Security Analysis of Auctionity: a blockchain based e-auction

Pascal Lafourcade, Mike Nopere, Jérémy Picot, Daniela Pizzuti, Etienne Roudeix

Security Analysis of Auctionity: a blockchain based e-auction

Pascal Lafourcade[1], Mike Nopere[2], Jérémy Picot[2], Daniela Pizzuti[2], and Etienne Roudeix[2]

[1] LIMOS, Université Clermont Auvergne, France
[2] Domraider, Clermont-Ferrand, France

Abstract. Auctions are widely used to sell products between different users. In this paper, we present *Auctionity*, an English e-auction based on blockchain. We describe the different protocols used in Auctionity. We also define the security models and the associated properties. We formally prove some security properties of this protocol using ProVerif.

Keywords: Blockchain, Security, E-auction, ProVerif.

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 [13]. Over the years, several kinds of auctions have been invented. The most well-known is *English auction*, in which the bidder who offers the highest price wins the auction. *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. All bidders simultaneously submit sealed bids and the highest bid wins the auction. *Vickery auction* is a sealed bid auction where the highest bid wins, but the winner only pays the second-highest bid value.

The easy access to the Internet 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 began to take place online, known as *e-auctions*. The e-auctions market is huge, as demonstrated by websites like eBay, which had more than *170* million active buyers in *2018* [10]. E-auction systems often apply cryptographic mechanisms to be secure, but they use a centralized authority to manage transactions between sellers and bidders.

With Bitcoin [16] and Ethereum [21], the blockchain technologies are nowadays a key component of the modern digital world. Essentialy, a blockchain is a distributed and decentralized ledger that does not allow the modification of data stored in it without the consensus of the peers. This property is generally called *immutability* and it clearly is a key feature for auctions based on the blockchain technology. Moreover, several blockchain platforms support smart contracts that can be defined as secure and unstoppable computer programs that represent an agreement to be automatically executed and enforced [1]. Our goal is to design a secure e-auction protocol based on a blockchain.

Contributions: Our main contributions are:

- Design of *Auctionity*, our e-auction system based on the Ethereum blockchain.
- Definition of the relevant security properties for Auctionity.
- Formal analysis of Auctionity security properties.
- Proof of concept of Auctionity implemented using Ethereum.

Auctionity relies on the main Ethereum network (also called mainchain or ELNET) and a private blockchain (also called sidechain or ACNET).

The main properties achieved by Auctionity are: *Highest Price Wins*: the bidder who submitted the highest valid bid is the one who wins the auction. *Non-cancellation*: all bids count to the result and the winning amount is the highest. *Non-repudiation*: a bidder who submitted a bid is not able to argue that she did not submit it. *Individual Verifiability*: a bidder can be sure her bid counts correctly for the result. *Universal Verifiability*: any observer may verify that the result of an auction is fair. *Auction End Voucher Validity*: the winner and the amount of its winning bid on the mainchain corresponds to an existing bid submitted by the winner to the sidechain. *Withdrawal Voucher Validity*: a withdrawal request on the mainchain corresponds to a withdrawal request submitted by the same user, with the same amount, on the sidechain.

Related Work: The closest work to Auctionity is STRAIN (Secure aucTions foR blockchAINs) presented in [3]. In this paper, the authors introduce a sealed bid auction system 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. The aim of Auctionity is different, since its design is of an English auction system where the bids are public. It leads Auctionity to a different paradigm.

Another work that can be considered to have some similarities with Auctionity is the protocol presented by Omote and Miyaji [18]. This work presents an English auction protocol where bids are registered on a bulletin board, which can be compared to the registration of bids done by Auctionity in its private blockchain. The protocol uses two authorities, one that registers bidders (registration manager) and the other that records the bids of each auction (auction manager). At the end of an auction, the auction manager publishes the winning value and the registration key used by the bidder, that corresponds to the bidder's identity stored by the registration manager. As for Auctionity, the bidders signature is always verified and the bids are publicly available. The main differences consist in the fact that this protocol aims to provide privacy for bidders and the protocol does not use blockchain.

Opensea offers a blockchain based auction system. Bids are public. Bidders can bid with any amount, not necessarily higher than the highest bid. Sellers can end the auction at anytime, accepting a bid, not necessarily the highest one. Bidders can cancel their bids at any time. Unlike Auctionity, this system does not guarantee non-cancellation of bids, neither that the highest price wins.

Portio offers a blockchain based auction system. Bidding amounts are deposited on a Portio Ethereum address that is not a smart contract. Therefore, bidders need to fully trust Portio. Unlike Auctionity, this system does not guarantee payment.

There exist several other e-auction protocols among [4,17,12,6,19,20,7]. However, none of them uses blockchain and they aim at providing anonymity mechanisms for the bids. Here again, the aim of Auctionity is clearly different.

Concerning the analysis of e-auction systems security properties, Dreier et al. [9] used the applied pi calculus to generically define auction protocols and the properties of fairness, authentication and privacy. They used ProVerif to analyze the auction protocols by Brandt [4], and by Curtis et al. [6]. In [8] they also defined the property of verifiability and analyzed the same protocols cited above. Our formal analysis of Auctionity follows these works. We also use Proverif in order to anlyse the security of Auctionity.

Outline: In the next section, we describe Auctionity. In Section 3, we present the security model of Auctionity and its formal model made with ProVerif to analyse the security of the protocol. In Section 4, we discuss the perfomance of our proof of concept of Auctionity deployed during 6 months.

2 Description of Auctionity

Auctionity uses Ethereum and is composed of the following principals:

- **Ethereum Live Net (ELNET or Γ)** is the Ethereum public blockchain, where the smart contract (SC) `Deposit`[3] (`D`) is running. It is responsible for holding bidder's deposit, processing withdrawal and payment demands on ELNET.
- **Auctionity Network (ACNET or Ω)** is the private blockchain owned by Auctionity, where the following smart contracts are running:
  - `Treasurer` (`T`) is responsible for holding bidders deposit and processing withdrawal demands on ACNET.
  - `Auction` (`A`) is responsible for processing auctions. However, while the previous smart contracts are instantiated only once, for each auction there is an `Auction` instance, created by the seller.
- **Oracle (O)** is the Auctionity server that transfers information between ELNET and ACNET.
- **Bidder (B)** is a user that participates in an auction on ACNET.
- **Seller (S)** is the user that auctions a product on ACNET.

In Figure 1, the exchanges between these principals in Auctionity are shown. It is important to 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.

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) [11] 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. A hash function, denoted by H(m), and a signature algorithm, denoted

[3] Smart contract names are written in true type.

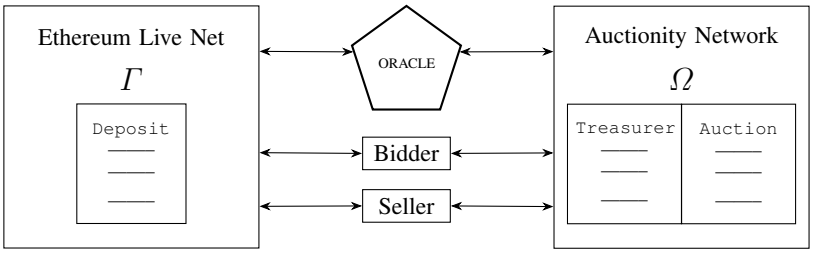

Fig. 1. General Structure of Auctionity.

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

Table 1. Notations, where u is a user.

by $\text{Sig}_{sk_u}(m)$ for signing a message m with the secret key sk_u are also considered. The notation SIG_u represents that a message is sent with its signature, as shown bellow: $\text{SIG}_u(m) = (m \parallel \text{Sig}_{sk_u}(\text{H}(m)))$. In Table 1, all the notations used to describe Auctionity are listed.

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 [21], this counter is called *nonce*.
- $gasPrice_u$: the price to be paid by u per *gas*[4] 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.

[4] Gas is the pricing value required to execute operations on the Ethereum Virtual Machine (EVM).

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

– to_u: it is the recipient of the message sent by u. It corresponds to an address ad_v of a SC or another user v.
– $\text{Sig}_{sk_u}(\text{H}(m))$: the transaction signature, where m contains the previous parameters in addition to specific content of each message.

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

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

Interacting with Γ and Ω a user can bid, withdraw her deposit (amount the bidder has on Ω used as *Payment Guarantee*[6]) and create an auction or end it (to obtain the payment locked on the Ω `Treasurer`. We have four protocols, one for each of those actions.

Create Auction Protocol: it allows a seller to create a new instance of an `Auction`. To create an auction, a seller S sends a signed message to ACNET, with the `Auction` binary code, denoted by "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 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 value of the type *uint24*, in seconds.
– *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 value of the type *uint16*, in seconds.

The parameter *endTime* is stored as *originalEndTime*. We consider *bidTime* as the time when a bid is accepted. The *endTime* is updated only if *bidTime* is greater than the difference between *antiSnippingTriggerPeriod* and *endTime*, which triggers the update of *endTime* to *bidTime* plus *antiSnippingDuration*.

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

1. $\text{S} \rightarrow \Omega$: $\text{SIG}_S(P_S \parallel \text{EVMCode}_{SC} \parallel in)$. 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` with these parameters.
2. $\Omega \rightarrow$ S: ad_{SC}. ACNET sends to S, through ACNET's WebSocket[7], the address of her `Auction` as a confirmation that it has been created. At this point, with the `Auction` instance written on the blockchain, it is not possible to cancel the auction.

[6] 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.

[7] 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.

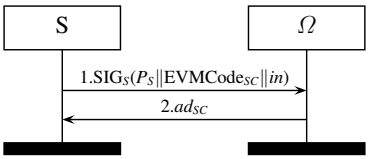

Fig. 2. Create auction protocol.

Close Auction Protocol: it allows a seller to receive the winning amount of her auction. Anyone can request to close an auction. As ACNET cannot emit an event by itself when the auction end time is reached, S is expected to call the function that closes her auction. However, as in the case if only S could do it, S could "freeze" 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` instance, an Auction End Voucher (AEV), which is a bit string issued by the Oracle with the auction results, as the winner's address and winning amount, is added to the `Auction`. 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 Close Auction Protocol is discribed in Figure 3 and works as follows:

1. S $\rightarrow \Omega$: SIG$_S$(P_S $\|$ **close**). Anyone can call the `Auction` function[8] **close** to close an auction, but this call is considered to be 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` function **close**, ACNET emits the Ethereum event[9], LogAuctionClosed, which indicates that this `Auction` instance received a valid close auction request.
2. $\Omega \rightarrow$ O: LogAuctionClosed. The Oracle O, listening to ACNET's WebSocket, gets the information of the Ethereum event, LogAuctionClosed, triggered by the `Auction` function **close**, of the `Auction` instance created by S on ACNET.
3. O $\rightarrow \Omega$: SIG$_O$(P_O $\|$ **set** $\|$ 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` function **set** with the parameter AEV_S, ACNET emits the Ethereum event LogAEVSet, which indicates that AEV_S was set to S's `Auction` instance.
4. $\Omega \rightarrow$ S: LogAEVSet. S, listening to ACNET's WebSocket, gets the information of the Ethereum event, LogAEVSet, triggered by the `Auction` function **set**, of her `Auction` instance. Then, S gets her AEV_S.
5. S $\rightarrow \Gamma$: SIG$_S$(P_S $\|$ **submit** $\|$ 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

[8] SC function names are written in bold.

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

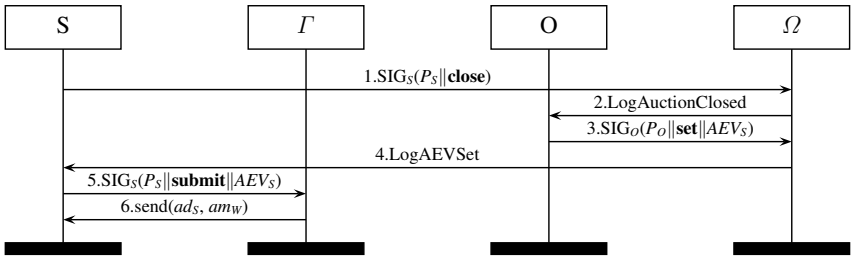

Fig. 3. Close auction protocol.

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: send(ad_S, am_W). The ELNET Deposit SC, provided with a valid AEV_S, uses the Solidity[10] function **send** to send the winning amount am_W, to S's address, denoted by ad_S.

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 Bid Protocol is described in Figure 4 and is composed of depositing (1 to 4) and bidding (5 and 6). It works as follows:

1. B $\rightarrow \Gamma$: SIG$_B$(P_B $\|$ **deposit** $\|$ 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 $\rightarrow \Omega$: SIG$_O$(P_O $\|$ **add** $\|$ ad_B $\|$ 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` 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`.
4. $\Omega \rightarrow$ B: LogDepositAdded. B, listening to ACNET's WebSocket, gets the information of the Ethereum event, LogDepositAdded, triggered by the `Treasurer` function **add**. Then, B gets the current balance of her deposit.
5. B $\rightarrow \Omega$: SIG$_B$(P_B $\|$ **bid** $\|$ 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` function **bid**, with the parameter am_B, ACNET emits the

[10] Solidity is a programming language for writing Ethereum smart contracts.

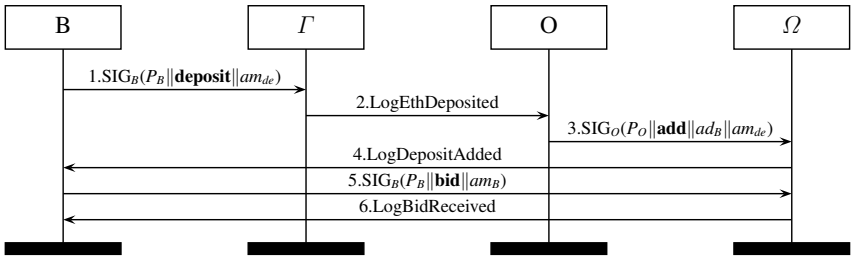

Fig. 4. Bid protocol.

Ethereum event, LogBidReceived, which indicates that B's bid of amount am_B on an `Auction` 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` instance ad_{SC}. Then, B gets the status of her bid (accepted or rejected).

Each `Auction` instance stores the address of the leader, denoted by ad_L, as leader is denoted by L, and its bid amount. The `Auction` is also responsible for checking the validity of each bid. In order to be valid, a bid must respect the following criteria:

- Sig_{sk_B} matches ad_B: the transaction is signed with a secret key that corresponds to the bidder's address (ad_B).
- $ts_{start} \leq ts_{cur} \leq ts_{end}$: the block timestamp ($ts_{block}$) is bigger or equal to the *startTime* (ts_{start}) of the auction and smaller or equal to the *endTime* (ts_{end}) of the auction.
- $ad_B \neq ad_S$: the bidder's address (ad_B) is different from the sellers (ad_S).
- $am_B > am_{min}$: the *bid amount* (am_B) is higher than the *minimumAmount* (am_{min}) or than the *leaderAmount* (am_L) and is it a multiple of the *bidIncrement* (am_{inc}) set by the seller.
- $de_B \geq am_B$: the bidder's deposit (de_B) on ACNET is higher or equal the payment guarantee, which is equal to the bid amount, required by the auction.

At the end of the auction, the bidder's address stored in the `Auction` variable ad_L is the winner, and the variable am_L is the winning amount.

Withdraw Deposit Protocol: it allows a bidder to get her deposit back to her account on ELNET. Another action made by a user as a bidder is to withdraw her deposit, which corresponds to having the Ether that she previously deposited but did not use, sent back to her address on ELNET. When a bidder wants to get 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, which 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 denoted am_{wi}.

The Withdrawal Protocol is described in Figure 5 and works as follows:

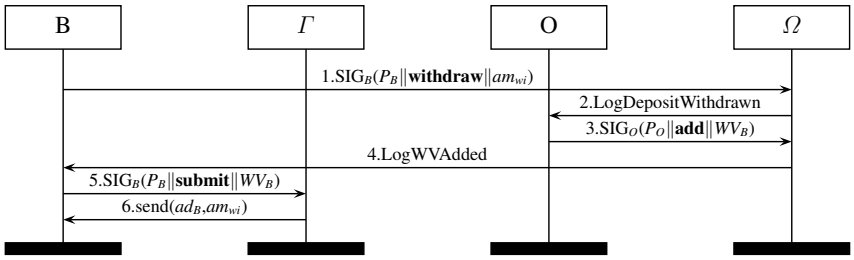

Fig. 5. Withdrawal deposit protocol.

1. B $\rightarrow \Omega$: 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` function **withdraw**, with the parameter am_{wi}, ACNET emits the event, LogDepositWithdrawn, which indicates that the `Treasurer` 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` function **withdraw**.
3. O $\rightarrow \Omega$: SIG$_O$(P_O || **add** || 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` 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` function **add**. Then, B gets her WV_B.
5. B $\rightarrow \Gamma$: SIG$_B$(P_B || **submit** || 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` 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: send(ad_B, am_{wi}). The ELNET `Deposit`, provided with a valid WV_B, uses the SC function **send** to send the withdrawal amount am_{wi}, to B's address, denoted by ad_B.

3 Security Models

We model Auctionity in applied pi calculus in order to use the ProVerif [2] tool to analyze its security properties. ProVerif provides automatic analysis of cryptographic protocols in the symbolic Dolev-Yao model for unbounded number of sessions. The tool can handle many different cryptographic primitives, including encryption, signatures and hash functions. We start by formally defining an Auction End Voucher (*AEV*).

Definition 1. *An* Auction End Voucher *is a tuple (pk_W, am_W, $creationProof_S$, $bidProof_W$, σ) where pk_W is the public key of the auction winning bidder, am_W is the value of the winning bid, $creationProof_S$ is the creation transaction submitted by S to ACNET, $bidProof_W$ is the bid transaction submitted by the bidder W to ACNET and σ is the Oracle's signature of the AEV.*

An Auction End Voucher is requested by a seller to her `Auction` after the auction end time is reached. Next, the `Auction` verifies if pk_L has am_L blocked on the `Treasurer` for the auction ad_A. If so, an Ethereum event is emitted by the `Auction`, informing that the public key of the leader pk_L is the winner pk_W of the auction. The Oracle gets this event, verifies the validity of the request and sends a signed AEV_S to the Auction SC. The seller retrieves the AEV_S and submits it to the `Deposit` on ELNET, that verifies its validity, sends am_W to the seller's public key and update the balance of pk_W on ELNET. We now define a Withdrawal Voucher (*WV*).

Definition 2. *A* Withdrawal Voucher *is defined by a tuple (pk_B, am_B, $withdrawalProof_B$, σ) where pk_B is the public key of the user who requested the withdrawal, am_B is the value the user requested to withdraw, $withdrawalProof_B$ is the withdrawal request transaction submitted by B, and σ is the Oracle's signature of the other components of the WV.*

A Withdrawal Voucher is requested by a bidder to the `Treasurer` to get her money back on ELNET. Next, the `Treasurer` verifies if pk_B has am_B as balance. If so, an event is emitted by the `Treasurer`, informing that the public key of the bidder pk_B requested to withdraw the amount am_B. The Oracle verifies the validity of the request and sends a signed WV_B to the `Treasurer`. The bidder retrieves the WV_B and submits it to the `Deposit` on ELNET, that verifies its the validity, sends am_B to the bidder's public key and updates the balance of pk_B on ELNET.

For each security propertiy, we prove either manually or with ProVerif the corresponding theorems[11].

3.1 Highest Price Wins

This property presented in [9] establishes that the bidder who submitted the highest valid bid is the one who wins the auction. As Auctionity is based on blockchain, this property can be accomplished under the statement that the bidder who submitted the highest **accepted** bid is the one who wins the auction. So, in the model, when a bid is accepted, the address of the `Auction` to which it was submitted, the public key of the bidder and her bid amount are inserted in the table called *auction*. When ACNET receives a bid, it verifies if there is an entry on the ProVerif table of a bid with an amount greater or equal the bid received. If there is not, the bid is accepted, if there is, the bid is rejected and the event bidRejected is emitted.

Theorem 1. *Auctionity ensures the property* Highest Price Wins (HPW) *if for any auction process AP there is no AP'[$B\sigma_{pk_p}\sigma_{am_p}|(B\sigma_{pk_q}\sigma_{am_q})$] where am_p is an accepted bid with the highest amount, and there is a trace containing the event won for bidder pk_q that had her bid rejected.*

[11] The detailed proofs are avaible in [14] and the Proverif code in [15]

3.2 Authentication

We consider two authentication properties defined in [9]: Non-cancellation and Non-repudiation.

Non-cancellation. In an English auction each valid bid must be greater than the current lastest bid, therefore, the bidder who submitted the last accepted bid must win the auction. The Non-cancellation property aims at preventing the annulment of bids. To model this in ProVerif, we introduce the event bidAccepted(*ad*, *pk*,*am*) that corresponds to the acceptance of a bid. All the accepted bids are stored as events and update the variables that store the current leader and current winning amount, modeled by the event won(*ad*, *pk*, *am*). This leads us to the following formal property of Non-cancellation.

Theorem 2. *Auctionity ensures* Non-cancellation (NC) *if for any auction process AP which contains a bidder* $B(\sigma_{ad_p}\sigma_{am_p})$ *who submits the highest accepted bid, i.e.,* $\forall\ p \neq q$*:* $am_p > am_q$*, there is no trace containing the events bidAccepted(ad, pk, am) and won(ad, pk, am) for another bid with* $am \neq am_p$*.*

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

Theorem 3. *Auctionity ensures* Non-repudiation (NR) *if for every auction process AP on every possible execution trace the event won(ad, pk, am) is preceded by a corresponding event bid(ad, pk, am).*

3.3 Verifiability.

Verifiability is one of the key properties for Auctionity, as each participant needs to trust that the system operates correctly. Verifiability can be Individual or Universal.

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

Theorem 4. *Auctionity ensures* Individual Verifiability *if for any* $bidReceived(pk_B, am_B)$*, the bidder B can verify the correctness of her bid outcome to* $bidRejected(pk_B, am_B)$ *or* $bidAccepted(pk_B, am_B)$*.*

Universal Verifiability: This property defined in [9] establishes that any observer may verify that the result of an auction is correct, with the public information available. 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.

Outcome Verifiability depends on the different participants:

– For a loosing bidder, she needs to verify that her bid was inferior to the winning bid, and that the winning bid was sent by another bidder.
– For the winner, she needs to verify 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.
– For the seller, she needs to verify that the winning amount is actually the highest submitted bid and the announced winner is correct.

Theorem 5. *Auctionity ensures* Universal Verifiability *if there exist Verification Tests* IV_b, IV_w, OV_l, OV_w, OV_s *respecting the following soundness conditions:*

1. *Integrity Verifiability (IV):*
   – *Anyone can verify that all the accepted bids are valid.*
   $IV_b = true \Rightarrow \forall accBids(pk_B, am_B)$*:*
   - Sig_{sk_B} *matches* ad_B,
   - $ts_{start} \leq ts_{cur} \leq ts_{end}$,
   - $pk_B \neq pk_S$,
   - $am_B > am_{min}$ *OR* $am_B >$ *previous* am_L,
   - $de_B \geq am_B$,

   $IV_w = true \Rightarrow winBid \in bidAccepted(L)$
2. *Outcome Verifiability (OV):*
   – *A loosing bidder can verify that her bid was not the winning bid:*
   $OV_l = true \Rightarrow myBid \neq win(getAmount(L))$.
   – *A winning bidder can verify that her bid was the winning bid:*
   $OV_w = true \Rightarrow myBid = win(getAmount(L))$.
   – *The seller can verify that the winning bid is actually the highest submitted bid:*
   $OV_s = true \Rightarrow winBid = win(getAmount(L))$.

   as well as the following completeness condition:

– *If all participants follow the protocol correctly, the above tests succeed (i.e., the implications hold in the opposite direction,* ($\Leftarrow$*, as well).*
where - with abuse of notation - getAmount(L) is written for getAmount(L[1]), ..., getAmount(L[n]).

Informally, we have that:

– Any bidder can verify that her bid was accepted by checking the event emitted by `Auction` to which the bid was made. An accepted bid triggers an event *bidAccepted* and updates the pk_L and the am_L variables.
– Loosing bidders: any loosing bidder can verify that the winning bid is superior to her bid, as both bids are publicly available on the blockchain. Also, she can be convinced that the winning bid was submitted by another bidder due to the message signature registered as part of the bid transaction on the blockchain.
– Winning bidder: the winner can check that she submitted the winning bid, as well as the bid amount correctness, by verifying if the signature of the bid transaction hash was signed with her secret key. Besides, she can verify that all the other bids were originated by real users, by checking their signature and that they lost, by checking their amount. It is also possible to verify that the seller did not bid, or at least, that her public key is not linked to any bid because if it is, the `Auction` rejects the bid and registers it as an event.

– A seller can create another public key in order to bid in her own auction and it can not be prevented, which is the same for any online auction protocol where the seller can create different accounts or even to physical auctions, to which the seller can send someone else to bid in her behalf aiming to raise the final amount. To prevent this, the payment guarantee discourages the seller to apply this tactic. Even if the money will come back to her public key in the case she wins, she will still have her money blocked for some time plus the cost on ELNET for deposit and withdraw operations.
– Seller: the seller is interested in verifying that the winner's public key and the final amount are correct. In order to do that, she can see all the history of bids made to her `Auction` to check that the highest amount is the final amount and that the winner is the public key linked to this highest amount bid.

3.4 Validity

Auctionity is based in two blockchain networks: ACNET and ELNET. They do not have a direct communication channel and use the Oracle to exchange information. Validity proofs of the content issued by the Oracle are covered by the two following properties.

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 pk_W and the amount of her winning bid am_W correspond to a bid signed by W to the `Auction` instance to which the AEV_S was issued.

Theorem 6. *Auctionity ensures* Auction End Voucher Validity *if for any Auction End Voucher submitted on ELNET:*
– *The issuance of the AEV was made after the end time of the corresponding auction:* $(ev_t) = true \Rightarrow ts_{end} < ts_{AEV}$.
– *The seller's public key corresponds to the public key that created the auction:* $(ev_a) = true \Rightarrow pk_S$ = *creator of* $AEV(ad_{SC})$.
– *The winner's public key and the amount of its winning bid are equal to the ones on the* `Auction`: $(ev_w) = true \Rightarrow won(pk, am) = AEV(won(pk, am))$.

Withdrawal Voucher Validity. As ELNET is not aware of the transactions processed on ACNET, the WV_B is also used to carry information to ELNET, in this case, the withdrawal requests. 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` that the bidder's public key corresponds to the public key that signed the Withdrawal Voucher request and this request is of the same amount as the one contained in the WV_B.

Theorem 7. *Auctionity ensures* Withdrawal Voucher Authenticity *if for any Withdrawal Voucher submitted on ELNET:*

			Payer		
			ACNET	ELNET	
Protocol	Net	Function and Caller	Auctionity	Bidder	Seller
Create auction	Ω	Deploy A 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	0	85,476	0
	Γ	another **deposit** by same B	0	30,421	0
	Ω	**add** by O	107,839	0	0
	Ω	1st **bid** by 1st B	167,672	0	0
	Ω	1st **bid** by another B	136,081	0	0
	Ω	another **bid** by 1st B	80,464	0	0
	Ω	another **bid** by another B	67,994	0	0
Withdrawal	Ω	**withdraw** by any B	30,160	0	0
	Ω	**add** by O	247,071	0	0
	Γ	**submit** by any B	0	111,736	0

Table 2. Cost of SC function calls in gas.

- *The bidder's public key corresponds to the public key that signed the Withdrawal Voucher request.* $(wv_a) = true \Rightarrow pk_B = pk_{(withdrawal \parallel z \parallel Sig_B)}$
- *The bidder public key has the same value withdrawn in both ELNET and ACNET after the application of the Withdrawal Voucher.* $(wv_v) = true \Rightarrow (\Gamma deposits.am_B - am_{WV}) = \Omega\ deposits.am_B$

4 Experimental Results

The four protocols of Auctionity were tested over a period of 6 months. We describe how the gas is used in Auctionity, what is the block time duration and some statistics of this experiment.

Gas cost: Gas is a term used in Ethereum network to denote a fee for computations on the EVM. Its complete usage and calculation is presented in the Ethereum Yellow Paper [21]. The cost of each action, measured with the Geth function eth_estimateGas, is shown in Table 2. B_1 is the first bidder to bid on an auction and B_n is any other bidder.

In the Auctionity system, users only pay for actions that take place on ELNET, which means, deposits and withdrawals. The *gasPrice* depends of the usage of ELNET. *transactionCost* = *gasUsed* * (*gasPrice* * 1 *G*wei)[12]

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 is about 8,000,000.

[12] GWei is equal to 10^9 Wei and 10^{-9} Ether.

Protocol	Properties
Create Auction	-
Bid	**HPW**, **NC**, **NR**, IV, UV
End Auction	AEV Validity
Withdraw Deposit	WV Validity

Table 3. Protocols and properties (properties in bold proven with ProVerif).

Block Time: The time period between 2 blocks is 1 second on ACNET and 14 seconds on ELNET. The Oracle waits 10 blocks to confirm a deposit made on ELNET, following the proof made by Vitalik Buterin [5]. Therefore, a deposit takes place in 140 seconds after being included in a block. A user can bid as many times as she wants, as long as she has a sufficient deposit amount for the bids she wants to submit. Considering that the deposit can be withdrawn at any time, and in order to avoid having to wait the deposit validation time before bidding, it is expected of bidders to deposit a sufficient amount for the auctions they intend to participate in and withdraw it whenever needed.

Statistics: Firstly, we have tested, with some automated scripts, all protocols and their chronological auction sequence: Create Auction Protocol, Bid Protocols and Close Auction Protocol. These scripts were also used to estimate the growing load of Auctionity blockchain. The following data includes the different script tests, and some real human behaviours using the Auctionity website. This represents a total of 462 users. The users sent 38,000 deposits on ELNET and got back 35,000 withdraws on ACNET. The difference can be explained by the fact that a user can deposit 4 ETH and after 6 ETH, which makes two deposits, and later withdraw 10 ETH at once, which makes only one withdraw. They have also created 23,000 auctions, and the same number of closed auctions. The average auction duration was of 5 days and the maximum auction duration was of 21 days. The total number of bids was equal to 144,000 and the number of auctions with no bids was equal to 377.

5 Conclusion

In this paper we presented Auctionity, our English auction protocol based on the blockchain. We presented the details of the protocols used, defined security models and properties that Auctionity should satisify and provide a formal analysis and experimental results. The protocols and their properties are summerized in Table 3.

Further developments will be made to improve decentralization of the Oracle and sidechain while keeping or improving throughput, and allowing usage of tokens from other blockchains than Ethereum.

References

1. I. Bashir. *Mastering Blockchain: Distributed ledger technology, decentralization, and smart contracts explained*. Packt Publishing Ltd, 2018.

2. B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre. Proverif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial. 2018.
3. E.-O. Blass and F. Kerschbaum. Strain: A secure auction for blockchains. In *23rd European Symposium on Research in Computer Security, ESORICS'18*, LNCS, 2018.
4. F. Brandt. How to obtain full privacy in auctions. *International Journal of Information Security*, 5:201–216, 2006.
5. V. Buterin. On slow and fast block times @ONLINE, July 2015.
6. B. Curtis, J. Pieprzyk, and J. Seruga. An efficient eAuction protocol. In *ARES*, pages 417–421. IEEE Computer Society, 2007.
7. J. Dreier, J. Dumas, and P. Lafourcade. Brandt's fully private auction protocol revisited. *Journal of Computer Security*, 23(5):587–610, 2015.
8. J. Dreier, H. Jonker, and P. Lafourcade. Defining verifiability in e-auction protocols. *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security - ASIA CCS '13*, 2013.
9. J. Dreier, P. Lafourcade, and Y. Lakhnech. Formal verification of e-auction protocols. In D. Basin and J. C. Mitchell, editors, *Principles of Security and Trust*, pages 247–266, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
10. ebay. Our company webpage @ONLINE, July 2018.
11. D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.
12. A. Juels and M. Szydlo. A two-server, sealed-bid auction protocol. In M. Blaze, editor, *Financial Cryptography*, volume 2357 of *LNCS*, pages 72–86. Springer, 2002.
13. V. Krishna. *Auction theory*. Academic press, 2009.
14. P. Lafourcade, J. Picot, D. Pizzuli, M. Nopere, and E. Roudeix. Formal definition of the auctionity protocol and its security properties. Technical report, LIMOS, 2018. `http://sancy.univ-bpclermont.fr/~lafourcade/technical.pdf`.
15. P. Lafourcade, J. Picot, D. Pizzuli, M. Nopere, and E. Roudeix, 2019. `http://sancy.univ-bpclermont.fr/~lafourcade/auctionity.tar`.
16. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.
17. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, pages 129–139, 1999.
18. K. Omote and A. Miyaji. A practical English auction with one-time registration. In V. Varadharajan and Y. Mu, editors, *ACISP*, volume 2119 of *LNCS*, pages 221–234, 2001.
19. K. Peng, C. Boyd, E. Dawson, and K. Viswanathan. Robust, privacy protecting and publicly verifiable sealed-bid auction. In R. H. Deng, S. Qing, F. Bao, and J. Zhou, editors, *ICICS*, volume 2513 of *LNCS*, pages 147–159. Springer, 2002.
20. K. Sako. An auction protocol which hides bids of losers. In H. Imai and Y. Zheng, editors, *Public Key Cryptography*, volume 1751 of *LNCS*, pages 422–432. Springer, 2000.
21. G. Wood. Ethereum: A secure decentralised genereralised transaction ledger. 2018.