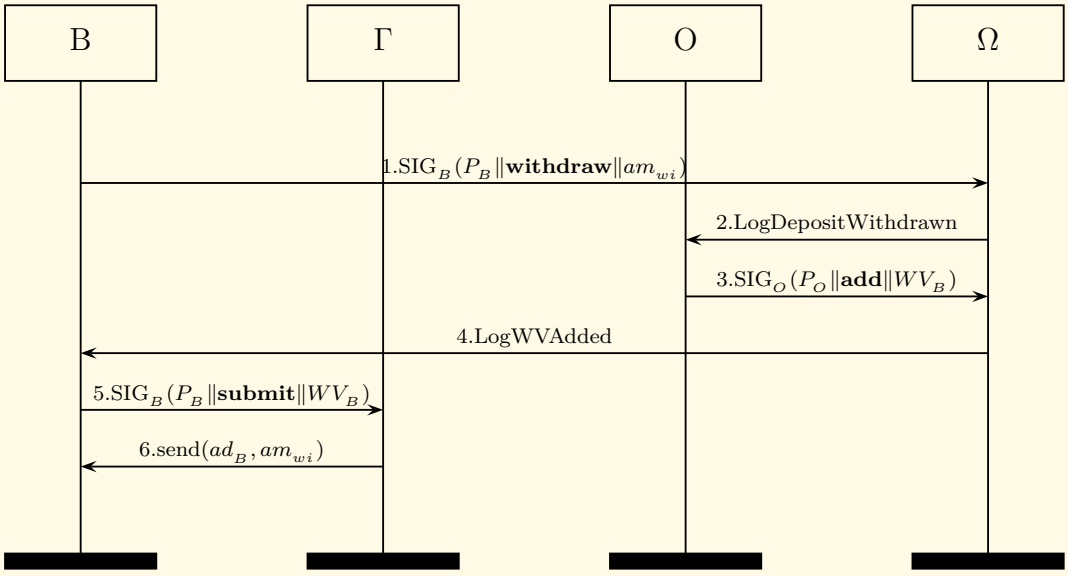

Figure 6: Withdrawal deposit.

1. B → Ω: SIG$_{_B}$($P_{_B}$ ‖ **withdraw** ‖ $am_{_{wi}}$). A bidder B signs with her secret key the following parameters: $P_{_B}$, **withdraw** and $am_{_{wi}}$. Then, B sends it to ACNET. Finally, as a result of the call of the Treasurer SC function **withdraw**, with the parameter $am_{_{wi}}$, ACNET emits the event, LogDepositWithdrawn, which indicates that the Treasurer SC received a withdrawal request of amount $am_{_{wi}}$ from B.

2. $\Omega \rightarrow$ O: LogDepositWithdrawn. The Oracle O, listening to ACNET's WebSocket, gets the information of the Ethereum event, LogDepositWithdrawn, triggered by the Treasurer SC function **withdraw**.

3. O $\rightarrow \Omega$: $\text{SIG}_{O}(P_{O} \parallel \textbf{add} \parallel WV_{B})$. O signs with its secret key the following parameters: P_{O}, **add** and WV_{B}. Then, O sends it to ACNET. Finally, as a result of the call of the Treasurer SC function **add**, with the parameter WV_{B}, ACNET emits the event LogWVAdded, which indicates that a Withdrawal Voucher was added for B.

4. $\Omega \rightarrow$ B: LogWVAdded. B, listening to ACNET's WebSocket, gets the information of the Ethereum Event, LogWVAdded, triggered by the Treasurer SC function **add**. Then, B gets her WV_{B}.

5. B $\rightarrow \Gamma$: $\text{SIG}_{B}(P_{B} \parallel \textbf{submit} \parallel WV_{B})$. B signs with her secret key the following parameters: P_{B}, **submit** and WV_{B}. Then B sends it to ELNET. Finally, as a result of the call of the Deposit SC function **submit**, with the parameter WV_{B}, ELNET emits the Ethereum event LogWVSubmitted, which indicates that ELNET received a valid WV_{B} from B.

6. $\Gamma \rightarrow$ B: $\text{send}(ad_{B},\ am_{wi})$. The ELNET Deposit SC, provided with a valid WV_{B}, uses the Solidity function **send** to send the withdrawal amount am_{wi}, to B's address, denoted by ad_{B}.

3 Gas Cost and Block Time

The cost of each action, measured with the Geth function eth_estimateGas, is shown in Table 2, where B_{1} is the first bidder to bid on an auction and B_{n} is any other bidder:

			Payer		
			ACNET	ELNET	
Protocol	Net	Function and Caller	Auctionity	Bidder	Seller
Create auction	Ω	Deploy SC by S	2,870,670	0	0
Close auction	Ω	**close** by O	55,950	0	0
	Ω	**set** by O	196,777	0	0
	Γ	**submit** by S	0	0	112,921
Bid	Γ	**deposit** by B_{1}	0	85,476	0
	Γ	**deposit** by B_{1}	0	30,421	0
	Ω	**add** by O	107,839	0	0
	Ω	1_{st} **bid** by B_{1}	167,672	0	0
	Ω	1_{st} **bid** by B_{n}	136,081	0	0
	Ω	n_{th} **bid** by B_{n}	67,994	0	0
	Ω	n_{th} **bid** by B_{1}	80,464	0	0
Withdrawal	Ω	**withdraw** by B	30,160	0	0
	Ω	**add** by O	247,071	0	0
	Γ	**submit** by B	0	111,736	0

Table 2: Cost of SC function calls in gas.

Users only pay for actions that take place on ELNET, which means, deposits and withdrawals. Also, in order to accept a large number of transactions on each block, the ACNET block gas limit is 4,503,599,627,370,496 while the ELNET block gas limit tends to 8,000,000. The time period between 2 blocks is 1 second on ACNET and 14 seconds on ELNET. The Oracle waits 8 blocks to confirm a deposit made on ELNET. So, a deposit takes place in 112 seconds after included in a block on ELNET. Considering that the deposit can be withdrawn at any time, and in order to avoid the need to wait for the deposit validation time before bidding, it is recommended for bidders to make deposits of enough amount for the auctions they intend to participate.

4 Security Properties

As Auctionity is composed of 4 protocols, each one of them has its own properties. For the Create Auction Protocol, those properties are to be defined with the development of the Delivery Guarantee feature, a functionality to be offered by the system, that has the goal of ensuring bidders that they will receive the product sold on the auctions they won. This will be done thanks to the transfer of a crypto asset, for instance, a *Non-fungible Token*[10] (NFT), made by sellers to the Deposit SC, that will blocked until the end of the auction, when it is then transfered to the winner.

The Auction End Voucher Validity property applies to the Close Auction Protocol.

Auction End Voucher Validity. As ELNET is not aware of the transactions processed on ACNET, the AEV_S is used to carry relevant information of the auctions to ELNET. So, to ensure that this information is valid, the property of Auction End Voucher Validity establishes that, for any AEV_S submitted on ELNET, the contents of the AEV_S need to be enough to prove to the Deposit SC that the winner address ad_W and the amount of its winning bid am_B correspond to a bid signed by W to the Auction SC instance to which the AEV_S was issued.

The property that applies to the Withdrawal Deposit Protocol is Withdrawal Voucher Validity.

Withdrawal Voucher Validity. As ELNET is not aware of the transactions processed on ACNET, the WV_B is used to carry the withdrawal requests to ELNET. So, to ensure that this information is valid, the property of Withdrawal Voucher Validity establishes that for any Withdrawal Voucher submitted on ELNET, the contents of the WV_B are enough to prove to the Deposit SC that the bidder address corresponds to the address that signed the Withdrawal Voucher request and this request is of the same amount as the one contained in the WV_B.

The Bid Protocol process the auctions, so, security properties for auctions apply to this protocol. Auction properties were generically defined on [DLL13] and [DJL13] and based on those definitions, for Auctionity, five properties are

[10] A Non-fungible Token is cryptographic token that represents something unique, and so, it is not interchangeable.

pertinent, given the blockchain peculiarities and the type of auction (English auction) that Auctionity implements. Those properties were remodeled in order to fit better the Auctionity protocol and are presented in this chapter.

Highest Price Wins. The Highest Price Wins property establishes that the bidder who submitted the highest valid bid is the one who wins the auction. If we have an honest bidder B_1 who submits the highest bid, an attacker that has completely corrupted another bidder B_2 must be unable to win the auction on his behalf on a bid lower than B_1's.

Non-Cancellation. As in an English auction all bids count to the result, the Non-Cancellation property for Auctionity assumes that the winning amount is correct. So, if a bidder B_i submitted an accepted bid she must be the leader until the acceptance of a higher bid. As a result, the bidder who submitted the last accepted bid must win the auction. If, however there is the possibility that even though her bid was accepted she did not win, this would mean that someone, an intruder or the bidder herself, was able to cancel the bid even after acceptance.

Non-Repudiation. If it was possible that a bidder would win without submitting the winning bid, he could try to claim that he did not submit the winning bid even in a case where he rightfully won. To ensure Non-Repudiation, a bidder who submitted a bid must not be able to argue that he did not submit it.

Individual Verifiability. This property establishes that, as any bidder must be able to verify that the bid she sent counts correctly for the result, for any received bid, the bidder needs to be able to verify the correctness of her bid outcome to rejection or acceptance.

Universal Verifiability. This property establishes that any observer may verify that the result of an auction is correct. The property can be divided into Integrity Verifiability and Outcome Verifiability. To ensure Integrity Verifiability, anyone needs to be able to verify that all the accepted bids satisfied the validation criteria by the time the transaction containing it was mined and that the winning bid is one of the accepted bids. To ensure Outcome Verifiability, different stakeholders need to be able to verify: for a loosing bidder, that her bid was inferior to the winning bid, and that the winning bid was sent by another bidder; for the winner, that she actually submitted the winning bid, that the winning amount is correctly computed, that all other bids originated from bidders, and that no bid was modified; and for the seller, that the winning amount is actually the highest submitted bid and the announced winner is correct.

5 Conclusion

In this paper, we presented Auctionity, our English auction protocol based on the blockchain. We explained the details of all the different protocols used. We also defined the security properties that Auctionity should satisfy.

References

[BK18] Erik-Oliver Blass and Florian Kerschbaum. Strain: A secure auction for blockchains. In *23rd European Symposium on Research in Computer Security, ESORICS'18*, LNCS, 2018.

[Bra06] Felix Brandt. How to obtain full privacy in auctions. *International Journal of Information Security*, 5:201–216, 2006.

[CPS07] Brian Curtis, Josef Pieprzyk, and Jan Seruga. An efficient eAuction protocol. In *ARES*, pages 417–421. IEEE Computer Society, 2007.

[DDL15] Jannik Dreier, Jean-Guillaume Dumas, and Pascal Lafourcade. Brandt's fully private auction protocol revisited. *Journal of Computer Security*, 23(5):587–610, 2015.

[DJL13] Jannik Dreier, Hugo Jonker, and Pascal Lafourcade. Defining verifiability in e-auction protocols. *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security - ASIA CCS '13*, 2013.

[DLL13] Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. Formal verification of e-auction protocols. In David Basin and John C. Mitchell, editors, *Principles of Security and Trust*, pages 247–266, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[eba18] ebay. Our company webpage @ONLINE, July 2018.

[JS02] Ari Juels and Michael Szydlo. A two-server, sealed-bid auction protocol. In Matt Blaze, editor, *Financial Cryptography*, volume 2357 of *LNCS*, pages 72–86. Springer, 2002.

[Kri09] Vijay Krishna. *Auction theory*. Academic press, 2009.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[OM01] Kazumasa Omote and Atsuko Miyaji. A practical English auction with one-time registration. In Vijay Varadharajan and Yi Mu, editors, *ACISP*, volume 2119 of *LNCS*, pages 221–234, 2001.

[PBDV02] Kun Peng, Colin Boyd, Ed Dawson, and Kapali Viswanathan. Robust, privacy protecting and publicly verifiable sealed-bid auction. In Robert H. Deng, Sihan Qing, Feng Bao, and Jianying Zhou, editors, *ICICS*, volume 2513 of *LNCS*, pages 147–159. Springer, 2002.

[Sak00] Kazue Sako. An auction protocol which hides bids of losers. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, volume 1751 of *LNCS*, pages 422–432. Springer, 2000.

[Woo18] Gavin Wood. Ethereum: A secure decentralised genereralised transaction ledger. 2018.